Elektra

0.11.0

Generated by Doxygen 1.9.1

Chapter 1

Elektra Initiative Overview

Elektra serves as a universal and secure framework to access configuration parameters in a global, hierarchical key database and provides a mature, consistent and easily comprehensible API. Its modularity effectively avoids code duplication across applications and tools regarding configuration tasks. Elektra abstracts from cross-platform-related issues and allows applications to be aware of other applications' configurations, leveraging easy application integration.

See the readme for more introduction. See the glossary for the used terminology.

1.1 API Docu

This document's main goal is to describe the API. It covers:

- external C-API (see Modules above), which are the essential core parts
- C++-API (see Data Structures above) from a direct binding to high-level functionality, such as mounting functionality
- plugins API, see Plugins
- all other documentation of Elektra (see Related Pages next to Main Page)

On the one hand it gives an overview and an introduction for developers using Elektra, on the other hand it gives an informal description what methods must and may provide to allow an alternative implementation of the API.

The latest released version (for stable releases) of this document can be found at `https://doc.libelektra.org/api/latest/html`

The Git master version of this document can be found at `https://doc.libelektra.org/api/master/html`

Important: On GitHub links to API functions are broken, so it is recommended that you continue reading in one of these links above.

1.2 Using the Elektra Library

A C or C++ source file that wants to use Elektra should include:

```
#include <kdb.h>
```

To link an executable with the Elektra library, one way is to use the `pkg-config` tool:

```
gcc -o application `pkg-config --cflags --libs elektra` application.c
```

Another way is to use CMake:

```
find_package(Elektra REQUIRED)
include_directories (${ELEKTRA_INCLUDE_DIR})
target_link_libraries (application ${ELEKTRA_LIBRARIES})
```

Read about compiling elektra.

1.2.1 Tutorials

- Application Integration
- Compilation Variants
- Export
- Import
- Merge
- Namespaces
- Plugins
- Java Plugins

List of all available Plugins and get started by developing your own plugins Plugins.

1.3 Elektra API

The API was written in pure C because Elektra was designed to be useful even for the most basic system programs.

The API follows an object-oriented design, and there are 3 main classes as shown by the figure:

Some general things you can do with each class are:

KDB (Key Database)

- Open and Close the Key Database
- Get and Set KeySet in the Key Database
- See class documentation for more

Key

- Create and Delete
- Get and Set key the name
- Get and Set string or binary values
- Get and Set Metadata
- See class documentation for more

KeySet

- Create and Delete
- Append a single key or an entire KeySet
- Lookup keys
- Pop the last key, a key by name, or every key
- Get a key at a specific position
- Get the number of elements in a KeySet
- See class documentation for more

More background information about the classes

1.4 Namespaces

There are 5 trees (=namespaces) of keys: `spec`, `proc`, `dir`, `user` and `system` that are all unified (in the given order) in one cascading tree starting with `/`.

The cascading tree is the logical tree to be used in applications. The other trees are the physical ones that stem from configuration sources. When using cascading key the best key will be searched at run-time, which appears like a tree on its own. See cascading in the documentation of ksLookupByName() on how the selection of keys works.

- The `spec` tree

  This tree specifies how the lookup should take place and also allows us to define defaults or document a key. The metadata of a key contains this information:

  - `override/#`: use these keys *in favor* of the key itself (note that `#` is the syntax for arrays, e.g. `#0` for the first element, `#10` for the 11th and so on)
  - `namespace/#`: instead of using all namespaces in the predefined order, one can specify which namespaces should be searched in which order
  - `fallback/#`: when no key was found in any of the (specified) namespaces the `fallback`-keys will be searched
  - `default`: this value will be used if nothing else was found

- The `proc` tree

  Is the only read-only tree. The configuration does not stem from the KDB (Key Database), but any other source, e.g. command-line arguments or environment.

- The `dir` tree

  Allows us to have a per-directory overwrite of configuration files, e.g. for project specific settings.

- The `user` tree

  Used to store user-specific configurations, like the personal settings of a user to certain programs. The user subtree will always be favored if present (except for security concerns the user subtree may not be considered).

- The `system` tree

  It is provided to store system-wide configuration keys, that is, the last fallback for applications but the only resort for daemons and system services.

Read more about namespaces and a tutorial for namespaces.

1.5 Rules for Key Names

When using Elektra to store your application's configuration and state, please keep in mind the following rules:

- You are not allowed to create keys right under the root. They are reserved for more generic purposes.
- The keys for your application, called say *myapp*, should be created under `/sw/org/myapp/#0/current`
  - sw is for software
  - org is the organization. For uniqueness a full reverse url encoded with '/' instead of '.' is useful.
  - `#0` is the major version of the configuration
  - current is the default configuration profile.
  - That means you just need to kdbGet() `/sw/org/myapp/#0/profile` and then ksLookupByName() in `/sw/org/myapp/#0/profile/key` where profile is from command-line arguments and defaults to current.

Read more about key names

1.6 Backend Overview

The core of Elektra does not store configuration itself to the hard disk. Instead this work is delegated to backends.

If you want to develop a backend, you should already have some experience with Elektra from the user point of view. You should be familiar with the data structures: Key and KeySet Then you can start reading about Backends that are composed out of Plugins. To get started with writing plugins, first read our plugin tutorial and then lookup details in the API description in Plugins.

Read more about mounting

1.7 See Also

- See elektra-glossary(7)
- More information about elektra-backends(7)
- More information about plugins-framework

Chapter 2

README

This folder contains the core libraries of Elektra.

> **Note:** Some information in this document is outdated and will change before the release of Elektra 1.0.

2.1 Content

Since 0.8.15 this folder contains multiple libraries:

2.1.1 Libelektra

`libelektra.so`

Libelektra is now only a stub for legacy applications. It basically only links all previous libraries together. It should *not* be used for new applications or plugins.

2.1.2 Libfull

`libelektra-full.so`

Contains all sources of Elektra linked to together in one large library. Useful if you do not want dynamically loaded plugins. Should only be used on embedded systems (where whole application stack is done by you) and for tests.

2.1.3 Libstatic

`libelektra-static.so`

Contains all sources of Elektra linked to together in one large library. Useful if you need your application to be linked statically against Elektra. Should only be used on embedded systems (where whole application stack is done by you) and for tests.

2.1.4 Libkdb

`libelektra-kdb.so`
`<kdb.h> (kdb*)`

Contains `kdb*` symbols and applications should link against it.

Chapter 3

High-Level API

3.1 Introduction

The goal of the high-level API is to increase the usability of libelektra for developers who want to integrate Elektra into their applications. Applications usually do not want to use low-level APIs. `KDB` and `KeySet` are useful for plugins and to implement APIs, but not to be directly used in applications. The high-level API should be extremely easy to get started with and at the same time it should be hard to use it in a wrong way. This tutorial gives an introduction for developers who want to elektrify their application using the high-level API.

The API supports all CORBA Basic Data Types, except for `wchar`, as well as the `string` type (see also Data Types below).

3.2 Setup

First you have to add `elektra-highlevel`, `elektra-kdb` and `elektra-ease` to the linked libraries of your application. To be able to use it in your source file, just include the main header with `#include <elektra.h>` at the top of your file.

3.3 Quickstart

The quickest way to get started is to adapt the following piece of code to your needs:

```
ElektraError * error = NULL;
Elektra * elektra = elektraOpen ("/sw/org/myapp/#0/current", NULL, NULL, &error);
if (elektra == NULL)
{
        printf ("An error occurred: %s", elektraErrorDescription (error));
        elektraErrorReset (&error);
        return -1;
}
kdb_long_t mylong = elektraGetLong (elektra, "mylong");
printf ("got long " ELEKTRA_LONG_F "\n", mylong);
elektraSetBoolean (elektra, "mybool", true, &error);
if (error != NULL)
{
        printf ("An error occurred: %s", elektraErrorDescription (error));
        elektraErrorReset (&error);
}
elektraClose (elektra);
```

To run the application, the configuration should be specified:

```
sudo kdb meta-set /sw/org/myapp/#0/current/mylong type long
sudo kdb meta-set /sw/org/myapp/#0/current/mylong default 5
```

The getter and setter functions follow the simple naming scheme `elektra(Get/Set)`[Type]. Additionally for each one there is a variant to access array elements with the suffix `ArrayElement`. For more information see below.

You can find a complete example at the end of this document and here.

3.4 Core Concepts

3.4.1 Metadata and Specification

In Elektra keys may have attached metadata describing additional properties of the key. By using Elektra'snamespaces" and @ref doc_tutorials_cascading_md "cascading keys" it is also possible to have a full specification of your applications configuration.

This specification should be placed into the `spec` namespace. From there the high-level API and Elektra's plugins will access it. A specification for use with the high-level API has to **define at least the `default` and the `type` metadata for each key the application is going to use**. The `default` metakey simply defines which value will be returned, if the user didn't set a value. `type` defines the data type of key. For more information on data types `see below`.

The API also supports passing a `KeySet` to `elektraOpen` that contains the specification. This is, however, not recommended for general use and is mainly useful for debugging and testing purposes.

3.4.2 Struct <tt>Elektra</tt>

`Elektra` is the handle you use to access the underlying KDB (hierarchical key database) that stores the configuration key-value pairs. All key-value read and write operations expect this handle to be passed as in as a parameter. To create the handle, you simply write:

```
ElektraError * error = NULL;
Elektra * elektra = elektraOpen ("/sw/org/myapp/#0/current", NULL, NULL, &error);
```

Please replace `"/sw/org/myapp/#0/current"` with an appropriate value for your application (see here for more information). You can use the parameter `defaults` to pass a `KeySet` containing `Key`s with default values to the `Elektra` instance.

The `ElektraError` can be used to check for initialization errors. You can detect initialization errors by comparing the result of `elektraOpen` to NULL:

```
if (elektra == NULL)
{
  // handle the error, e.g. print description
  elektraErrorReset(&error);
}
```

If an error occurred, you must call `elektraErrorReset` before using the same error pointer in any other function calls (e.g. `elektraSet*` calls). It is also safe to call `elektraErrorReset`, if no error occurred.

In order to give Elektra the chance to clean up all its allocated resources, you have to close your instance, when you are done using it, by calling:

```
elektraClose (elektra);
```

NOTE: Elektra is only thread-safe when you use one handle per thread or protect your handle. If you have multiple threads accessing key-values, create a separate handle for each thread to avoid concurrency issues.

3.4.3 Struct <tt>ElektraError</tt>

The library is designed to shield developers from the many errors one can encounter when using KDB directly. However it is not possible to hide all those issues. As with every library, things can go wrong and there needs to be a way to react to errors once they have occurred at runtime. Therefore the high-level API introduces a struct called `ElektraError`, which encapsulates all information necessary for the developer to handle runtime-errors appropriately in the application.

Functions that can produce errors, despite correct use of the API, accept an `ElektraError` pointer as parameter, for example:

```
Elektra * elektraOpen (const char * application, KeySet * defaults, KeySet * contract, ElektraError **
       error);
```

In most cases you'll want to set the error variable to `NULL` before passing it to the function. You can do this either by declaring and initializing a new variable with `ElektraError * error = NULL` or by reusing an already existing error variable by resetting it with `elektraErrorReset (&error)`.

Notice, that you should always check if an error occurred by comparing it to `NULL` after the function call.

If an error happened, it is often useful to show an error message to the user. A description of what went wrong is provided in the `ElektraError` struct and can be accessed using `elektraErrorDescription (error)`. Additionally the error code can be accessed through `elektraErrorCode (error)`. NOTE: The error API is still a work in progress, so more functions will likely be added in the future.

To avoid leakage of memory, you have to call `elektraErrorReset (&error)` (ideally as soon as you are finished resolving the error):

```
ElektraError * error = NULL;
// Call a function and pass the error variable as an argument.
// ...
if (error != NULL)
{
  // An error occurred, do something about it.
  // ...
  elektraErrorReset (&error);
}
```

3.4.4 Configuration

Currently there is only one way to configure an `Elektra` instance:

```
void elektraFatalErrorHandler (Elektra * elektra, ElektraErrorHandler fatalErrorHandler);
```

This allows you to set the callback called by Elektra, when a fatal error occurs. Technically a fatal error could occur at any time, but the most common use case for this callback is inside of functions that do not take a separate `ElektraError` argument. For example, this function will be called, when any of the getter-functions is called on a non-existent key which is not part of any specification, and therefore has no specified default value.

If you provide your own callback, it must interrupt the thread of execution in some way (e.g. by calling `exit()` or throwing an exception in C++). It *must not* return to the calling function.

The handler will also be called whenever you pass `NULL` where a function expects an `ElektraError **`. In this case the error code will be `ELEKTRA_ERROR_CODE_NULL_ERROR`.

The default callback simply logs the error with `ELEKTRA_LOG_DEBUG` and then calls `exit()` with exit code `EXIT_FAILURE` It is expected that you implement your own callback, so that you get proper error message logged in your applications preferred format. Using the default callback is only viable for very simple applications, because you won't get any indication as to which key caused the error (unless you compiled Elektra with debug logging enabled).

3.5 Data Types

The API determines the data type of a given key, by reading its `type` metadata. The API supports the following types, which are taken from the CORBA specification:

- **String**: a string of characters, represented by `string` in metadata
- **Boolean**: a boolean value `true` or `false`, represented by `boolean` in metadata, in the KDB the raw value `"1"` is regarded as true, `"0"` regarded as false and any other value is an error
- **Char**: a single character, represented by `char` in metadata
- **Octet**: a single byte, represented by `octet` in metadata
- **(Unsigned) Short**: a 16-bit (unsigned) integer, represented by `short` (`unsigned_short`) in metadata
- **(Unsigned) Long**: a 32-bit (unsigned) integer, represented by `long` (`unsigned_long`) in metadata
- **(Unsigned) Long Long**: a 64-bit (unsigned) integer, represented by `long_long` (`unsigned_long_long`) in metadata
- **Float**: whatever your compiler treats as `float`, probably IEEE-754 single-precision, represented by `float` in metadata
- **Double**: whatever your compiler treats as `double`, probably IEEE-754 double-precision, represented by `double` in metadata
- **Long Double**: whatever your compiler treats as `long double`, not always available, represented by `long_double` in metadata

The API contains one header that is not automatically included from `elektra.h`. You can use it with `#include <elektra/conversion.h>`. The header provides the functions Elektra uses to convert your configuration values to and from strings. In most cases, you won't need to use these functions directly, but they might still be useful sometimes (e.g. in combination with `elektraGetType` and `elektraGetRawString`). We also provide a `KDB_TPYE_*` constant for each of the types listed above. Again, most users won't use these but, if you ever do need to use the raw type metadata using constants enables code completion and protects against typos.

There is also the type `enum` with constant `KDB_TYPE_ENUM`. It is only supported via the code-generation API.

3.5.0.0.1 Note about Floating Point Types

We enforce a few minimum properties for floating point types. They are taken from the IEE-754 specification and are:

- For `float`: 32 bits, binary, 24 mantissa digits and exponent range of at least -125 to 128
- For `double`: 64 bits, binary, 53 mantissa digits and exponent range of at least -1021 to 1024
- For `long double`: at least 80 bits, binary, at least 64 mantissa digits and exponent range of at least -2^14 + 3 to 2^14

Additionally for C++ compilers we use a `static_assert` that will fail if `std::numeric_limits<T>::is_iec559` is `false` when `T` is any of `float`, `double` or `long double`.

While these checks won't ensure actual IEEE-754 arithmetic, they will at least ensure all values can be represented correctly.

3.6 Reading and Writing Values

3.6.1 Key Names

When calling `elektraOpen` you pass the parent key for your application. Afterwards getters and setters get passed in only the part below that key in the KDB. For example, if you call `elektraOpen` with `"/sw/org/myapp/#0/current"`, you can access your applications configuration value for the key `"/sw/org/myapp/#0/current/message"` with the provided getters and setters by passing them only `"message"` as the name for the configuration value.

3.6.2 Read Values from the KDB

A typical application wants to read some configuration values at start. This should be made as easy as possible for the developer. Reading configuration data in most cases is not part of the business logic of the application and therefore should not "pollute" the applications source code with cumbersome setup and file-parsing code. This is exactly where Elektra comes in handy, because you can leave all the configuration file handling and parsing to the underlying layers of Elektra and just use the high-level API to access the desired data. Reading values from KDB can be done with elektra-getter functions that follow a simple naming scheme:

`elektraGet` + the type of the value you want to read.

For example, you can get the value for the key named "message" like this:
```
const char * message = elektraGetString (elektra, "message");
```

Sometimes you'll want to access arrays as well. You can access single elements of an array using the provided array-getters following again a simple naming scheme:

`elektraGet` + the type of the value you want to read + `ArrayElement`.

For example, you can get the value at index 3 for the array "message" like this:
```
const char * message = elektraGetStringArrayElement (elektra, "message", 3);
```

To get the size of the array you would like to access you can use the function `elektraArraySize`:
```
kdb_long_long_t arraySize = elektraArraySize (elektra, "message");
```

For some background information on arrays in Elektra see the Array tutorial, as well as our decision document on this topic. Please note that the high level API does not support arrays with missing elements. If an element is missing (and the specification provides no default value), getters will fail.

Notice that both the getters for primitive types and the getters for array types do not accept error parameters. The library expects you to run a correct Elektra setup. If the configuration is well specified, no runtime errors can occur when reading a value. Therefore the getters do not accept an error variable as argument. If there is however a severe internal error, or you try to access a key which you have not specified correctly, then the library will call the error callback set with `elektraFatalErrorHandler` to prevent data inconsistencies or exceptions further down in your application.

You can find the complete list of the available functions for all supported value types in `elektra.h`

3.6.3 Writing Values to the KDB

Sometimes, after having read a value from the KDB, you will want to write back a modified value. As described in `Read values from the KDB` we follow a naming scheme for getters. The high-level API provides setters follow an analogous naming scheme as well. For example, to write back a modified "message", you can call `elektraSetString`:
```
elektraSetString (elektra, "message", "This is the new message", NULL);
```

The counterpart for array-getters again follows the same naming scheme:
```
elektraSetStringArrayElement (elektra, "message", "This is the third new message", NULL);
```

Because even the best specification and perfect usage as intended can not prevent any error from occurring, when saving the configuration, all setter-functions take an additional `ElektraError` argument, which will be set if an error occurs.

3.6.4 Raw Values

You can use `const char * elektraGetRawString (Elektra * elektra, const char * name)` to read the raw (string) value of a key. No type checking or type conversion will be attempted. Additionally this function does not call the fatal error handler. It will simply return `NULL`, if the key was not found.

If you want to set a raw value, use `void elektraSetRawString (Elektra * elektra, const char * name, const char * value, KDBType type, ElektraError ** error)`. Obviously you have to provide a type for the value you set, so that the API can perform type checking, when reading the value next time.

Similar functions are provided for array elements:

```
const char * elektraGetRawStringArrayElement (Elektra * elektra, const char * name, kdb_long_long_t index);
void elektraSetRawStringArrayElement (Elektra * elektra, const char * name, kdb_long_long_t index, const
       char * value, KDBType type, ElektraError ** error);
```

3.6.4.1 Type Information

The type information is stored in the `"type"` metakey. `KDBType elektraGetType (Elektra * elektra, const char * keyname)` (or `KDBType elektraGetArrayElementType (Elektra * elektra, const char * name, kdb_long_long_t index)` for array elements) lets you access this information. A setter is not provided, because Elektra assumes keys to always have the same type (as specified).

3.6.4.2 Use cases for raw values

`elektraGetType`, `elektraGetRawString` and `elektraSetRawString` can be used together to create custom data types. If your application for example uses arbitrary-precision integers, you could something similar to these functions:

```
bignum * elektraGetBigNum (Elektra * elektra, const char * keyname)
{
  KDBType type = elektraGetType (elektra, keyname);
  if (strcmp (type, "bignum") != 0)
  {
    return NULL;
  }
  const char * rawValue = elektraGetRawString (elektra, keyname);
  return rawValue == NULL ? NULL : stringToBigNum (rawValue);
}
void elektraSetBigNum (Elektra * elektra, const char * keyname, bignum * value, ElektraError ** error)
{
  const char * rawValue = bigNumToString (value);
  elektraSetRawString (elektra, keyname, rawValue, "bignum", error);
}
```

To get the `type` plugin to validate your custom types you should make sure the `check/type` metadata is set to `string` (or `any`) on all keys that use custom types. This works, because the `type` plugin prefers the value of `check/type` over that of `type`.

3.6.5 Binary Values

The high-level API does not support binary key values at this time.

3.7 Example

```
#include <stdio.h>
#include <elektra.h>
int main ()
{
  ElektraError * error = NULL;
  Elektra * elektra = elektraOpen ("/sw/org/myapp/#0/current", NULL, NULL, &error);
  if (elektra == NULL)
  {
    printf ("Sorry, there seems to be an error with your Elektra setup: %s\n", elektraErrorDescription
       (error));
    elektraErrorReset (&error);
    printf ("Will exit now...\n");
    exit (EXIT_FAILURE);
  }
  const char * message = elektraGetString (elektra, "message");
  printf ("%s", message);
  elektraClose (elektra);
  return 0;
}
```

Chapter 4

README.md

infos/maintainer = Maximilian Irlinger max@maxirlinger.at

Chapter 5

elektra-libs(7) – libs overview

Note: Some information in this document is outdated and will change before the release of Elektra 1.0.

5.1 Highlevel APIs

5.1.1 Highlevel

`libelektra-highlevel.so`

Contains the **highlevel API**. See also examples.

5.1.2 Notification

`libelektra-notification.so`

notification provides the `notification API`. Usage examples:

- `Basic notifications using polling`
- `Using asynchronous I/O bindings`
- `Reload KDB when Elektra's configuration has changed`

5.2 Base Elektra Libraries

Since version **0.8.15 libelektra** is split into following libraries:

5.2.1 Libkdb

`libelektra-kdb.so`

Accesses the configuration files by orchestrating the plugins. The implementation lives in elektra.

It coordinates the interactions between the applications and the plugins.

loader contains source files that implement the plugin loader functionality as used by `libelektra-kdb`.

5.2.2 Libcore

```
libelektra-core.so
<kdbhelper.h>
<kdb.h> (key* and ks*)
```

Contains the fundamental data-structures every participant of Elektra needs to link against. It should be the only part that access the internal data structures. The implementation lives in elektra.

5.2.3 Libease

```
libelektra-ease.so
```

libease contains data-structure operations on top of libcore which do not depend on internals. Applications and plugins can choose to not link against it if they want to stay minimal. Its main goal is to make programming with Elektra easier if some extra kB are not an issue.

5.2.4 Libplugin

```
libelektra-plugin.so
```

libplugin contains `elektraPlugin*` symbols to be used by plugins.

5.2.5 Libmeta

```
libelektra-meta.so
```

libmeta contains metadata operations as described in **METADATA.ini**. Currently mainly contains legacy code and some generic metadata operations.

5.2.6 Libmerge

```
libelektra-merge.so
```

libmerge provides functionality for 3-way merges of keysets.

5.2.7 Libelektra

Is a legacy library that provides the same functionality as `libelektra-kdb` and `libelektra-core`. The sources can be found in **libelektra**.

5.3 Other Libraries

5.3.1 Libpluginprocess

```
libelektra-pluginprocess.so
```

libpluginprocess contains functions aiding in executing plugins in a separate process and communicating with those child processes. This child process is forked from Elektra's main process each time such plugin is used and gets closed again afterwards. It uses a simple communication protocol based on a KeySet that gets serialized through a pipe via the dump plugin to orchestrate the processes.

This is useful for plugins which cause memory leaks to be isolated in an own process. Furthermore this is useful for runtimes or libraries that cannot be reinitialized in the same process after they have been used.

5.3.2 Libtools

libtools is a high-level C++ shared-code for tools. It includes:

- plugin interface
- backend interface
- 3-way merge

5.3.3 Utility

libutility provides utility functions to be used in plugins.

5.3.4 Libinvoke

`libelektra-invoke.so`

libinvoke provides a simple API allowing us to call functions exported by plugins.

5.3.5 IO

`libelektra-io.so`

io provides the `common API` for using asynchronous I/O bindings.

5.3.6 Globbing

`libelektra-globbing.so`

globbing provides globbing functionality for Elektra.

The supported syntax is a superset of the syntax used by `glob(7)`. The following extensions are supported:

- `#`, when used as `/#/` (or `/#"` at the end of the pattern), matches a valid array item
- _ is the exact opposite; it matches anything but a valid array item
- if the pattern ends with `/__`, matching key names may contain arbitrary suffixes

For more info take a look a the documentation of `elektraKeyGlob()` and `elektraKsGlob()`.

5.3.7 Record

`libelektra-record`

record implements management functionality for session recording.

Chapter 6

README

- infos =

infos/author =

- infos/licence = BSD
- infos/status = experimental maintained
- infos/provides =
- infos/description =

6.0.1 Introduction

Explain what this library does. What is its purpose? What is its responsibility?

6.0.2 Files

- Link to the header files.
- Name of `.so` file.

6.0.3 Classes

Explain which classes the library contains. The method names are `elektraClass*`. Add a link to the API documentation.

6.0.4 Dependencies

None.

6.0.5 Quality

More information than `infos/status` including Performance Characteristics. Mention open issues here.

6.0.6 Further Links

Link to tutorials etc.

6.0.7 Notes

None.

Chapter 7

README

- infos =

infos/author = Thomas Wahringer waht@libelektra.org

- infos/licence = BSD
- infos/status =
- infos/provides =
- infos/description =

7.1 Example I/O Binding

This directory contains a complete example of an I/O binding for documentation purposes. This binding is not functional.

Chapter 8

Plugin: ansible

- infos = Information about the ansible plugin is in keys below
- infos/author = Maximilian Irlinger `max@maxirlinger.at`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/ansible
- infos/recommends =
- infos/placements = setstorage
- infos/status = recommended productive maintained conformant unittest writeonly
- infos/metadata =
- infos/description = export Ansible playbooks

8.1 Introduction

Provides a write-only storage plugin for use with `kdb export` and `kdb record-export`. The output format is an Ansible playbook that utilized the `ansible-libelektra` module.

Keys that possess the metakey `meta:/elektra/removed` will be removed using the `remove` option in ansible-libelektra.

8.2 Plugin Configuration

You can use the following configuration keys to modify the behavior and output of the plugin:

Key	Default Value	Description
`playbook`	`1`	Whether to generate a whole playbook or just a (list of) task(s) (`0`)
`playbook/name`	My Elektra Playbook	The `name` property of the playbook
`playbook/hosts`	all	The `hosts` property of the playbook
`task/main/name`	Set Elektra Keys	The name of the 'main' task of the playbook

8.3 Dependencies

This plugin requires `yaml-cpp`. On a Debian based OS the package for the library is called `libyaml-cpp-dev`. On macOS you can install the package `yaml-cpp` via `HomeBrew`.

8.4 Examples

```
# Backup-and-Restore: user:/tests
kdb set user:/tests/company/roles/ceo Hans
#> Create a new key user:/tests/company/roles/ceo with string "Hans"
kdb export user:/tests/company ansible -c playbook/name="Company Roles",task/main/name="I can customize this
       too"
# RET:0
---
- name: Company Roles
  hosts: all
  collections:
    - elektra_initiative.libelektra
  tasks:
    - name: I can customize this too
      elektra:
        keys:
          - user:
              tests:
                company:
                  roles:
                    ceo:
                      - value: Hans
```

8.5 Limitations

- This plugin only supports writing of Ansible Playbooks. It is currently not possible to read them with this plugin. Do not use this plugin as a general-purpose storage plugin.
- If keys below `system:/elektra/mountpoints` are included, we will always create a second task for them instead of using the `mount` operation of the ansible-libelektra module. This task will be created first. Keys created during the 'main' task will then already be able to use the correctly mounted files.

Chapter 9

Plugin: augeas

- infos = Information about augeas plugin is in keys below
- infos/author = Felix Berlakovich elektra@berlakovich.net
- infos/licence = BSD
- infos/provides = storage
- infos/needs =
- infos/recommends = glob keytometa
- infos/placements = getstorage setstorage
- infos/status = maintained unittest libc configurable
- infos/metadata = order
- infos/description = reading and writing configurations via libaugeas

9.1 Introduction

This is a plugin for reading and writing configuration files with help from Augeas. The plugin should be able to read all configuration files for which an Augeas lens exists. However, not all stock lenses of Augeas have been tested yet. A detailed description of the lens language and a tutorial on how to write new lenses can be found at http://augeas.net/

9.2 Dependencies

- `libaugeas-dev`: You need version 0.16 or higher

9.3 Installation

See installation. The package is called `libelektra5-augeas`.

If you have installed Augeas manually, it may be necessary to update the ld configuration. This is especially true if an older version of Augeas is installed also. Such a situation may lead to an error similar to this:

```
/usr/lib/libaugeas.so.0: version `AUGEAS_0.16.0' not found (required by kdb)
```

This is because `ld` tries to link `/usr/lib/libaugeas.so.0` which is an older version of Augeas. Simply add the path to the newer library to your ld search paths (consult your system documentation on how to do this).

9.4 Mounting and Configuration

In order to mount the hosts file with the augeas plugin, issue the following command:
```
sudo kdb mount /etc/hosts system:/tests/hosts augeas lens=Hosts.lns
```

And to unmount the file run:
```
sudo kdb umount system:/tests/hosts
```

The value of the plugin configuration option "lens" should be the module name of the lens (Hosts in the example) with a '.lns' suffix. Depending on your distribution and kind of installation, lenses can be found at `/usr/share/augeas/lenses/dist`, `/usr/local/share/augeas/lenses/dist`, or something similar. The lens module name is equal to the filename without extension in pascal notation. For example, the lens `/usr/share/augeas/lenses/dist/hosts.aug` contains the module Hosts.

Note that, without configuring the plugin to use a lens, the plugin will print an error message on the first usage:
```
sudo kdb mount /etc/hosts system:/tests/hosts augeas #The lens is missing here!
kdb ls system:/tests/hosts
# RET: 5
# ERROR: *C01200*
```

This happens because the plugin does not know which lens it should use to read and write the configuration. For that reason, the lens configuration option was supplied together with the mount command.

And again to unmount the `hosts` file simply run:
```
sudo kdb umount system:/tests/hosts
```

9.5 Restrictions

9.5.1 Inner Node Values

Currently no Augeas lens supports values for inner nodes. Unfortunately no validation plugin exists yet that would prevent such modifications early:
```
sudo kdb mount /etc/hosts system:/tests/hosts augeas lens=Hosts.lns
kdb set system:/tests/hosts/1 somevalue
# RET: 5
# ERROR: *C01100*
```

The operation simply fails with an undescriptive error.

Finally, to unmount the `hosts` file simply run:
```
sudo kdb umount system:/tests/hosts
```

9.5.2 Leaky Abstraction of Order

Most Augeas lenses require subtrees to be in a specific order. For example the hosts lens requires the ipaddr node of an entry to precede the canonical node. Unfortunately the Augeas storage plugin has no knowledge about this required order. Therefore the correct order must be ensured via order metakeys. Otherwise saving the KeySet may fail. As an example consider the following kdb shell script:
```
kdbGet system:/hosts
keySetName system:/hosts/6
ksAppendKey
keySetName system:/hosts/6/ipaddr
keySetString 14.14.14.14
ksAppendKey
keySetName system:/hosts/6/canonical
keySetString newhost
ksAppendKey
kdbSet system:/hosts
```

This fails with an error similar to this

```
Sorry, module storage issued the error C03100:
an Augeas error occurred: Failed to match
some augeas match expression
with tree
{ \"canonical\" = \"newhost\" } { \"ipaddr\" = \"14.14.14.14\" }
```

Whereas the following script succeeds due to the correct order

```
kdbGet system:/hosts
keySetName system:/hosts/6
ksAppendKey
keySetName system:/hosts/6/ipaddr
keySetString 14.14.14.14
keySetMeta order 100
ksAppendKey
keySetName system:/hosts/6/canonical
keySetString newhost
keySetMeta order 110
ksAppendKey
kdbSet system:/hosts
```

9.6 Planned Improvements

- a validation plugin preventing inner node values

Chapter 10

Plugin: backend

- infos = Information about the backend plugin is in the keys below
- infos/author = Vid Leskovar vid.leskovar5@gmail.com, Klemens Böswirth k.boeswirth+git@gmail.com
- infos/licence = BSD
- infos/provides = backend
- infos/needs =
- infos/recommends =
- infos/placements = backend
- infos/status = nodep nodoc
- infos/description = Plugin implementing full backend functionality

10.1 Introduction

This plugin is the default backend plugin.

> **Note** If you want to implement your own backend plugin, please read the general documentation about backend plugins first. Only use this plugin as an example, if you intentionally want to create a very similar backend plugin. The general API is more flexible than what this plugin implements.

Chapter 11

Plugin: backend_odbc

- infos = Information about the backendOdbc plugin is in the keys below
- infos/author = Florian Lindner florian.lindner@student.tuwien.ac.at
- infos/licence = BSD
- infos/provides = backend
- infos/needs =
- infos/recommends =
- infos/placements = backend
- infos/status = reviewed unittest experimental
- infos/description = Plugin implementing full backend functionality for ODBC data sources

11.1 Introduction

This plugin is a backend plugin that stores and retrieves data using ODBC (Open Database Connectivity) data sources. It was tested with unixODBC on Linux, but should also work with iODBC and Microsoft ODBC (on Windows).

> If you want to use it with one of the latter two ODBC implementations, feel free to update this documentation with your experiences!

11.2 Required database scheme

The minimum requirement is a table with at least two columns:

- Key-name (string, primary key (PK))
- Key-value (string)

Additionally, a second table with at least three columns is required.

- Key-name (string, foreign key (FK) to the first table)
- Metakey-name (string)
- Metakey-value (string)

The primary key of this table consists of two columns: the **key-name** and the **metakey-name**. In the language of ER-modelling, the metatable can therefore by considered a **weak-entity**.

> Currently, only data sources with tables for metadata are supported! So you have to define a meta-table. Data sources without a table for metadata will probably be supported in the future. If you want to use metadata, the ODBC driver for your data source has to support **outer joins**. This implies that currently, only ODBC drivers with support for outer joins are supported by the ODBC backend.

The tables may also contain other columns, but they are not processed by this plugin and must support NULL- or DEFAULT-values, if you want to add tuples to the table via Elektra (e.g. by calling `kdb set <key> <value>`).

If the column for the key-name is not defined as a primary key and multiple rows contain the same key-name, they are treated as one key, where the value is taken from the first row and the metadata is combined. If multiple metakeys with the same metakey-name exist for a key, the last metavalue is used. This behavior is part of the internal algorithm of the plugin and could change in the future. So if you want a predictable and expectable behavior, make sure that the described primary- and foreign-key constraints are respected.

11.3 Mounting

The mountpoint definition for ODBC mountpoints, which is stored at `system:/elektra/mountpoints/<MP>/definition` is defined as follows. The placeholder `<MP>` is the path in the KDB where the root of the defined mountpoint is located. This is the path you specify when creating the mountpoint.

The key-names for an ODBC mountpoint-definition are:

```
system:/elektra/mountpoints/<mp>
system:/elektra/mountpoints/<mp>/plugins/backend
system:/elektra/mountpoints/<mp>/plugins/backend/name
system:/elektra/mountpoints/<mp>/definition/dataSourceName
system:/elektra/mountpoints/<mp>/definition/userName
system:/elektra/mountpoints/<mp>/definition/password
system:/elektra/mountpoints/<mp>/definition/timeout
system:/elektra/mountpoints/<mp>/definition/table/name
system:/elektra/mountpoints/<mp>/definition/table/keyColName
system:/elektra/mountpoints/<mp>/definition/table/valColName
system:/elektra/mountpoints/<mp>/definition/metaTable/name
system:/elektra/mountpoints/<mp>/definition/metaTable/keyColName
system:/elektra/mountpoints/<mp>/definition/metaTable/metaKeyColName
system:/elektra/mountpoints/<mp>/definition/metaTable/metaValColName
```

The key-values for the keys under `definition` are defined as follows:

- *dataSourceName:* The name of the ODBC data source (as defined in the odbc.ini file)
- *userName:* Name of the user that should be used to connect to the data source
  - can be empty if no username is required or if the value should be read from the odbc.ini file
- *password:* Password for the user who wants to connect
  - can be empty if no password is required or if the value should be read from the odbc.ini file
- *timeout:* The number of seconds to wait after a connection attempt is considered as failed (0 = wait indefinitely)
  - if empty, a default value is used

- *table/name:* Name of the table where the data is stored
- *table/keyColName:* The name of the column where the keynames are stored
- *table/valColName:* The name of the column where the key-values (strings) are stored
- *metaTable/name:* The name of the table where the metadata is stored
- *metaTable/keyColName:* The name of the column in the MetaTable where the name of the key the metakey belongs to is stored
- *metaTable/metaKeyColName:* The name of the column where the name of the metakeys is stored
- *metaTable/metaValColName:* The name of the column where the value (string) of the metakeys is stored

There is a new command for the kdb-tool: `kdb mountOdbc`. Please be aware that the ODBC backend, in contrast to the classic file-based backend, currently does not support adding other plugins to the mountpoint.

The `kdb mountOdbc` command accepts arguments for the previously described values of the *mountpoint definition*. For further details, please refer to the man page for that command.

For unmounting, there is no new command necessary. Just use the well-known `kdb umount <mountpoint>`, exactly like for the file-based backend.

For more information about the ODBC backend, there is a tutorial available at /doc/tutorials/odbc-backend.md.

11.4 Testing and Benchmarking

There are no dedicated unit tests that actually store, read or delete data from an ODBC data source. The reason is, that with the current concept, a valid ODBC data source definition (for unixODBC in `/etc/unixODBC/odbc.ini`) must be present. We don't want to write to such an important system file that can influence other applications. It would also be cumbersome if we require the user to set up a specific ODBC configuration before running the tests .

However, for basic functionality testing and performance evaluation, you can create a valid ODBC mountpoint yourself using `kdb mountOdbc` and then run the benchmark which is implemented in `/benchmarks/mountpoint.c` for the created mountpoint. This approach makes it possible to test and benchmark any valid mountpoint in the KDB. So different backends and plugins with different configurations can be tested and benchmarked. Just create a mountpoint with the backends, plugins and configuration that you want and run the benchmark with e.g. `bin/benchmark_mountpoint <MP path>/benchmark` for this mountpoint.

A short example that shows creating a new ODBC mountpoint and running benchmarks for KeySets and metadata:

```
bin/kdb mountOdbc Selektra "" "" elektraKeys keyName keyValue metaKeys keyName metaKeyName metaKeyValue ""
       user:/odbcSqlite
bin/benchmark_mountpoint -c 1000 user:/odbcSqlite
bin/benchmark_mountpoint -Mc 1000 user:/odbcSqlite
```

More information about benchmarking is available at /benchmarks/README.md.

Chapter 12

Plugin: base64

- infos = Information about base64 plugin is in keys below
- infos/author = Peter Nirschl peter.nirschl@gmail.com
- infos/licence = BSD
- infos/provides = binary
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest nodep libc final configurable
- infos/metadata =
- infos/description = Base64 Encoding

12.1 Base64

12.1.1 Introduction

The Base64 Encoding (specified in RFC4648) is used to encode arbitrary binary data to ASCII strings.

This is useful for configuration files that must contain ASCII strings only.

The `base64` plugin encodes binary values before `kdb set` writes the configuration to the file. The values are decoded back to their original value after `kdb get` has read from the configuration file.

12.1.2 Usage

To mount a simple backend that uses the Base64 encoding, you can use:
```
sudo kdb mount test.ecf /tests/base64/test base64
```

. To unmount the plugin use the following command:
```
sudo kdb umount /tests/base64/test
```

. All encoded binary values will look something like this:
```
@BASE64SGVsbG8gV29ybGQhCg==
```

. And for a null-key it will be:
```
@BASE64
```

12.1.3 Modes

The plugin supports two different modes:

1. Escaping Mode
2. Meta Mode

. By default the plugin uses escaping mode which has the advantage that a storage plugin does not have to change its behavior at all to work in conjunction with Base64.

12.1.3.1 Escaping Mode

In order to identify the base64 encoded content, the values are marked with the prefix `@BASE64`. To distinguish between the `@` as character and `@` as Base64 marker, all strings starting with `@` will be modified so that they begin with `@@`.

12.1.3.1.1 Example The following example shows how you can use this plugin together with the TOML plugin to store binary data.

```
# Mount the TOML and Base64 plugin
sudo kdb mount test_config.toml user:/tests/base64 toml base64
# Copy binary data
kdb cp system:/elektra/modules/base64/exports/get user:/tests/base64/binary
# Print binary data
kdb get user:/tests/base64/binary
# STDOUT-REGEX: ^(\\x[0-9a-f]{1,2})+$
# The value inside the configuration file is encoded by the Base64 plugin
kdb file user:/tests/base64 | xargs cat
# STDOUT-REGEX: binary.*=.*"@BASE64[a-zA-Z0-9+/]+={0,2}"
# Undo modifications
kdb rm -r user:/tests/base64
sudo kdb umount user:/tests/base64
```

12.1.3.2 Meta Mode

Some file formats such as `YAML` already support Base64 encoded data. In YAML `binary data` starts with the tag `!!binary` followed by a Base64 encoded scalar:

```
!!binary SGVsbG8sIFlBTUwh # "Hello, YAML!"
```

. For YAML it would not make sense to use the format of the escaping mode:

```
# Another YAML implementation will not be able to decode this data
# because of the prefix `@BASE64`!
!!binary "@BASE64SGVsbG8sIFlBTUwh"
```

. Base64 supports another mode called "meta mode". In this mode the Base64 plugin encodes the value, but does not add a prefix. To use the escaping mode a plugin must add the configuration key `/binary/meta`. Afterwards the Base64 plugin encodes and decodes all data that contains the metakey `type` with the value `binary`.

The diagram below shows how the Base64 conversion process works in conjunction with the YAML CPP plugin.

12.1.3.2.1 Example The following example shows you how you can use the TOML plugin together with Base64's meta mode.

```
# Mount ni and Base64 plugin (provides `binary`) with the configuration key `binary/meta`
sudo kdb mount test_config.ni user:/tests/base64 ni base64 binary/meta=
# Save base64 encoded data `"value"` (`0x76616c7565`)
kdb set user:/tests/base64/encoded dmFsdWUA
kdb file user:/tests/base64/encoded | xargs cat | grep encoded
#> encoded = dmFsdWUA
# Tell Base64 plugin to decode and encode key value
kdb meta-set user:/tests/base64/encoded type binary
# Receive key data (the `\x0` at the end marks the end of the string)
kdb get user:/tests/base64/encoded
#> \x76\x61\x6c\x75\x65\x0
# Undo modifications
kdb rm -r user:/tests/base64
sudo kdb umount user:/tests/base64
```

For another usage example, please take a look at the ReadMe of the YAML CPP plugin.

Chapter 13

Plugin: blacklist

- infos = Information about the blacklist plugin is in keys below
- infos/author = Robert Sowula `robert@sowula.at`
- infos/licence = BSD
- infos/needs =
- infos/provides = check
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest nodep
- infos/metadata = check/blacklist/#
- infos/description = blacklisting key values

13.1 Introduction

This plugin is a blacklist plugin that blocks the assignment of specific values to a key.

13.2 Usage

The plugin looks for the metadata array `check/blacklist/#` and reject all matching key values.

For example:

```
check/blacklist = #2
check/blacklist/#0 = water
check/blacklist/#1 = fire
check/blacklist/#2 = air
```

All values in the array will be rejected. The array indices don't have to be continuous, using e.g. only `#1`, `#2` and `#4` is also allowed. Just make sure `check/blacklist` is set to the largest index in the array.

13.3 Installation

See installation. The package is called `libelektra5-extra`.

13.4 Examples

```
sudo kdb mount blacklist.ecf /tests/blacklist blacklist
# valid initial value + setup valid blacklist list
kdb set user:/tests/blacklist ""
kdb set user:/tests/blacklist/value water
kdb meta-set spec:/tests/blacklist/value check/blacklist '#2'
kdb meta-set spec:/tests/blacklist/value check/blacklist/#0 fire
kdb meta-set spec:/tests/blacklist/value check/blacklist/#1 air
kdb meta-set spec:/tests/blacklist/value check/blacklist/#2 cold/water
# should succeed
kdb set user:/tests/blacklist/value earth
# should fail
kdb set user:/tests/blacklist/value fire
# RET:5
# ERROR:C03200
# should fail
kdb set user:/tests/blacklist/value air
# RET:5
# ERROR:C03200
# should fail
kdb set user:/tests/blacklist/value cold/water
# RET:5
# ERROR:C03200
```

It is also possible to blacklist empty values:

```
kdb set user:/tests/blacklist/empty water
kdb meta-set spec:/tests/blacklist/empty check/blacklist '#0'
kdb meta-set spec:/tests/blacklist/empty check/blacklist/#0 ''
# should succeed
kdb set user:/tests/blacklist/empty earth
# should fail
kdb set user:/tests/blacklist/empty ''
# RET:5
# ERROR:C03200
# Undo changes
kdb rm -r spec:/tests/blacklist
kdb rm -r user:/tests/blacklist || kdb rm -r system:/tests/blacklist
# sudo kdb umount spec:/tests/blacklist
sudo kdb umount /tests/blacklist
```

Chapter 14

Plugin: blockresolver

- infos = Information about the blockresolver plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = resolver
- infos/recommends =
- infos/placements = rollback getresolver setresolver commit
- infos/status = unittest nodep libc configurable preview experimental difficult unfinished concept
- infos/metadata =
- infos/description = resolver for parts in a file configuration file

14.1 Introduction

The `blockresolver` can be used to only resolve a tagged block inside a configuration file.

14.2 Installation

See installation. The package is called `libelektra5-experimental`.

14.2.1 Implementation Details

`blockresolver` extracts the requested block from the configurations file and writes it into a temporary file. Afterwards Elektra will only work on the temporary file until kdbSet is called. On kdbSet the contents of the temporary file will be merged with parts outside of the requested block from the original file.

14.3 Usage

```
`kdb mount -R blockresolver /path/to/my/file /mountpoint -c identifier="identifier-tag"`
```

where `identifier` specifies the tag `blockresolver` will search for in the configuration file.

A block consists of 2 parts:

- beginning: the identifier suffixed with `start`
- end: the identifier suffixed with `stop`

14.4 Limitations

Currently the identifier must be unique.

14.5 Example

```
# Backup-and-Restore: system:/tests/blockresolver
# create testfile
kdb set system:/tests/blockfile $(mktemp)
echo 'text'                  >  $(kdb get system:/tests/blockfile)
echo 'more text'             » $(kdb get system:/tests/blockfile)
echo 'some more text'        » $(kdb get system:/tests/blockfile)
echo '»> block config start' » $(kdb get system:/tests/blockfile)
echo 'key1=val1'            » $(kdb get system:/tests/blockfile)
echo '»> block config stop'  » $(kdb get system:/tests/blockfile)
echo 'text again'            » $(kdb get system:/tests/blockfile)
echo 'and more text'         » $(kdb get system:/tests/blockfile)
echo 'text'                  » $(kdb get system:/tests/blockfile)
sudo kdb mount -R blockresolver $(kdb get system:/tests/blockfile) system:/tests/blockresolver -c
       identifier="»> block config" mini
# check testfile
cat $(kdb get system:/tests/blockfile)
#> text
#> more text
#> some more text
#> »> block config start
#> key1=val1
#> »> block config stop
#> text again
#> and more text
#> text
# only the block between the tags is read!
kdb export system:/tests/blockresolver mini
# STDOUT-REGEX: key1.*=.*val1
# add a new key to the resolved block
kdb set system:/tests/blockresolver/key12 val12
cat $(kdb get system:/tests/blockfile)
#> text
#> more text
#> some more text
#> »> block config start
#> key1=val1
#> key12=val12
#> »> block config stop
#> text again
#> and more text
#> text
# cleanup
kdb rm -r system:/tests/blockresolver
rm $(kdb get system:/tests/blockfile)
kdb rm system:/tests/blockfile
sudo kdb umount system:/tests/blockresolver
```

Chapter 15

Plugin: c

- infos = Information about the c plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/maintainer = Florian Lindner florian.lindner@student.tuwien.ac.at
- infos/licence = BSD
- infos/needs = ccode
- infos/provides = storage/c
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained reviewed unittest nodep libc writeonly preview
- infos/metadata =
- infos/description = C code KeySet exports for Elektra

15.1 Usage

Export a subset of the KDB as C code which creates a KeySet (`ksNew(<number of keys>, keyNew(...), ...)`). This can be useful for generating test data or help to generate code for already defined configuration data.

```
kdb export user:/testdata c
```

The output can be used in C code for generating KeySets and the keys inside them. So one use case is to extract a part of the KDB and copy the generated code to create the exported part of the KDB programmatically, e.g. for another Elektra installation.

It can also be useful if you develop your own application and first create the necessary keys manually. Then you can export them from the KDB and just paste the generated code in the right place of the codebase of your application.

> Please note that if you use this plugin for creating a mountpoint (e.g. by calling `kdb mount <filename> <key> c`), only the content written by the last `kdbSet (...)` which includes the mountpoint will be inside the file. If you call `kdbSet (...)` for such a mountpoint, the previous content of the file is erased.
>
> This is because the c plugin is implemented as a write-only plugin. It's recommended to only use it for exporting parts of the KDB by calling `kdb export <parent key>`.

15.2 Example

In this example, we add some keys and metakeys to the KDB and export them with the c plugin. The output can directly be used inside C source code which uses Elektra.

```
kdb set user:/tests/cplugin/key1 value1
#> Create a new key user:/tests/cplugin/key1 with string "value1"
kdb set user:/tests/cplugin/key2 value2
#> Create a new key user:/tests/cplugin/key2 with string "value2"
kdb meta-set user:/tests/cplugin/key2 metakey2.1 metaval2.1
kdb set user:/tests/cplugin/key2 metakey2.2 metaval2.2
kdb set user:/tests/cplugin/key3 value3
#> Create a new key user:/tests/cplugin/key3 with string "value3"
kdb export user:/tests/cplugin c
#> ksNew (3,
#>      keyNew ("user:/tests/cplugin/key1", KEY_VALUE, "value1", KEY_END),
#>      keyNew ("user:/tests/cplugin/key2", KEY_VALUE, "value2", KEY_META, "metakey2.1", "metaval2.1",
       KEY_META, "metakey2.2", "metaval2.2", KEY_END),
#>      keyNew ("user:/tests/cplugin/key3", KEY_VALUE, "value3", KEY_END),
#>      KS_END);
```

Chapter 16

Plugin: cache

- infos = Information about the cache plugin is in keys below
- infos/author = Mihael Pranjić mpranj@limun.org
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = pregetcache postgetcache
- infos/status = unittest experimental
- infos/metadata =
- infos/description = caches keysets from previous `kdbGet()` calls

16.1 Introduction

This caching plugin stores keysets from previous `kdbGet()` calls to improve performance when reading configuration files.

16.2 Usage

The cache plugin is compiled and enabled on compatible systems by default. No actions are needed to enable it.

16.3 Dependencies

POSIX compliant system (including XSI extensions). The plugin is only compiled if the plugins `resolver` and `mmapstorage` are also available.

16.4 Location of Cache

The cache files are located in the user's home directory below ~/.cache/elektra/ and shall not be altered, otherwise the behavior is undefined. If XDG_CACHE_HOME is set, the cache files are located below $XDG_↩ CACHE_HOME/elektra.

16.5 Configuration of Cache

Use the tool kdb cache to enable, disable or clear the cache.

16.6 Limitations

Incompatible with storage plugins, which do not always produce the same keyset on any invocation concerning the same configuration file. A notable example here is the ini plugin (see issue #2592).

Chapter 17

Plugin: cache

17.1 Additional shell tests for cache

This file contains important shell tests for mmapstorage which do not fit well into the plugin README.

17.2 Test kdb cp with cache and default resolver (refression test)

```
rm -rf $(dirname $(kdb file user:/))/multitest || $(exit 0)
mkdir -p $(dirname $(kdb file user:/))/multitest || $(exit 0)
echo "col1;col2" > $(dirname $(kdb file user:/))/multitest/first.csv
echo "l1c1;l2c2" » $(dirname $(kdb file user:/))/multitest/first.csv
echo "l2c1;l2c2" » $(dirname $(kdb file user:/))/multitest/first.csv
echo "" > $(dirname $(kdb file user:/))/multitest/empty.csv
kdb mount multitest/first.csv user:/tests/multifile/first.csv csvstorage
kdb mount multitest/empty.csv user:/tests/multifile/empty.csv csvstorage
kdb ls user:/tests/multifile/first.csv
#> user:/tests/multifile/first.csv/#0
#> user:/tests/multifile/first.csv/#0/#0
#> user:/tests/multifile/first.csv/#1
#> user:/tests/multifile/first.csv/#1/#0
#> user:/tests/multifile/first.csv/#2
#> user:/tests/multifile/first.csv/#2/#0
kdb ls user:/tests/multifile/empty.csv
#> user:/tests/multifile/empty.csv/#0
#> user:/tests/multifile/empty.csv/#0/#0
kdb cp -rf user:/tests/multifile/first.csv user:/tests/multifile/empty.csv
kdb ls user:/tests/multifile/first.csv
#> user:/tests/multifile/first.csv/#0
#> user:/tests/multifile/first.csv/#0/#0
#> user:/tests/multifile/first.csv/#1
#> user:/tests/multifile/first.csv/#1/#0
#> user:/tests/multifile/first.csv/#2
#> user:/tests/multifile/first.csv/#2/#0
kdb ls user:/tests/multifile/empty.csv
#> user:/tests/multifile/empty.csv/#0
#> user:/tests/multifile/empty.csv/#0/#0
#> user:/tests/multifile/empty.csv/#1
#> user:/tests/multifile/empty.csv/#1/#0
#> user:/tests/multifile/empty.csv/#2
#> user:/tests/multifile/empty.csv/#2/#0
rm -rf $(dirname $(kdb file user:/))/multitest
kdb umount user:/tests/multifile/first.csv
kdb umount user:/tests/multifile/empty.csv
```

17.3 Test kdb cp with cache and multifile resolver (refression test)

```
rm -rf $(dirname $(kdb file user:/))/multitest || $(exit 0)
mkdir -p $(dirname $(kdb file user:/))/multitest || $(exit 0)
echo "col1;col2" > $(dirname $(kdb file user:/))/multitest/first.csv
echo "l1c1;l2c2" » $(dirname $(kdb file user:/))/multitest/first.csv
echo "l2c1;l2c2" » $(dirname $(kdb file user:/))/multitest/first.csv
```

```
echo "col1;col2;col3" > $(dirname $(kdb file user:/))/multitest/other.csv
echo "l1c1;l2c2;l2c3" » $(dirname $(kdb file user:/))/multitest/other.csv
echo "l2c1;l2c2;l3c3" » $(dirname $(kdb file user:/))/multitest/other.csv
echo "" > $(dirname $(kdb file user:/))/multitest/empty.csv
sudo kdb mount -R multifile -c storage="csvstorage",pattern="*.csv",resolver="resolver" multitest
       user:/tests/multifile
kdb ls user:/tests/multifile/first.csv
#> user:/tests/multifile/first.csv/#0
#> user:/tests/multifile/first.csv/#0/#0
#> user:/tests/multifile/first.csv/#1
#> user:/tests/multifile/first.csv/#1/#0
#> user:/tests/multifile/first.csv/#2
#> user:/tests/multifile/first.csv/#2/#0
kdb ls user:/tests/multifile/empty.csv
#> user:/tests/multifile/empty.csv/#0
#> user:/tests/multifile/empty.csv/#0/#0
kdb cp -rf user:/tests/multifile/first.csv user:/tests/multifile/empty.csv
kdb ls user:/tests/multifile/first.csv
#> user:/tests/multifile/first.csv/#0
#> user:/tests/multifile/first.csv/#0/#0
#> user:/tests/multifile/first.csv/#1
#> user:/tests/multifile/first.csv/#1/#0
#> user:/tests/multifile/first.csv/#2
#> user:/tests/multifile/first.csv/#2/#0
kdb ls user:/tests/multifile/empty.csv
#> user:/tests/multifile/empty.csv/#0
#> user:/tests/multifile/empty.csv/#0/#0
#> user:/tests/multifile/empty.csv/#1
#> user:/tests/multifile/empty.csv/#1/#0
#> user:/tests/multifile/empty.csv/#2
#> user:/tests/multifile/empty.csv/#2/#0
rm -rf $(dirname $(kdb file user:/))/multitest
kdb umount user:/tests/multifile
```

Chapter 18

Plugin: ccode

- infos = Information about ccode plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = code
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = unittest nodep libc configurable discouraged
- infos/description = Decoding/Encoding engine which escapes unwanted characters.

18.1 CCode

18.1.1 Introduction

The `ccode` plugin replaces (escapes) any special characters with two characters:

- an escape character (default: `\`) and
- another character representing the escaped character (e.g `n` for newline)

before writing a `KeySet`. The plugin undoes this modification after reading a `KeySet`.

CCode provides a reasonable default configuration, using the usual escape sequences for C strings (e.g. `\n` for newline, `\t` for tab). You can also configure the escape character (`/escape`) and the mapping for special characters (`chars`).

18.1.2 Installation

See installation. The package is called `libelektra5-extra`.

18.1.3 Restrictions

This method of encoding characters is not as powerful as the hexcode plugin in terms of reduction. The hexcode plugin allows reduction of the character set to '0'-'9', 'a'-'f' and one escape character. So it can represent any key value with only 17 characters. On the other hand, ccode cannot reduce the set more than by half.

So when all control characters and non-ASCII characters need to vanish, it cannot be done with the ccode plugin. But it is perfectly suitable to reduce by some characters. The advantages are that the size only doubles in the worst case and that it is much easier to read.

18.1.4 C

In the C language, the following escape characters are present.

- `b`: backspace, hex: 08
- `t`: horizontal tab, hex: 09
- `n`: new line feed, hex: 0A
- `v`: vertical tab, hex: 0B
- `f`: form feed, hex: 0C
- `r`: carriage return, hex: 0D
- `\\`: back slash, hex: 5C
- `'`: single quote, hex: 27 `"`: double quote, hex: 22 `0`: null, hex: 00

This is also the default mapping.

18.1.4.1 Contract

Add `ccode` to `infos/needs` for any plugin that you want to be filtered by ccode.

Chapter 19

Plugin: conditionals

- infos = Information about the conditionals plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = unittest nodep libc global preview
- infos/metadata = check/condition assign/condition condition/validsuffix check/condition/any/# check/condition/all/# check/condition/none/# assign/condition/#
- infos/description = ensures key values through conditions

19.1 Introduction

This plugin uses if-then-else like conditions. It also works as global plugin.

19.2 Installation

See installation. The package is called `libelektra5-extra`.

19.3 Check Syntax

Stored in the metakey `check/condition` to validate data is:

`(IF-condition) ? (THEN-condition) : (ELSE-condition)` where the ELSE-condition is optional

Condition: `Key` *Operation* '('String' | '1234.56' | Key | '')`

Operations: `!=, ==, <, <=, =>, >, :=`, where:

- `:=` is used to set a key value
- others are for comparison as in C

19.3.1 Testing if Key Exists

`(! a/key)` evaluates to true if the key `a/key` doesn't exist, to false if it exists.

19.3.2 Assign Syntax

```
(IF-condition) ? ('ThenValue') : ('ElseValue')
```

Depending on if the condition is met, either 'ThenValue' or 'ElseValue' will be assigned as key value if the metakey `assign/condition` is used.

19.3.3 Experimental: Nested Conditions

Multiple conditions can be nested and combined using parentheses and `&&` (logical AND) or || (logical OR). Additional parentheses must be used to form valid conditions again. `( (condition 1) && (condition 2) )`

19.3.4 Valid Suffix

The `condition/validsuffix` can be used to define a list of valid suffixes to numeric values. If two operants have the same valid suffix or one of them no suffix they will be treated by their numeric value ignoring their suffix. ‘condition/validsuffix = 'm', 'cm', 'km'`would treat`2.3m`just as the numeric value`2.3` when comparing to another value having the same or no suffix.

19.3.5 Keynames

Keynames are all either relative to to-be-tested key (starting with `./` or `../`), relative to the parentkey (starting with `@/`) or absolute (e.g. `system:/key`).

19.3.6 Multiple Statements

It's also possible to test multiple conditions using `check/condition/{any,all,none}` as a meta array. Where `any` means that at least one statement has to evaluate to true, `all` that all statements have to evaluate to true, and `none` that no statement is allowed to evaluate to false (default). For multiple assign statements use `assign/condition` as a meta array. The first `assign/condition/#` statement that evaluates to true will be assigned and the rest ignored.

19.4 Example

```
(this/key  != 'value') ? (then/key == some/other/key) : (or/key <= '125')
```

Meaning: IF `this/key` NOT EQUAL TO "value"`THEN`then/key`MUST EQUAL`some/other/key`ELSE`or/key`MUST BE LESS THAN`125`

Another full example:

```
#Backup-and-Restore:user:/tests/conditionals
sudo kdb mount conditionals.dump user:/tests/conditionals conditionals dump
kdb set user:/tests/conditionals/fkey 3.0
kdb set user:/tests/conditionals/hkey hello
# will succeed
kdb meta-set user:/tests/conditionals/key check/condition "(../hkey == 'hello') ? (../fkey == '3.0')"
# will fail
kdb meta-set user:/tests/conditionals/key check/condition "(../hkey == 'hello') ? (../fkey == '5.0')"
# RET:5
# ERROR:C03200
```

Assignment example:

```
kdb set user:/tests/conditionals/hkey Hello
kdb meta-set user:/tests/conditionals/hkey assign/condition "(./ == 'Hello') ? ('World')"
# alternative syntax: "(../hkey == 'Hello') ? ('World')
kdb get user:/tests/conditionals/hkey
#> World
# cleanup
kdb rm -r user:/tests/conditionals
sudo kdb umount user:/tests/conditionals
```

Global plugin example:

```
# Backup old list of global plugins
kdb set user:/tests/msr $(mktemp)
kdb export system:/elektra/globalplugins > $(kdb get user:/tests/msr)
sudo kdb mount main.ini /tests/conditionals ni
sudo kdb mount sub.ini /tests/conditionals/sub ni
# mount conditionals as global plugin
sudo kdb global-mount conditionals || $(exit 0)
# create testfiles
echo 'key1=val1'                                                  >  `kdb file system:/tests/conditionals`
echo '[key1]'                                                       » `kdb file system:/tests/conditionals`
echo "check/condition=(./ == 'val1') ? (../sub/key == 'true')" » `kdb file system:/tests/conditionals`
echo "key=false" > `kdb file system:/tests/conditionals/sub`
# should fail and yield an error
kdb export system:/tests/conditionals ni
# ERROR:C03200
# Sorry, module conditionals issued the error C03200:
# Validation failed: Validation of Key key1: (./ == 'val1') ? (../sub/key == 'true') failed. ((../sub/key ==
       'true') failed)
kdb set system:/tests/conditionals/sub/key true
# should succeed
kdb export system:/tests/conditionals ni
# cleanup
kdb rm -r /tests/conditionals
sudo kdb umount /tests/conditionals/sub
sudo kdb umount /tests/conditionals
sudo kdb global-umount conditionals
kdb rm -r system:/elektra/globalplugins
kdb import system:/elektra/globalplugins < $(kdb get user:/tests/msr)
rm $(kdb get user:/tests/msr)
kdb rm user:/tests/msr
```

Chapter 20

Plugin: constants

- infos = All information you want to know
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = storage/info
- infos/needs =
- infos/recommends =
- infos/placements = setstorage getstorage
- infos/status = maintained nodep libc readonly limited
- infos/description = includes constants information into kdb

20.1 Introduction

Includes constants information into the key database. The constants are defined during CMake build.

The plugin is readonly.

20.2 Usage

To mount it, use
```
kdb mount -R noresolver none user:/tests/constants constants
```

To list all constants, use:
```
kdb ls user:/tests/constants
#> user:/tests/constants
#> user:/tests/constants/cmake
#> user:/tests/constants/cmake/BINDINGS
#> user:/tests/constants/cmake/BUILD_FULL
#> user:/tests/constants/cmake/BUILD_SHARED
#> user:/tests/constants/cmake/BUILD_STATIC
#> user:/tests/constants/cmake/BUILTIN_DATA_FOLDER
#> user:/tests/constants/cmake/BUILTIN_EXEC_FOLDER
#> user:/tests/constants/cmake/BUILTIN_PLUGIN_FOLDER
#> user:/tests/constants/cmake/CMAKE_INSTALL_PREFIX
#> user:/tests/constants/cmake/ENABLE_ASAN
#> user:/tests/constants/cmake/ENABLE_DEBUG
```

```
#> user:/tests/constants/cmake/ENABLE_LOGGER
#> user:/tests/constants/cmake/GTEST_ROOT
#> user:/tests/constants/cmake/KDB_DB_DIR
#> user:/tests/constants/cmake/KDB_DB_FILE
#> user:/tests/constants/cmake/KDB_DB_HOME
#> user:/tests/constants/cmake/KDB_DB_INIT
#> user:/tests/constants/cmake/KDB_DB_SPEC
#> user:/tests/constants/cmake/KDB_DB_SYSTEM
#> user:/tests/constants/cmake/KDB_DB_USER
#> user:/tests/constants/cmake/KDB_DEFAULT_RESOLVER
#> user:/tests/constants/cmake/KDB_DEFAULT_STORAGE
#> user:/tests/constants/cmake/LIB_SUFFIX
#> user:/tests/constants/cmake/PLUGINS
#> user:/tests/constants/cmake/TARGET_CMAKE_FOLDER
#> user:/tests/constants/cmake/TARGET_DOCUMENTATION_HTML_FOLDER
#> user:/tests/constants/cmake/TARGET_DOCUMENTATION_LATEX_FOLDER
#> user:/tests/constants/cmake/TARGET_DOCUMENTATION_MAN_FOLDER
#> user:/tests/constants/cmake/TARGET_DOCUMENTATION_TEXT_FOLDER
#> user:/tests/constants/cmake/TARGET_INCLUDE_FOLDER
#> user:/tests/constants/cmake/TARGET_PKGCONFIG_FOLDER
#> user:/tests/constants/cmake/TARGET_PLUGIN_FOLDER
#> user:/tests/constants/cmake/TARGET_TEMPLATE_FOLDER
#> user:/tests/constants/cmake/TARGET_TEST_DATA_FOLDER
#> user:/tests/constants/cmake/TARGET_TOOL_DATA_FOLDER
#> user:/tests/constants/cmake/TARGET_TOOL_EXEC_FOLDER
#> user:/tests/constants/cmake/TOOLS
#> user:/tests/constants/compiler
#> user:/tests/constants/compiler/c_flags
#> user:/tests/constants/compiler/coverage
#> user:/tests/constants/compiler/cxx_flags
#> user:/tests/constants/compiler/id
#> user:/tests/constants/compiler/pic_flags
#> user:/tests/constants/compiler/static_flags
#> user:/tests/constants/macros
#> user:/tests/constants/macros/KDB_DIR_MODE
#> user:/tests/constants/macros/KDB_FILE_MODE
#> user:/tests/constants/macros/KDB_MAX_PATH_LENGTH
#> user:/tests/constants/macros/KDB_PATH_ESCAPE
#> user:/tests/constants/macros/KDB_PATH_SEPARATOR
#> user:/tests/constants/package
#> user:/tests/constants/package/cflags
#> user:/tests/constants/package/includedir
#> user:/tests/constants/package/libdir
#> user:/tests/constants/package/libs
#> user:/tests/constants/package/plugindir
#> user:/tests/constants/package/prefix
#> user:/tests/constants/package/templatedir
#> user:/tests/constants/package/tool_execdir
#> user:/tests/constants/version
#> user:/tests/constants/version/KDB_VERSION_MAJOR
#> user:/tests/constants/version/KDB_VERSION_MINOR
#> user:/tests/constants/version/KDB_VERSION_PATCH
#> user:/tests/constants/version/SO_VERSION
#> user:/tests/constants/version/SO_VERSION_GETENV
#> user:/tests/constants/version/SO_VERSION_TOOLS
#> user:/tests/constants/version/version/KDB_VERSION
```

You can unmount the plugin with:

```
kdb umount user:/tests/constants
```

Chapter 21

Plugin: counter

- infos = Information about the counter plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = tracing
- infos/needs =
- infos/recommends =
- infos/placements = pregetstorage
- infos/status = maintained nodep configurable final global nodoc
- infos/metadata =
- infos/description = counts and prints usage statistics

21.1 Introduction

Counts and prints usage statistics. Only useful for debugging the plugin framework.

21.2 Installation

See installation. The package is called `libelektra5-extra`.

21.3 Module Loading

Will not log when loaded as module (config `/module` present), unless `/logmodule` is set:

```
kdb plugin-check -c "logmodule=" counter
```

Chapter 22

Plugin: cpptemplate

- infos = Information about the cpptemplate plugin is in keys below
- infos/author = Author Name elektra@libelektra.org
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/info
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest nodep experimental concept
- infos/metadata =
- infos/description = A template for C++ based plugins

22.1 CPP Template

22.1.1 Introduction

Please use the script copy-template to create a new C++ plugin based on this template:
`scripts/dev/copy-template -p pluginname`

. For more information please take a look here

22.1.2 Installation

See installation. The package is called `libelektra5-experimental`.

22.1.3 Example

```
sudo kdb mount -R noresolver none user:/tests/cpptemplate cpptemplate some=thing config=value
# This example plugin adds configuration values at the mount point
kdb ls user:/tests/cpptemplate
#> user:/tests/cpptemplate/%
#> user:/tests/cpptemplate/config
#> user:/tests/cpptemplate/some
kdb get user:/tests/cpptemplate/config
#> value
sudo kdb umount user:/tests/cpptemplate
```

Chapter 23

Plugin: crypto

- infos = Information about crypto plugin is in keys below
- infos/author = Peter Nirschl peter.nirschl@gmail.com
- infos/licence = BSD
- infos/provides = crypto
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = unittest configurable memleak unfinished discouraged
- infos/metadata = crypto/encrypt
- infos/description = Cryptographic operations

23.1 Introduction

This plugin is a filter plugin allowing Elektra to encrypt values before they are persisted and to decrypt values after they have been read from a backend.

The idea is to provide protection of sensible values before they are persisted. This means the value of a key needs to be encrypted before it is written to a file or a database. It also needs to be decrypted whenever an admissible access (read) is being performed.

The users of Elektra should not be bothered too much with the internals of the cryptographic operations. Also the cryptographic keys must never be exposed to the outside of the crypto module.

The crypto plugin uses libgcrypt as provider of cryptographic operations.

23.2 Installation

See installation. The package is called `libelektra5-crypto`.

23.3 Dependencies

- libgcrypt20-dev or libgcrypt-devel

23.3.1 GnuPG (GPG)

GPG is a run-time dependency of the crypto plugin. Either the gpg or the gpg2 binary must be installed when using the plugin. Note that gpg2 will be preferred if both versions are available. The GPG binary can be configured in the plugin configuration as /gpg/bin (see *GPG Configuration* below). If no such configuration is provided, the plugin will look at the PATH environment variable to find the GPG binaries.

23.4 How to compile

Add "crypto" to the PLUGINS variable in CMakeCache.txt and re-run cmake.

An example CMakeCache.txt may contain the following variable:
```
PLUGINS=crypto
```

23.4.1 macOS

The crypto plugin works under macOS Sierra (Version 10.12.3 (16D32)).

To set up the build environment on macOS Sierra we recommend using Homebrew. Follow these steps to get everything up and running:
```
brew install libgcrypt pkg-config cmake
```

Also a GPG installation is required. The GPG Tools work fine for us.

23.5 Restrictions

At the moment the plugin will only run on Unix/Linux-like systems, that provide implementations for fork () and execv ().

23.6 Examples

To mount a backend with the crypto plugin that uses the GPG key 9CCC3B514E196C6308CCD230666260C14↩ A525406, use:
```
sudo kdb mount test.ecf user:/tests/t crypto "crypto/key=9CCC3B514E196C6308CCD230666260C14A525406"
```

Now you can specify a key user:/t/a and protect its content by using:
```
kdb set user:/tests/t/a
kdb meta-set user:/tests/t/a crypto/encrypt 1
kdb set user:/tests/t/a "secret"
```

The value of user:/t/a will be stored encrypted. But you can still access the original value using kdb get:
```
kdb get user:/tests/t/a
```

To unmount it you can run:
```
sudo kdb umount user:/tests/t
```

23.7 Configuration

23.7.1 GPG Configuration

The path to the gpg binary can be specified in
`/gpg/bin`

The GPG recipient keys can be specified as `encrypt/key` directly. If you want to use more than one key, just enumerate like:
```
encrypt/key/#0
encrypt/key/#1
```

If more than one key is defined, every owner of the corresponding private key can decrypt the values of the backend. This might be useful if applications run with their own user but the administrator has to update the configuration. The administrator then only needs the public key of the application user in her keyring, set the values and the application will be able to decrypt the values.

If you are not sure which keys are available to you, the `kdb` program will give you suggestions in the error description. For example you can type:
```
sudo kdb mount test.ecf user:/tests/t crypto
# RET: 7
```

In the error description you should see something like:
```
The command ./bin/kdb mount terminated unsuccessfully with the info:
The provided plugin configuration is not valid!
Errors/Warnings during the check were:
Sorry, module crypto issued the error C01310:
Failed to create handle. Reason: Missing GPG key (specified as encrypt/key) in plugin configuration.
       Available key IDs are:
       B815F1334CF4F830187A784256CFA3A5C54DF8E4,847378ABCF0A552B48082A80C52E8E92F785163F
Please report the issue at https://issues.libelektra.org/
```

This means that the following keys are available:

- B815F1334CF4F830187A784256CFA3A5C54DF8E4
- 847378ABCF0A552B48082A80C52E8E92F785163F

So the full mount command could look like this:
```
sudo kdb mount test.ecf user:/tests/t crypto "crypto/key=847378ABCF0A552B48082A80C52E8E92F785163F"
```

To unmount it you can run:
```
sudo kdb umount user:/tests/t
```

23.7.2 Cryptographic Operations

Please note that these options are meant for experts only. If you do not provide these configuration options, secure defaults are being used.

The length of the master password that protects all the other keys can be set in:
`/crypto/masterpasswordlength`

The number of iterations that are to be performed in the PBKDF2 call can be set in:
`/crypto/iterations`

23.7.3 Library Shutdown

The following key must be set to "1" within the plugin configuration, if the plugin should shut down the crypto library:
/shutdown

Per default shutdown is disabled to prevent applications like the qt-gui from crashing. Shutdown is enabled in the unit tests to prevent memory leaks.

23.8 Technical Details

23.8.1 Ciphers and Mode of Operation

The crypto plugin uses the Advanced Encryption Standard (AES) in Cipher Block Chaining Mode (CBC) with a key size of 256 bit.

Chapter 24

Plugin: csvstorage

- infos = Information about the csvstorage plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/provides = storage/csv
- infos/needs =
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = unittest nodep libc configurable limited
- infos/description = parses CSV files

24.1 Introduction

This plugin allows you to read and write CSV files using Elektra. It aims to be compatible with RFC 4180. Rows and columns are written using Elektra's arrays (`#0`, `#1`,..). By configuring the plugin you can give columns a name.

24.2 Configuration

`delimiter` Tells the plugin what delimiter is used in the file. The default delimiter is `,` and will be used if `delimiter` is not set.

`header` Tells the plugin to use the first line as a header if it is set to `colname`. The columns will get the corresponding names. Skip the first line if it is set to `skip` or treat the first line as a record if it is set to `record`. If `header` is not set, or set to `record`, the columns get named #0,#1,... (array key naming)

`columns` If this key is set the plugin will yield an error for every file that doesn't have exactly the amount of columns as specified in `columns`.

`columns/names` Sets the column names. Only usable in combination with the `columns` key. The number of subkeys must match the number of columns. Conflicts with usage of `header`.

`columns/index` Specifies which column should be used to index records instead of the record number.

`export=,export/<column name>=` Export column `column name`:

- The key `export` must be present, additionally to `export/<column name>`
- Also multiple column names can be given for different columns to export.
  - Then `delimiter` will be used as delimiter (`,` as default).
  - The order depends on the alphabetic order of the column names. Use ‘awk -F’,' 'BEGIN{OFS=","} {print $2, $1, $3}‘ or similar to reorder.
  - Unknown column names are ignored.

24.3 Examples

First line should determine the headers:

```
kdb mount test.csv /csv csvstorage \
  "delimiter=;,header=colname,columns=2,columns/names,columns/names/#0=col0Name,columns/names/#1=col1Name"
```

24.3.1 Usage

The example below shows how you can use this plugin to read and write CSV files.

```
# Mount plugin to ‘user:/tests/csv‘
# We use the column names from the first line of the
# config file as key names
sudo kdb mount config.csv user:/tests/csv csvstorage
       "header=colname,columns/names/#0=col0Name,columns/names/#1=col1Name"
# Add some data
printf ’band,album\n’                          » ‘kdb file user:/tests/csv‘
printf ’Converge,All We Love We Leave Behind\n’ » ‘kdb file user:/tests/csv‘
printf ’mewithoutYou,Pale Horses\n’            » ‘kdb file user:/tests/csv‘
printf ’Kate Tempest,Everybody Down\n’         » ‘kdb file user:/tests/csv‘
kdb ls user:/tests/csv
#> user:/tests/csv/#0
#> user:/tests/csv/#0/album
#> user:/tests/csv/#0/band
#> user:/tests/csv/#1
#> user:/tests/csv/#1/album
#> user:/tests/csv/#1/band
#> user:/tests/csv/#2
#> user:/tests/csv/#2/album
#> user:/tests/csv/#2/band
#> user:/tests/csv/#3
#> user:/tests/csv/#3/album
#> user:/tests/csv/#3/band
# The first array element contains the column names
kdb get user:/tests/csv/#0/band
#> band
kdb get user:/tests/csv/#0/album
#> album
# Retrieve data from the last entry
kdb get user:/tests/csv/#3/album
#> Everybody Down
kdb get user:/tests/csv/#3/band
#> Kate Tempest
# Change an existing item
kdb set user:/tests/csv/#1/album ’You Fail Me’
# Retrieve the new item
kdb get user:/tests/csv/#1/album
#> You Fail Me
# The plugin stores the index of the last column
# in all of the parent keys.
kdb get user:/tests/csv/#0
#> #1
kdb get user:/tests/csv/#1
#> #1
kdb get user:/tests/csv/#2
#> #1
kdb get user:/tests/csv/#3
#> #1
# The configuration file reflects the changes
kdb file user:/tests/csv | xargs cat
#> album,band
#> You Fail Me,Converge
#> Pale Horses,mewithoutYou
#> Everybody Down,Kate Tempest
# Undo changes to the key database
kdb rm -r user:/tests/csv
sudo kdb umount user:/tests/csv
```

24.4 Column as index

```
kdb mount config.csv /tests/csv csvstorage "delimiter=;,header=colname,columns/index=IMDB"
printf 'IMDB;Title;Year\n'                          » `kdb file /tests/csv`
printf 'tt0108052;Schindler's List;1993\n'          » `kdb file /tests/csv`
printf 'tt0110413;Léon: The Professional;1994\n'    » `kdb file /tests/csv`
kdb ls /tests/csv
#> user:/tests/csv/tt0108052
#> user:/tests/csv/tt0108052/IMDB
#> user:/tests/csv/tt0108052/Title
#> user:/tests/csv/tt0108052/Year
#> user:/tests/csv/tt0110413
#> user:/tests/csv/tt0110413/IMDB
#> user:/tests/csv/tt0110413/Title
#> user:/tests/csv/tt0110413/Year
kdb get /tests/csv/tt0108052/Title
#> Schindler's List
kdb rm -r /tests/csv
sudo kdb umount /tests/csv
```

24.5 Export filter

```
kdb mount config.csv /tests/csv csvstorage "delimiter=;,header=colname,columns/index=IMDB"
printf 'IMDB;Title;Year\n'                          » `kdb file /tests/csv`
printf 'tt0108052;Schindler's List;1993\n'          » `kdb file /tests/csv`
printf 'tt0110413;Léon: The Professional;1994\n'    » `kdb file /tests/csv`
kdb export /tests/csv csvstorage -c
       "delimiter=;,header=colname,columns/index=IMDB,export=,export/IMDB=,export/Title="
#> IMDB;Title
#> tt0108052;Schindler's List
#> tt0110413;Léon: The Professional
kdb export /tests/csv csvstorage -c
       "delimiter=;,header=colname,columns/index=IMDB,export=,export/IMDB=,export/Year="
#> IMDB;Year
#> tt0108052;1993
#> tt0110413;1994
kdb rm -r /tests/csv
sudo kdb umount /tests/csv
```

24.5.1 Array metakey

```
kdb mount config.csv user:/tests/csvstorage csvstorage
kdb set user:/tests/csvstorage/test test
printf 'one,two,three\nfour,five,six\n' > `kdb file user:/tests/csvstorage`
kdb ls user:/tests/csvstorage
#> user:/tests/csvstorage/#0
#> user:/tests/csvstorage/#0/#0
#> user:/tests/csvstorage/#0/#1
#> user:/tests/csvstorage/#0/#2
#> user:/tests/csvstorage/#1
#> user:/tests/csvstorage/#1/#0
#> user:/tests/csvstorage/#1/#1
#> user:/tests/csvstorage/#1/#2
kdb meta-get user:/tests/csvstorage/#0 array
#> #2
kdb rm -r user:/tests/csvstorage
kdb umount user:/tests/csvstorage
```

24.5.2 Limitations

- Does not work on file streams (e.g. `kdb import` without file)
- When using csvstorage for exporting, all parent keys must be present (see https://issues.libelektra.org/2304)

Chapter 25

Plugin: curlget

- infos = Information about the curlget plugin is in keys below
- infos/author = Thomas Waser thomas.waser@libelektra.org
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = getresolver setresolver commit rollback
- infos/status = configurable readonly preview unfinished
- infos/metadata =
- infos/description = mount remote config files via curl

25.1 Description

The `curlget` plugin is a resolver using libcurl to upload and download files from/to remote hosts. When mounted with a `URL` as configuration file there will be no changes to the file system. When mounted with a (local) path to a configuration a copy of the remote configuration is kept and used as fallback in `kdbGet()` if fetching the remote file from the server fails.

25.2 Installation

See installation. The package is called `libelektra5-curl`.

25.3 Configuration

25.3.1 definitions

URL:

an URL has to be prefixed with the protocol. valid protocols: http://, https://, ftp://, ftps://, scp://, sftp://, smb:// (currently not supported)

Filename:

can bei either an URL or a local configuration file.

if the filename is an URL the plugin operates on temporary files only and keeps no local copy of the configuration. unless specified otherwise the URL is used for both upload and download.

25.3.2 plugin configuration

- url:

  URL used for both upload and download of the configuration. might be overwritten by url/get and url/put.

- url/get:

  the URL of the remote configuration file.

- url/put:

  the URL used to upload the configuration on kdbSet

- upload/method:

  only used for HTTP requests. use POST for POST-requests, or PUT for PUT-requests.

- upload/postfield:

  for HTTP POST-requests: the name of the field containing the file

- upload/filename:

  name of the uploaded file. if present it will be appended to url/put on uploads except for HTTP POST uploads where it overrides the filename field of the header. if not specified the value defaults to url/put is assumed to be a valid upload URL already containing the filename, except for HTTP POST-requests where the basename(3) of the local mounted configuration file is used if present, or of url/get

- user:

  username for authentication

- password:

  password for authentication

- ssl/verify:

  if set 1 enforce the use of SSL. HOSTNAME and PEER verifications are enabled but might be overwritten by ssl/verify/host and ssl/verify/peer.

  if not set or set to 0 the plugin will try to use SSL but not fail if not possible.

- ssl/verify/host:

  if set to 1 use hostname verification, if set to 0, skip it.

- `ssl/verify/peer`:

  if set to `1` verify the ssl certificate, if set to `0`, skip the verification.

- `prefer`:

  if set to `local` don't update the configuration if the remote version has changed **within** succeeding `kdbGet()` calls.

- `ssh/auth`:

  ssh authentication method for sftp and scp. possible values: `agent` for using `ssh-agent`, `password` for password authentication, `pubkey` for public key authentication and `pubkeypw` for public key + password authentication.

- `ssh/key`:

  path of the private key file for ssh public key authentication. if not set, default to `$HOME/.ssh/id_dsa` or `$HOME/.ssh/id_rsa`

- `ssh/key/passwd`:

  password for the private key file

25.4 Example

```
rm /tmp/curltest.ini || $(exit 0)
sudo kdb mount -R curlget -c
       url/get="http://127.0.0.1:8000/curltest.ini",url/put="http://127.0.0.1:8000",user="thomas",password="pass",upload/metho
       /tmp/curltest.ini system:/curl ini
kdb ls system:/curl
#> system:/curl/section1
#> system:/curl/section1/key1
stat /tmp/curltest.ini
# RET:0
kdb set system:/curl/section1/key2 val2
sudo kdb umount system:/curl
stat /tmp/curltest.ini
# RET:0
cat /tmp/curltest.ini
#> [section1]
#> key1=val1
#> key2=val2
rm /tmp/curltest.ini || $(exit 0)
sudo kdb mount -R curlget -c
       url/put="http://127.0.0.1:8000",user="thomas",password="pass",upload/method="POST",upload/postfield="file"
       "http://127.0.0.1:8000/curltest.ini" system:/curl ini
kdb ls system:/curl
#> system:/curl/section1
#> system:/curl/section1/key1
#> system:/curl/section1/key2
stat /tmp/curltest.ini
# RET:1
mv /tmp/httproot/curltest.ini /tmp/httproot/curltest.ini_moved
kdb ls system:/curl
# RET:5
mv /tmp/httproot/curltest.ini_moved /tmp/httproot/curltest.ini
kdb rm system:/curl/section1/key2
sudo kdb umount system:/curl
cat /tmp/httproot/curltest.ini
#> [section1]
#> key1=val1
```

25.4.1 Mount with HTTP GET + POST and keep local copy

```
kdb mount -R curlget -c
       url/get="http://127.0.0.1:8000/curltest.ini",url/put="http://127.0.0.1:8000",user="thomas",password="pass",upload/metho
       /tmp/curltest.ini system:/curl ini
```

25.4.2 Mount with HTTP GET + POST and keep no local copys

```
kdb mount -R curlget -c
       url/put="http://127.0.0.1:8000",user="thomas",password="pass",upload/method="POST",upload/postfield="file"
       "http://127.0.0.1:8000/curltest.ini" system:/curl ini
```

25.4.3 Mount with FTP GET + PUT and keep local copy

```
kdb mount -R curlget -c
       url/get="ftp://127.0.0.1:21/test.ini",url/put="ftp://127.0.0.1:21/test.ini",user="thomas",password="pass",upload/metho
       /tmp/curltest.ini system:/curl ini
```

Chapter 26

Plugin: date

- infos = Information about the date plugin is in keys below
- infos/author = Thomas Waser thomas.waser@libelektra.org
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = presetstorage postgetstorage
- infos/status = recommended productive maintained reviewed conformant compatible coverage specific unittest tested libc nodep final
- infos/metadata = check/date check/date/format
- infos/description = validates date and time strings

26.1 Installation

See installation. The package is called `libelektra5-extra`.

26.2 Validation options

The following representation standards of dates and times are currently supported and can be use by setting `check/date` to:

- `POSIX`

  see `STRPTIME(3)` for more information. A valid format has to be specified in `check/date/format`
- `ISO8601`

  see `ISO8601`. Possible format strings specified in `check/date/format`, default: `datetime complete+truncated`:
  - Dates/Time:

    - Date:
        - `calendardate` calendar dates: day of month - month - year.
        - `weekdate` calendar week and day numbers, e.g. YYYY-Www-D
        - `ordinaldate` year + day of the year
        - `date` `calendardate`, `weekdate`, and `ordinaldate` combined
    - Time:
        - `timeofday` 24-hour timekeeyping system
        - `utc` coordinates universal time. either by appending a time-zone designator or the time difference to `UTC` to `timeofday`
    - Combined: `datetime` combination of Dates and Time according to th ISO8601 specification.
  - Representation: if no representation is specified, `complete+reduced+truncated` is used as default.
    - `complete` complete representation, dates are separated by hyphens, times by colon, e.g. YYYY-MM-DD or hh:mm:ss
    - `reduced` reduced precision, e.g. YYYY-MM, or hh
    - `truncated` truncated representation, hyphens used to indicate omitted components, e.g. –MM-DD or –ss
    - `complete+reduced+truncated` allow all 3 representations
    - `complete+reduced` allow only `complete` + `reduced` representation
    - `complete+truncated` allow only `complete` + `truncated` representation
    - `reduced+truncated` allow only `reduced` + `truncated` representation.
  - Format: if no format is specified both `basic` and `extended` are treated as valid.
    - `basic` no separating character between individual components of a `date`, `time` or `datetime` expression.
    - `extended` separating characters between components. `date` components separated by hyphen, `time` components by colon.

- `RFC2822`

  a set of possible format strings derived from rfc2822 3.3, no format string needed.

26.3 Dependencies

POSIX.1-2001

26.4 Examples

TBD

26.5 Limitations

Testing timezone designators currently only works with glibc. Unit tests using timezone designators are locale dependent.

Chapter 27

Plugin: dbus

- infos = Information about the dbus plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/maintainer = Maximilian Irlinger `max@maxirlinger.at`
- infos/licence = BSD
- infos/provides = notification
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage postcommit
- infos/status = maintained unittest libc global
- infos/description = Sends notifications for every change via D-Bus

27.1 Introduction

This plugin is a notification plugin, which sends a signal to D-Bus when the key database (KDB) has been modified.

27.2 Installation

See installation. The package is called `libelektra5-dbus`.

27.3 Dependencies

- `libdbus-1-dev`

27.4 Dbus

A preferred way to interconnect desktop applications and even embedded system applications on mobile devices running Linux is D-Bus. The idea of D-Bus accords to that of Elektra: to provide standards to let software work together more tightly. D-Bus provides a simple and lightweight IPC (Inter-Process Communication) system to be used within desktop systems. Next to RPC (Remote Procedure Call), which is not used in this plugin, it supports signals which can notify an arbitrary number of other applications about changes. Given software like a D-Bus library, notification itself is a rather easy task, but it involves additional library dependences. So it is the perfect task to be implemented as a plugin. The information about the channels to be used can be stored in the global key database.

D-Bus supports a **system-wide bus** and a **session bus**. The system configuration can be accessed by each user and the user configuration is limited to a single user. Both buses can immediately be used for the system and user configuration notification updates to get pleasing results. But, there is a problem with the session bus: It is possible within D-Bus that a user starts several sessions. The user configuration should be global to the user and is not aware of these sessions. So if several sessions are started, some of the user's processes will miss notification updates.

The namespaces are mapped to the buses the following way:

- system: system-wide bus
- user: session bus

Following signal names are used to notify about changes in the Elektra's KeySet:

- KeyAdded: a key has been added
- KeyChanged: a key has been changed
- KeyDeleted: a key has been deleted

Alternatively, (with the option announce=once) only a single message is send:

- Commit: a key has been added, changed or deleted

27.5 Usage

The recommended way is to globally mount the plugin:
```
kdb global-mount dbus
```

Alternatively one can mount the plugin additionally to a storage plugin, e.g.:
```
kdb mount file.dump / dump dbus
```

For openicc one would use (mounts with announce=once):
```
kdb mount-openicc
```

27.5.1 Shell

Then we can receive the notification events using:
```
dbus-monitor type='signal',interface='org.libelektra',path='/org/libelektra/configuration'
```

Or via the supplied test program:
```
kdb testmod_dbus receive_session
```

We can trigger a message with:
```
kdb set user:/dbus/x b
```

Note that changes in `user` fire on the dbus `session`, and changes in namespace `system` in the dbus `system` bus. To receive `system` changes we will use:
```
kdb testmod_dbus receive_system
dbus-monitor --system type='signal',interface='org.libelektra',path='/org/libelektra/configuration'
```

And then fire it with:
```
kdb set system:/dbus/y a
```

27.5.2 C

```
dbus_bus_add_match (connection,
       "type='signal',interface='org.libelektra',path='/org/libelektra/configuration'", &error);
```

See the full example here.

27.5.3 Qt

Here a small example for QDBusConnection:

Place this in your Qt class header:
```
public slots:
  void configChanged( QString msg );
```

Put this in your Qt class, e.g. the constructor:
```
if( QDBusConnection::sessionBus().connect( QString(), "/org/libelektra/configuration", "org.libelektra",
       QString(),
                                       this, SLOT( configChanged( QString ) )) )
    fprintf(stderr, "==================== Done connect\n" );
```

Here comes the org.libelektra signals:
```
void SynnefoApp::configChanged( QString msg )
{
  fprintf( stdout, "config changed: %s\n", msg.toLocal8Bit().data() );
};
```

27.5.4 Python

In Python the DBus notifications can be used as follows
```
import dbus
import gobject
gobject.threads_init()  # important: initialize threads if gobject main loop is used
from dbus.mainloop.glib import DBusGMainLoop
class DBusTest():
    def __init__(self):
        DBusGMainLoop(set_as_default=True)
        bus = dbus.SystemBus()  # may use session bus for user db
        bus.add_signal_receiver(self.elektra_dbus_key_changed_cb,
            signal_name="KeyChanged",
            dbus_interface="org.libelektra",
            path="/org/libelektra/configuration")
    def elektra_dbus_key_changed_cb(self, key):
        print('key changed %s' % key)
test = DBusTest()
loop = gobject.MainLoop()
try:
    loop.run()
except KeyboardInterrupt:
    loop.quit()
```

27.6 Background

Today, programs are often interconnected in a dense way. Such applications should always be informed when something in their environment changes. For user interactive software, notification about configuration changes is expected. The only alternative is polling, which wastes resources. It additionally is no option for interactive software, where the latency needs to be low. Instead, the software which changes the configuration has to notify all other interested applications that can reread their configuration without significant delay. In Elektra, a notification plugin ensures that a notification is actually sent on each change.

Applications can wait for such a notification with hand-written code. Bindings, however, allow for better integration. It is a common approach for toolkits to provide a main loop. Applications using such toolkits can integrate notification services into this main loop.

The actions that occur in such events are application or toolkit specific because of the non-invasive nature of Elektra. Software reacts in many different ways to update events. Hence, the frequency of update events should be kept at a minimum. Changes are kept atomic with a single attempt to write out configuration. Notification callbacks shall not change configuration because this can lead to a longer chain of unwanted modifications. That might not be true, however, if a programmer of the whole system knows that a chain of reactions will terminate. When doing such event-driven programming, care is needed to avoid infinite loops. Elektra guarantees consistency of the key database even in such cases.

27.7 Transport Plugin

Mount this plugin globally with default settings to use it as *sending* transport plugin for Elektra's notification feature:

```
kdb global-mount dbus announce=once
```

27.8 Notification Format

This plugin uses D-Bus signal messages as notifications. Notifications have the path `/org/libelektra/configuration` and the D-Bus interface `org.libelektra`. The following signal names are used:

- Commit: preferred, keys below the changed key have changed
- KeyAdded: a key has been added
- KeyChanged: a key has been changed

The first argument contains the name of the changed key. The system bus is used if the affected keys is below the `system` namespace. If the key is below the `user` namespace the session bus is used.

Example output from `dbus-monitor`:

```
signal time=1520805003.227723 sender=:1.8 -> destination=(null destination) serial=15
       path=/org/libelektra/configuration; interface=org.libelektra; member=Commit
   string "system:/tests/foo"
```

27.8.1 Problems

Key names that are not valid utf-8 cause a warning within the D-Bus library:

```
This is normally a bug in some application using the D-Bus library.
Couldn't add message argumentprocess 6139: arguments to dbus_message_iter_append_basic() were incorrect,
       assertion "_dbus_check_is_valid_utf8 (*string_p)" failed in file ../../dbus/dbus-message.c line 2676.
```

Chapter 28

Plugin: dbusrecv

- infos = Information about the dbus plugin is in keys below
- infos/author = Thomas Wahringer `waht@libelektra.org`
- infos/licence = BSD
- infos/provides = notification
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage
- infos/status = maintained unittest libc global experimental
- infos/description = Receives notifications via D-Bus

28.1 Introduction

This plugin is a notification plugin, which receives a signal from D-Bus when the key database (KDB) has been modified. It is compatible with the sending D-Bus plugin.

28.2 Installation

See installation. The package is called `libelektra5-dbus`.

28.3 Dependencies

- `libdbus-1-dev`

28.4 Usage

The recommended way is to globally mount the plugin together with the dbus plugin:
`kdb global-mount dbus dbusrecv`

This plugin is designed to be used as a transport plugin for Elektra's notification feature. If notification is not enabled (i.e. in the tool `kdb` or in any other application that does not use `elektraNotifiationContract()`) this plugin performs no actions.

This plugin cannot be directly used to receive notifications. Applications that use the notification feature implicitly use it when this plugin is mounted globally.

28.5 Transport Plugin

Mount this plugin globally with default settings to use it as *receiving* transport plugin for Elektra's notification feature:

```
kdb global-mount dbusrecv
```

For the message format please see `the dbus plugin documentation`.

Chapter 29

Plugin: desktop

- infos = Information about the desktop plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/info
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = recommended maintained specific unittest nodep libc readonly limited unfinished concept
- infos/metadata =
- infos/description = reads desktop information

29.1 Introduction

The plugin is informational and mainly be used to provide context for other configuration. See elektrify-getenv.

29.2 Installation

See installation. The package is called `libelektra5-extra`.

29.3 Usage

To mount the plugin please use:
```
sudo kdb mount --resolver noresolver none system:/info/desktop desktop
```

or it is already included if you already mounted the info plugins with:
```
kdb mount-info
```

Then you can get desktop information via:
```
kdb get system:/info/desktop
```

You either get a *lower-case* string (supported desktops see below) or no key if no desktop was detected.

29.4 Supported Desktops

Currently supported desktops are:

- GNOME
- KDE
- TDE
- Unity
- XDG conformant desktops (`XDG_CURRENT_DESKTOP`)

Currently the detection relies on environment variables, which will not work in setuid or otherwise secured binaries. Please open a bug report if the detection does not work for you: https://issues.libelektra.org

29.5 Unmount the plugin

To unmount the plugin you can run

```
sudo kdb umount system:/info/desktop
```

Chapter 30

Plugin: directoryvalue

- infos = Information about the directoryvalue plugin is in keys below
- infos/author = René Schwaiger sanssecours@me.com
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = unittest nodep preview
- infos/metadata =
- infos/description = This plugin converts directory keys to leaf keys in the set direction

30.1 Directory Value

30.1.1 Introduction

The Directory Value plugin converts

1. directory (non-leaf) keys to leaf keys in the “set” direction, and
2. converts them back to directory keys in the “get” direction.

A directory key is a key that has children. For example in the key set:

```
user:/grandparent                = Grandparent
user:/grandparent/leaf           = Leaf
user:/grandparent/parent         = Parent
user:/grandparent/parent/child   = Child
user:/mother                     = Mother
user:/mother/daughter            = Daughter
user:/mother/son                 = Son
```

the keys

```
user:/grandparent
user:/grandparent/parent
user:/mother
```

represent directory keys, while the keys

```
user:/grandparent/leaf
user:/grandparent/parent/child
user:/mother/daughter
user:/mother/son
```

are leaf keys. You can easily check this by drawing the key set in the form of a rooted tree:

```
            user
      /               \
  granparent        mother
  /      |          /    \
leaf  parent    daughter son
        |
       child
```

. The Directory Value plugin converts all directory keys to leaf keys in the “set” direction by adding new keys with the postfix `___dirdata`. Theses keys then store the old value of their parent keys

```
user:/grandparent                 =
user:/grandparent/___dirdata      = Grandparent
user:/grandparent/leaf            = Leaf
user:/grandparent/parent          =
user:/grandparent/parent/___dirdata  = Parent
user:/grandparent/parent/child    = Child
user:/mother                      =
user:/mother/___dirdata           = Mother
user:/mother/daughter             = Daughter
user:/mother/son                  = Son
```

. You might ask why we need the Directory Value plugin at all. The reason why we created this plugin is that some storage plugins like `yajl` or `yamlcpp` are only able to save values inside leaf keys. By loading the Directory Value plugin these storage plugins are also able to represent directory values properly.

30.1.1.1 Array Values

The Directory value plugin also converts array values. Let us take a look at an example key set:

```
user:/array    = Array Value
user:/array/#0 = Firt Value
user:/array/#1 = Second Value
user:/array/#2 = Third Value
```

. The plugin **does not** convert this key set into:

```
user:/array           =
user:/array/___dirdata = Array Value
user:/array/#0        = First Value
user:/array/#1        = Second Value
user:/array/#2        = Third Value
```

, since then `user:/array` **would not be an array** any more. Instead the plugin inserts a new element at index 0 with the **value prefix** `___dirdata::`

```
user:/array           =
user:/array/#0        = ___dirdata: Array Value
user:/array/#1        = First Value
user:/array/#2        = Second Value
user:/array/#3        = Third Value
```

. This way a storage plugin such as YAJL or YAML CPP are still able to store `user:/array` as an array.

30.1.1.1.1 Remarks

- The plugin only converts array parents that store **string values**
- The array metakey of the array parent (increased by one) will still be stored in the original parent key after conversion. This is important, since otherwise the storage plugin would lose information about which key represents an array.

30.1.2 Usage

To mount the plugin use the command:
```
# Mount plugin at `user:/tests/directoryvalue`
sudo kdb mount config.file user:/tests/directoryvalue directoryvalue
```

. To unmount the plugin use the command
```
sudo kdb umount user:/tests/directoryvalue
```

30.1.3 Example

```
# Mount plugin
sudo kdb mount config.file user:/tests/directoryvalue directoryvalue
# Add a directory value
kdb set user:/tests/directoryvalue/harold 'Father of SpongeBob SquarePants'
# Add a leaf value
kdb set user:/tests/directoryvalue/harold/spongebob 'I am ready!'
# Add an array
kdb set user:/tests/directoryvalue/patrick Star
kdb set user:/tests/directoryvalue/patrick/#0 'Being grown-up is boring. Besides, I don't get Jazz.'
# Elektra requires that the array parent contains the metakey `array`.
# If this key is not present, then `user:/tests/directoryvalue/patrick`
# is **not an array**.
kdb meta-set user:/tests/directoryvalue/patrick array ''
# Since the plugin converts values back in the get direction
# a user of the database will not notice any changes.
kdb ls user:/tests/directoryvalue
#> user:/tests/directoryvalue/harold
#> user:/tests/directoryvalue/harold/spongebob
#> user:/tests/directoryvalue/patrick
#> user:/tests/directoryvalue/patrick/#0
kdb get user:/tests/directoryvalue/harold
#> Father of SpongeBob SquarePants
kdb get user:/tests/directoryvalue/harold/spongebob
#> I am ready!
kdb get user:/tests/directoryvalue/patrick
#> Star
kdb get user:/tests/directoryvalue/patrick/#0
#> Being grown-up is boring. Besides, I don't get Jazz.
# Retrieve index of last element in array.
# This also works if the storage plugin does not store this index.
kdb meta-get user:/tests/directoryvalue/patrick array
#> #0
# Undo changes to the key database
kdb rm -r user:/tests/directoryvalue
sudo kdb umount user:/tests/directoryvalue
```

30.2 Limitations

- **Escaping** is currently **not possible**. If you use the Directory Value plugin you can not
  - use the name `___dirdata` as the last part of a normal key,
  - use `___dirdata:` at the beginning of a normal value in the first array element

  !

Chapter 31

Plugin: doc

- infos = Information about the doc plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/recommends =
- infos/placements = prerollback rollback postrollback getresolver pregetstorage getstorage procgetstorage postgetstorage setresolver presetstorage setstorage precommit commit postcommit
- infos/status = nodep libc experimental discouraged -1000000
- infos/metadata =
- infos/description = documentation plugin without functionality

31.1 Introduction

This plugin has no functionality. It contains documentation explaining the basic makeup of a function. `The latest version of this documentation can be found on our documentation site.`

31.2 Installation

See installation. The package is called `libelektra5-experimental`.

Chapter 32

Plugin: dpkg

- infos = Information about the dpkg plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/dpkg
- infos/placements = getstorage setstorage
- infos/status = nodep limited nodoc unfinished
- infos/description = can be used to mount dpkg files

32.1 Installation

See installation. The package is called `libelektra5-extra`.

32.2 Example

```
kdb mount /var/lib/dpkg/available system:/dpkg/available dpkg
kdb mount /var/lib/dpkg/status system:/dpkg/available dpkg
```

Chapter 33

Plugin: dump

- infos = Information about the dump plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org, Klemens Böswirth k.boeswirth+git@gmail.com
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/dump
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = productive maintained conformant unittest tested nodep -1000
- infos/metadata =
- infos/description = Dumps into a format tailored for complete KeySet semantics

33.1 Introduction

This plugin is a storage plugin that supports full Elektra semantics. Combined with a resolver plugin it already assembles a fully featured backend. No other plugins are needed.

33.2 Format

The file starts with the magic word `kdbOpen` followed by a version number (currently `2`) and a newline. The plugin can read files of version `1` and `2`, but it only writes version `2`.

After this first line, the format consists of a series of commands. The supported commands are `$key`, `$meta` and `$copymeta`. We use the `$` prefix to make the commands standout more. However, `$` does not always mean command. Keynames and values could also start with `$`, but since the plugin always knows whether a command or something else comes next, we do not need any kind of escaping.

The `$key` command creates a new key. It takes 5 arguments. The first 3 are on the same line separated by a space. The other two are on separate lines:

```
$key <type> <nsize> <vsize>
```

```
<name>
<value>
```

<type> is either string or binary and indicates what kind of value the key contains. <nsize> and <vsize> are the name size and value size respectively. For the name and for string values, the size does not include the null-terminator present in C strings. For binary keys all bytes (including any null-terminators) are counted. <name> and <value> are the keyname and key value. Because we know their length, they can contain arbitrary characters. Even newlines are allowed. The newlines between <name> and <value>, and after <value> are just to make the file more readable. They must be present, but they do not determine where <name> or <value> end.

The $meta command adds a new metakey to the last key. It is very similar to the $key command, but it only takes 4 arguments. There is no type argument, because metakeys always have string values.

```
$meta <nsize> <vsize>
<name>
<value>
```

The arguments work just like they do for $key.

Finally, there is $copymeta. It is needed, because a keyCopyMeta call results in two keys with the same metakey (equal pointers). To achieve this, we indicate which metakey should be copied from which key. The $copymeta command also takes 4 arguments:

```
$copymeta <knsize> <mnsize>
<keyname>
<metaname>
```

<keyname> is the name of the key from which the metadata is copied and <knsize> is its size (without the null-terminator). Similarly, <metaname> is the name of the metakey that is copied and <mnsize> is its size (without the null-terminator).

There is also $end. It is used to signal the end of the data to the plugin. The $end command is completely optional. Without it, the plugin will just read the file until the end. However, in streaming use $end is needed, because there is no end of the "file".

33.2.1 Format Examples

The following is an example dump file that was mounted at system:/elektra/mountpoints:

```
kdbOpen 2
$key binary 0 0
$meta 6 0
binary
$meta 7 27
comment
Below are the mount points.
$key string 4 18
dbus
serialized Backend
$key string 11 0
dbus/config
$meta 7 71
comment
This is a configuration for a backend,
see subkeys for more information
$key string 12 0
fstab/config
$copymeta 11 7
dbus/config
comment
$end
```

A few things you might have noticed:

- The first key has an empty name, because it is the root key of this mountpoint.
- The value size of 0 for the first key makes it a NULL key, but only because it is binary. The third key ($key string 11 0) also has value size 0, but is a string key. This means its value is an empty string "".
- The empty lines after $key binary 0 -1 and dbus/config are because the respective names/values are empty.
- The comment above $key string 25 0 shows that newlines in key values are completely fine, because we know what size the value has to be.

33.3 Limitations

(status -1000)

- It is quite slow

33.4 Examples

Export a KeySet using `dump`:
```
kdb export system:/example dump > example.ecf
```

Import a KeySet using `dump`:
```
cat example.ecf | kdb import system:/example dump
```

Using grep/diff or other Unix tools on the dump file. Make sure that you treat it as text file, e.g.:
```
grep --text 'mount points' example.ecf
```

Chapter 34

Plugin: email

- infos = Information about the email plugin is in keys below
- infos/author = Andreas Kraschitzer e01226853@student.tuwien.ac.at
- infos/licence = BSD
- infos/needs =
- infos/provides = check
- infos/recommends =
- infos/placements = presetstorage
- infos/status = maintained unittest nodep experimental
- infos/metadata = check/email
- infos/description = Validation for email addresses

34.1 Email Address Validation

34.1.1 Introduction

This plugin validates email addresses using regular expressions. The plugin does not support the full RFC5321 spec. For more information see Limitations below.

34.1.2 Installation

See installation. The package is called `libelektra5-experimental`.

34.1.3 Usage

```
# Mount `email` plugin to cascading namespace `/tests/email`
kdb mount config.dump /tests/email dump email
# Incorrect address is valid with incomplete configuration
kdb meta-set spec:/tests/email/noaddr check/email
kdb set user:/tests/email/noaddr invalid..address@com
# RET: 0
# Check the validity of the email stored in `/tests/email/adr`
kdb meta-set spec:/tests/email/adr check/email ""
# Set a correct email address
kdb set user:/tests/email/adr test+email@dev.libelektra.com
kdb get user:/tests/email/adr
#> test+email@dev.libelektra.com
# Try to set incorrect addresses
kdb set user:/tests/email/adr invalid..address@com
# STDERR: .*Validation Semantic.*
# ERROR:  C03200
# RET: 5
kdb set user:/tests/email/adr not.@email.com
# STDERR: .*Validation Semantic.*
# ERROR:  C03200
# RET: 5
kdb set user:/tests/email/adr @
# STDERR: .*Validation Semantic.*
# ERROR:  C03200
# RET: 5
# Undo modifications to the database
kdb rm -rf /tests/email
kdb umount /tests/email
```

34.1.4 Limitations

The plugin only checks email addresses for validity. It is not able to resolve if the host and or check if the address can receive emails. The validation does not completely support the RFC5321. The following valid email addresses are not supported:

- `(ele)ktra@elektra.io`
- `elektra@elektra.io(io)`
- `"hi@you"@elektra.io`
- `"hi you"@elektra.io`
- `" "@elektra.io`
- `"<\"\".!.#%$@elektra.io`
- `cow@[dead::beef]`
- `1@[23456789]`

There is no validation of top level domains and no length check. The following invalid email addresses will be allowed:

- `valid+part@nonexistenttopleveldomain`
- `12345678901234567890123456789012345678901234567890123456789012345678901234+x@example.↩com`

Chapter 35

Plugin: error

- infos = Information about error plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = error
- infos/needs =
- infos/recommends =
- infos/placements = presetstorage
- infos/status = productive maintained conformant shelltest tested nodep libc configurable discouraged
- infos/metadata = trigger/warnings trigger/error trigger/error/nofail
- infos/description = Provokes errors for testing the plugin framework

35.1 Introduction

Plugins (should) rarely return an error or warnings, e.g. writing the configuration basically only fails on file system problems. Such behavior is difficult to produce for tests.

This plugin tackles this issue by yielding error/warnings on request.

35.2 Installation

See installation. The package is called `libelektra5-extra`.

35.3 Usage

35.3.1 By metadata

Mount this plugin additionally with a working resolver and a storage e.g.:
```
sudo kdb mount error.dump /error error dump
```

When following metakey is present during storing (`kdbSet()`) the keyset:
```
trigger/warnings
```

a warning will be added. The plugin will still return success, but when the following metakey is present:
```
trigger/error
```

the plugin will return with an error.

The value of the metadata needs to contain the number of the requested error or warning.

So an error and warnings can be injected directly with the kdb tool. E.g. the warning number C01330:
```
kdb meta-set system:/error/key trigger/warnings C01330
```

or the error number C01200 (will not modify the KDB because `kdbSet()` will fail for the error plugin then):
```
kdb meta-set user:/error/key trigger/error C01200
# RET:5
```

When you are finished you can unmount it with:
```
sudo kdb umount /error
```

35.3.2 By config

To yield an error in kdbOpen() the metadata approach does not work. So the plugin also can yield warning/errors using configuration.

To do that, configure the plugin using:
```
on_open/warnings
on_open/error
```

E.g. you can use:
```
sudo kdb mount error.dump /error error on_open/error=C03100 dump
```

Then you get an error on any access, e.g.:
```
kdb ls system:/error
```

Will yield error C01200:
```
Description: Tried to get a key from a missing backend
Mountpoint: system:/error
```

because the opening of the plugin failed (resulting to a missing backend).

When you are finished you can unmount it with:
```
sudo kdb umount /error
```

Chapter 36

Plugin: fcrypt

- infos = Information about fcrypt plugin is in keys below
- infos/author = Peter Nirschl peter.nirschl@gmail.com
- infos/licence = BSD
- infos/provides = sync filefilter crypto
- infos/needs =
- infos/recommends =
- infos/placements = pregetstorage postgetstorage precommit
- infos/ordering = sync
- infos/status = unittest nodep configurable
- infos/metadata =
- infos/description = File Encryption

36.1 fcrypt Plugin

36.1.1 Introduction

This plugin enables file-based encryption and decryption using GPG. Also an option for signing and verifying files using GPG is provided.

This plugin encrypts backend files before the commit is executed (thus `precommit`). The plugin decrypts the backend files before the getstorage opens it (thus `pregetstorage`). After the getstorage plugin has read the backend file, the plugin decrypts the backend file again (thus `postgetstorage`).

36.1.2 Installation

See installation. The package is called `libelektra5-experimental`.

36.1.3 Security Considerations

There are two things to consider when using the `fcrypt` plugin:

1. Decrypted data is visible on the file system for a short period of time.
2. Decrypted data might end up on a hard disk or some other persistent storage.

The plugin directs GPG to write its (decrypted) output to a temporary directory. From there on the data can be processed by other plugins. After the `get` phase is over, `fcrypt` overwrites the temporary file and unlinks it afterwards. However, if the application crashes during `get` the decrypted data may remain in the temporary directory.

If the temporary directory is mounted on a hard disk, GPG writes the decrypted data on that disk. Thus we recommend to either mount `/tmp` to a RAM disk or specify another path as temporary directory within the plugin configuration (see Configuration below).

36.1.4 Known Issues

If you encounter the following error at `kdb mount`:

```
The command kdb mount terminated unsuccessfully with the info:
Too many plugins!
The plugin sync can't be positioned at position precommit anymore.
Try to reduce the number of plugins!
Failed because precommit with 7 is larger than 6
Please report the issue at https://issues.libelektra.org/
```

you might want to consider disabling the sync plugin by entering:

```
kdb set /sw/elektra/kdb/#0/current/plugins ""
kdb set system:/sw/elektra/kdb/#0/current/plugins ""
```

Please note that this is a workaround until a more sustainable solution is found.

36.1.5 Dependencies

36.1.5.1 Crypto Plugin

This plugin uses parts of the `crypto` plugin.

36.1.5.2 GnuPG (GPG)

Please refer to crypto.

36.1.6 Restrictions

Please refer to crypto.

36.1.7 Examples

You can mount the plugin with encryption enabled like this:
```
kdb mount test.ecf /t fcrypt "encrypt/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D"
```

If you only want to sign the configuration file, you can mount the plugin like this:
```
kdb mount test.ecf /t fcrypt "sign/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D"
```

Both options `encrypt/key` and `sign/key` can be combined:
```
kdb mount test.ecf /t fcrypt \
  "encrypt/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D,sign/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D"
```

If you create a key under `/t`
```
kdb set /t/a "hello world"
```

you will notice that you can not read the plain text of `test.ecf` because it has been encrypted by GPG.

But you can still access `/t/a` with `kdb get`:
```
kdb get /t/a
```

If you are looking for a more interactive example, have a look at the following ASCIIcast at:

https://asciinema.org/a/153014

36.1.8 Configuration

36.1.8.1 Signatures

The GPG signature keys can be specified as `sign/key` directly. If you want to use more than one key for signing, just enumerate like:
```
sign/key/#0
sign/key/#1
```

If more than one key is defined, every private key is used to sign the file of the backend.

If a signature is attached to a file, `fcrypt` automatically verifies its content whenever the file is being read.

Note that the signed file is stored in the internal format of GPG. So you only see binary data when opening the signed configuration file directly. However, you can simply display the plain text content of the file by using GPG:
```
gpg2 -d signed.ecf
```

36.1.8.2 GPG Configuration

The GPG Configuration is described in crypto.

36.1.8.3 Textmode

`fcrypt` operates in textmode per default. In textmode `fcrypt` uses the `--armor` option of GPG, thus the output of `fcrypt` is ASCII armored. If no encryption key is provided (i.e. only signature is requested) `fcrypt` uses the `--clearsign` option of GPG.

Textmode can be disabled by setting `fcrypt/textmode` to `0` in the plugin configuration.

36.1.8.4 Temporary Directory

`fcrypt` uses the configuration option `fcrypt/tmpdir` to generate paths for temporary files during encryption and decryption. If no such configuration option is provided, `fcrypt` will try to use the environment variable `TMPDIR`. If `TMPDIR` is not set in the environment, `/tmp` is used as default directory.

The path of the temporary directory is forwarded to GPG via the `-o` option, so GPG will output to this path. The directory must be readable and writable by the user.

We recommend to specify a path that is mounted to a RAM disk. It is advisable to set restrictive access rules to this path, so that other users on the system can not access it.

Chapter 37

Plugin: file

- infos = Information about the file plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/file
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = specific unittest tested nodep libc configurable preview experimental
- infos/metadata =
- infos/description = reads complete file into a key

37.1 Introduction

The file plugin reads the content of a file and stores it into the parent key.

37.2 Installation

See installation. The package is called `libelektra5-experimental`.

37.3 Configuration

- `binary`

  treats the file as a binary file instead of a text file

- `info`

  adds additional information about the file as metadata to the parent key.

  - `info/size` filesize
  - `info/ctime` time of last status change
  - `info/atime` time of last access
  - `info/mtime` time of last modification
  - `info/uid` user ID of owner
  - `info/gid` group ID of owner
  - `info/mode` protection
  - `info/inode` inode number

37.4 Usage

```
kdb mount file /testfile file
```

37.5 Dependencies

None.

37.6 Examples

```
# Mount the file `file/multiline` at `system:/tests/file`
sudo kdb mount "$PWD/src/plugins/file/file/singleline" system:/tests/file file info=
# Check the content of the file
kdb get system:/tests/file
#> this is a single line testfile
#  List available attributes of the mounted file
kdb meta-ls system:/tests/file
#> info/atime
#> info/ctime
#> info/gid
#> info/inode
#> info/mode
#> info/mtime
#> info/size
#> info/uid
# Check out the file's permissions
kdb meta-get system:/tests/file info/mode
# STDOUT-REGEX: 1006[46]4
# Unmount the file
sudo kdb umount system:/tests/file
```

37.7 Limitations

None.

Chapter 38

Plugin: filecheck

- infos = Information about the filecheck plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/recommends =
- infos/placements = pregetstorage precommit
- infos/status = maintained unittest tested libc configurable nodoc
- infos/metadata =
- infos/description = validates files (e.g. encoding)

38.1 Introduction

The filecheck plugin validates files. It tests: encoding, lineendings, BOM, printable characters and null bytes.

38.2 Installation

See installation. The package is called `libelektra5-extra`.

38.3 Configuration

`check/lineending` `valid/lineending` When the `check/lineending` key is present, the plugin checks the file for consistent line endings. If you want to validate for a specific line ending you can supply it with the `valid/lineending` key. Valid values are: `CR`, `LF`, `CRLF`, `LFCR`.

`check/encoding` `valid/encoding` When the `checkEncoding` key is present, the plugin validates the file encoding supplied by the key `encoding`, or, if not present, defaults to `UTF-8`

`reject/null` When the `reject/null` key is present, the plugin rejects the file if a NULL-Byte is found.

`reject/bom` When the `reject/bom` key is present, the plugin rejects the file if any BOM markers are found.

`reject/unprintable` When the `reject/unprintable` key is preset, the plugin rejects the file if an unprintable character is present (except `\r` and `\n`).

Chapter 39

Plugin: fstab

- infos = Information about FSTAB plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = storage/fstab
- infos/needs =
- infos/recommends = struct type path
- infos/placements = getstorage setstorage
- infos/status = unittest nodep difficult limited unfinished old
- infos/description = Parses files in a syntax like /etc/fstab file

39.1 Introduction

This plugin is an implementation of a parser and generator of the /etc/fstab file.

39.2 Installation

See installation. The package is called `libelektra5-extra`.

39.3 Old fstab Entries

(Deprecated, remove this section after it is reimplemented in the new way)

For each device in fstab elektra will store the following keys:

```
pseudoname/device
pseudoname/mpoint
pseudoname/type
pseudoname/options
pseudoname/dumpfreq
pseudoname/passno
```

Each represents a column in fstab.

The pseudoname can be any name for setting keys, the will be generated when getting keys, so don't expect the same name.

the directory / will be called rootfs

all swap devices will be called swapXX with a number from 00 on for XX

otherwise the mount point without the '/' character will be used.

At the other point there is the issue with the pseudonames, you can't rely on the pseudoname you have set.

The biggest issue is that you can't change or delete existing entries. All entries you set will be appended to the other filesystems.

So if you get the filesystems and change the type of the file system of the rootfs and set it again the resulting fstab will be like:

```
/dev/sda6       /               ext3>----   >----defaults,errors=remount-ro 0 1
/dev/sda6       /               jfs>----   >----defaults,errors=remount-ro 0 1
```

which will be not like you desired!

setmntent is used, so it is only conforming to BSD 4.3 and linux and you can't use any comments.

39.4 New fstab Entries

Specification:

```
[/_]
type = array
explanation = the name of the key is the mount point (so the former
  mpoint is not needed); the value is the number of entries in the
  array
[/_/#]
explanation = an entry in the array
[/_/#/device]
[/_/#/type]
[/_/#/options]
[/_/#/dumpfreq]
[/_/#/passno]
```

Example: A fstab that looks like:

```
/dev/sr0        /media/cdrom   udf,iso9660 user,noauto     0       0
```

would have a key name that is an array (so the value is the number of children, in this case 1):

```
system:/filesystems/\/media\/cdrom
```

with the array entry:

```
system:/filesystems/\/media\/cdrom0/#0/
```

So when following line is added

```
/dev/sr0        /media/cdrom   ramfs user,noauto     0       0
```

Implementation hint: use keyAddBaseName() to get escaping of /, then add array items below it

If a mount point exists more than once (that could be proc, swap or overlay mount points) the array below gets incremented (otherwise #0 is used for every unique entry).

The order of the array must, of course, be preserved. Other lines may be reordered for now, a proper "order" could be done later.

Spaces in the names are replaced by \040 in the fstab.

39.5 Example

Mount the plugin:

```
sudo kdb mount /etc/fstab system:/filesystems fstab struct type path
```

Chapter 40

Plugin: gitresolver

- infos = Information about the gitresolver plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = resolver
- infos/recommends =
- infos/placements = rollback getresolver setresolver commit
- infos/status = recommended productive reviewed conformant compatible coverage specific shelltest tested libc configurable final preview nodoc
- infos/metadata =
- infos/description = resolver for Git repositories

40.1 Description

gitresolver is a resolver that fetches from a local Git repository during the get-phase and commits them back at the end of the set-phase. It operates on a temporary copy of the latest version of your file fetched from the repository. If the temporary copy modified, a new commit with the modified version will be created. Local files won't be touched.

40.2 Installation

See installation. The package is called `libelektra5-gitresolver`.

40.3 Options

`branch` defines the branch to work on. Default: master `tracking` can be either `object` or `head` (default). if set to `object` a conflict will only occur if the file in the Git repository has been updated while you were working on it. `head` will cause a conflict if the `HEAD` commit has been updated. `pull` when present: tells the plugin to fast-forward pull the repository, fails if FF isn't possible. `checkout` when present: tells the plugin to checkout the file.

40.4 Limitations

Currently it only works inside existing Git repositories.

40.5 Examples

```
sudo kdb mount -R gitresolver /path/to/my/gitrepo/file.ini system:/gittest ini shell \
    execute/set='cd /path/to/my/gitrepo/ && git commit --amend'
```

Chapter 41

Plugin: glob

- infos = Information about glob plugin is in keys below
- infos/author = Felix Berlakovich elektra@berlakovich.net
- infos/licence = BSD
- infos/provides = apply
- infos/needs =
- infos/recommends =
- infos/ordering = check keytometa
- infos/stacking = no
- infos/placements = presetstorage postgetstorage
- infos/status = unittest nodep libc configurable difficult
- infos/description = copies metadata to keys with the help of globbing

41.1 Introduction

The glob plugin provides coping metadata given by the plugin's configuration to keys identified using *glob expressions*. Globbing resembles regular expressions. They do not have the same expressive power, but are easier to use. The semantics are more suitable to match path names:

- * matches any key name of just one hierarchy. This means it complies with any character except slash or null.
- ? satisfies single characters with the same exclusions.
- Additionally, there are ranges and character classes. They can also be inverted.

So this plugin adds metadata to keys identified by globbing expressions. The plugin copies the metadata of the corresponding globbing keys in its configuration. Globbing can be applied in get and set direction or both.

41.2 Globbing Keys

The plugin is configured with globbing keys in its configuration. Each key below the configuration is interpreted as a globbing key. The value of the key contains the globbing expression. When a key matching the glob expression contained in one of the globbing keys is found, the metakeys of the corresponding globbing key are copied. Once a match is found, no further keys will be considered for globbing. The reason for this are catch all globbing keys that can be used to match all keys that have not been matched by a preceding globbing key.

41.2.1 Globbing Direction

Globbing keys located directly below the configuration (e.g `config/glob/#1`) are applied in both directions (get and set). Keys below "get" (e.g. `config/glob/get/#1`) are applied only in the get direction and keys below set (e.g. `config/glob/set/#1`) are applied only in the set direction.

So the glob plugin iterates over a list of glob expressions for every key. Metadata is applied only for the first expression that matches. So later expressions can be used as default values.

41.2.2 Globbing Flags

Globbing keys may contain a subkey named "flags". This optional key contains the flags to be passed to the globbing function (currently fnmatch) as a comma separated list. Unknown flag names will be ignored. The allowed flag names are

- "noescape" which enables the FNM_NOESCAPE flag
- "pathname" which enables the FNM_PATHNAME flag
- "period" which enables the FNM_PERIOD flag

If the flag key does not exist, FNM_PATHNAME is used as a default (see fnmatch(3) for more details). An empty string disables all flags (i.e. also the default flag).

41.3 Contracts

Glob statements are very useful together with contracts. Storage plugins can request the glob plugin to fill up metadata before they receive the keys in `elektraPluginSet()`. In `config/needs`, the plugin declares which keys should obtain which metadata. If the glob expression starts with a slash, the contract checker will automatically prepend the mount point.

For example, the hosts plugin contract contained:

```
ksNew (30,
  // ...
  keyNew ("system:/elektra/modules/hosts/config/needs/glob/#1",
      KEY_VALUE, "/*",
      KEY_META, "check/ipaddr", "", /* Preferred way to check */
          /* Can be checked additionally */
      KEY_META, "check/validation", "^[0-9.:]+$",
      KEY_META, "check/validation/message",
          "Character present not suitable for ip address",
      KEY_END),
  keyNew ("system:/elektra/modules/hosts/config/needs/glob/#2",
      KEY_VALUE, "/*/*",
          /* Strict character validation */
      KEY_META, "check/validation", "^[0-9a-zA-Z.:]+$",
      KEY_META, "check/validation/message",
          "Character present not suitable for host address",
```

```
      KEY_END),
  // ...
);
```

We see that the `hosts` plugin added two glob statements with the clause `config/needs`. The first one matches with hostnames, the second with aliases.

The glob plugin only fills the metadata in `kdbSet()`. This makes a difference compared with adding the metadata already in `kdbGet()`. Using the glob plugin, the user will not see the metadata, but later plugins in `kdbSet()` will.

To sum up, the glob plugin replenishes the keys with metadata. The plugin applies metadata in a flexible way. This metadata can be used for later checks. Limited configuration storage plugins, like the `hosts` plugin, use this feature. They need it because they are not able to store metadata themselves. It is obviously not possible to apply values to non-existing keys.

Chapter 42

Plugin: gopts

- infos = Information about the gopts plugin is in keys below
- infos/author = Klemens Böswirth `k.boeswirth+git@gmail.com`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = hook procgetstorage
- infos/status = recommended productive maintained unittest nodep libc configurable
- infos/metadata = args args/index command opt opt/long opt/arg opt/flagvalue opt/help opt/hidden opt/# opt/#/long opt/#/arg opt/#/flagvalue opt/#/hidden env env/#
- infos/description = Parses command-line options using elektra-opts

42.1 Introduction

This plugin allows applications to access command-line options and environment variables via the KDB.

It is implemented as a simple frontend for the parser implemented in the internal `elektraGetOpts`.

For more information on how to write the necessary specification and on using command-line options in general, take a look at the dedicated tutorial

Depending on the calling context and configuration, this plugin might use operating system specific functions to determine command-line arguments and environment variables. Which operating system's functions the plugin uses is determined at compile-time.

> **Note:** One of the system-specific implementations of this plugin relies on procfs. Therefore, if you compile the plugin on a system with procfs, the plugin may not work on other machine with the same OS or even on the same machine, if procfs is not mounted.

42.2 Installation

See installation. The package is called `libelektra5-experimental`.

42.3 Basic Usage

The preferred way of using this plugin is via a `kdbOpen` contract:

```
KeySet * contract = ksNew (0, KS_END);
elektraGOptsContract (contract, argc, argv, environ, parentKey, NULL);
KDB * kdb = kdbOpen (contract, parentKey);
// gopts automatically mounted
KeySet * ks = ksNew (0, KS_END);
kdbGet (kdb, ks, parentKey);
```

42.4 Configuration Keys

The plugin accepts a number of configuration keys.

- `/offset`: You can set this key to an integer `n`. The plugin will then ignore the first `n` command-line arguments and only pass the rest on to the parser.
- `/help/usage`: The value of this key is used to replace the standard usage line in the auto-generated help message.
- `/help/prefix`: The value of this key is inserted between the usage line and the options list in the auto-generated help message.

42.5 Global KeySet

The plugin also takes part of its configuration from the global KeySet. All keys the plugin uses are below `system:/elektra/gopts`. This prefix is abbreviated to `//` below.

This plugin may use the following keys from the global KeySet:

- `//parent`: If present, the plugin will use this key instead of the one provided by `kdbGet` as the parent key pass on to the parser. Specifically, the plugin uses this keys value as the key name for a new key that is passed to the parser.
- `//argc`: If present and `//parent` is present as well, the plugin expects `//argv` to be present as well. This key must be binary and its value must be an `int`.
- `//argv`: If present and `//parent` is present as well, the plugin expects `//argc` to be present as well. The values of `//argc` and `//argv` will be used instead of using the OS specific implementation. This key must be binary and its value must be a `const char * const *`.
- `//envp`: If present and `//parent` is present as well, the value of this key will be used as the list of environment variables. This key must be binary and its value must be a `const char * const *`.
- `//args`: If present and `//parent` is present as well, but `//argc` and `//argv` are absent, this will be used as the list of command-line arguments. This key must be binary and its value must be a zero-byte separated (and terminated) list of strings.
- `//env`: If present and `//parent` is present as well, but `//envp` is absent, this will be used as the list of environment variable. This key must be binary and its value must be a zero-byte separated (and terminated) list of strings.

Chapter 43

Plugin: gpgme

- infos = Information about gpgme plugin is in keys below
- infos/author = Peter Nirschl peter.nirschl@gmail.com
- infos/licence = BSD
- infos/provides = crypto
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = unittest configurable memleak experimental unfinished
- infos/metadata = crypto/encrypt gpg/binary
- infos/description = Cryptographic operations wit GnuPG Made Easy (GPGME)

43.1 Introduction

The `gpgme` plugin is a filter plugin that enables users to encrypt values before they are persisted and to decrypt values after they have been read from a backend. The encryption and decryption is designed to work transparently.

The cryptographic operations are performed by GnuPG via the `libgpgme` library.

43.2 Installation

See installation. The package is called `libelektra5-experimental`.

43.3 Dependencies

- `libgpgme11` version 1.10 or later

43.4 Build Information

The plugin has been tested on Ubuntu 18.04 with `libgpgme` version 1.10.

43.5 Examples

You can mount the plugin like this:
```
kdb mount test.ecf /t gpgme "encrypt/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D"
```

Now you can specify a key `user:/t/a` and protect its content by using:
```
kdb set user:/t/a
kdb meta-set user:/t/a crypt/encrypt 1
kdb set user:/t/a "secret"
```

The value of `user:/t/a` (for this example: "secret") will be stored encrypted. You can still access the original value by using `kdb get`:
```
kdb get user:/t/a
```

43.6 Configuration

43.6.1 GnuPG Keys

The GPG recipient keys can be specified in two ways:

1. The GPG recipient key can be specified as `encrypt/key` directly.
2. If you want to specify multiple keys, you can enumerate them under `encrypt/key`.

The following example illustrates how multiple GPG recipient keys can be specified:
```
encrypt/key/#0
encrypt/key/#1
```

43.6.2 Textmode

`gpgme` operates in textmode per default. In textmode the output of GPG is ASCII armored.

Textmode can be disabled by setting `/gpgme/textmode` to `0` in the plugin configuration.

43.7 Technical Details

43.7.1 Message Format

The encrypted values are valid PGP messages, that can be decrypted and read solely by the GnuPG binary without Elektra.

Chapter 44

Plugin: hexcode

- infos = Information about hexcode plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = code
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest nodep libc configurable
- infos/description = Decoding/Encoding engine which escapes unwanted characters.

44.1 Introduction

This code plugin translates each unwanted character into a two cypher hexadecimal character. The escape character itself always needs to be encoded, otherwise the plugin would try to interpret the following two characters in the text as a hexadecimal sequence.

44.2 Installation

See installation. The package is called `libelektra5-extra`.

44.3 Restrictions

- The escape character itself always needs to be encoded, otherwise the plugin would try to interpret the following two characters in the text as a hexadecimal sequence.
- The length of the resulting string increases. In the worst case the hexcode plugin makes the value three times larger.

44.4 Example

Consider the following *value* of an key:
```
value=abc xyz
```

Assuming the escape character is % the input would be encoded to:
```
value%3Dabc%20xyz
```

The disadvantage is that the length of the resulting string increases. In the worst case the hexcode plugin makes the value three times larger.

44.5 Usage

Add `hexcode` to `infos/needs` for any plugin that you want to be filtered by hexcode.

Then, additionally define all characters you need to be escaped below `config/needs/chars` in your contract, e.g:
```
config/needs/chars/20 = 61
```

to transform a space (dec 20) to the escaped letter a (dec 61).

The escape letter itself can be changed by setting:
```
config/needs/escape
```

Chapter 45

Plugin: hexnumber

- infos = Information about the hexnumber plugin is in keys below
- infos/author = Klemens Böswirth `k.boeswirth+git@gmail.com`
- infos/licence = BSD
- infos/needs =
- infos/provides = conv check
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest nodep configurable
- infos/metadata = unit/base
- infos/ordering = type
- infos/description = converts hexadecimal values into decimal and back

45.1 Introduction

This plugin is used to read configuration files that use hexadecimal values. All "hex-values" (see below) will be converted into decimal when Elektra reads values from the mounted file. When Elektra writes back to the file the converted values will be converted back and stored as before (`0X` will be replaced with `0x`).

45.2 Installation

See installation. The package is called `libelektra5-extra`.

45.2.1 What are "hex-values"?

There are multiple ways you can signal to the hexnumber plugin, that a value should be converted:

1. If a Key has the metadata `unit/base` set to `hex` it will always be interpreted as a hex-value. The plugin will also produce an error, if the value contained in such a Key does not start with `0x` (or `0X`).
2. If `unit/base` is not present and
   - the `type` metadata is set to one of the recognized integer-types (default: `byte`, `short`, `unsigned_short`, `long`, `unsigned_long`, `long_long`, `unsigned_long_long`)
   - AND the configuration value itself starts with `0x` (or `0X`) it will be interpreted as a hex-value.
3. If forced conversion mode (`/force` plugin configuration, see below) is enabled all values starting with `0x` (or `0X`) are considered hex-values.

45.3 Configuration

When mounting a backend with the hexnumber plugin, a few settings can be configured.

1. To enable forced conversion mode set `/force` to any value. In forced conversion mode the plugin tries to convert **ALL** strings starting with `0x` (or `0X`) into decimal before passing the value on to the rest of Elektra. This can be useful for importing a configuration file that uses hexadecimal values into Elektra without writing a specification for the file.

   NOTE: be careful when using this option, as any configuration value that contains invalid non-hexadecimal characters (i.e. does not match `0[xX][0-9A-Fa-f]+`) will result in an error.

```
sudo kdb mount test.ecf /examples/hexnumber/forced hexnumber /force=1
```

1. The types recognized as integers can be configured. For this purpose specify all *additional* types you want to be considered for possible hexadecimal conversion as an Elektra array. All keys with a type from `/accept/types/#`, or one of the default types, will be converted to hexadecimal if the value starts with `0x` (or `0X`).

```
sudo kdb mount test.ecf /examples/hexnumber/customtypes hexnumber /accept/types/#0=customint
       /accept/types/#1=othercustomint
```

45.4 Usage & Example

- To mount a simple backend that uses hexadecimal numbers, you can use:
  `sudo kdb mount test.ecf user:/tests/hexnumber hexnumber`
- A few examples on how to use the plugin:

```
# Example 1: read hex value
kdb set user:/tests/hexnumber/hex 0x1F
kdb meta-set user:/tests/hexnumber/hex type long
kdb get user:/tests/hexnumber/hex
#> 31
# Example 2: decimal value not converted
kdb set user:/tests/hexnumber/dec 26
kdb meta-set user:/tests/hexnumber/dec type long
kdb get user:/tests/hexnumber/dec
#> 26
# Example 3: string untouched
kdb set user:/tests/hexnumber/string value
kdb meta-set user:/tests/hexnumber/string type string
kdb get user:/tests/hexnumber/string
#> value
# Example 4: read hex value with unit/base
kdb set user:/tests/hexnumber/hex2 0xF
kdb meta-set user:/tests/hexnumber/hex2 unit/base hex
kdb get user:/tests/hexnumber/hex2
#> 15
# Undo changes
kdb rm -r user:/tests/hexnumber
```

- To unmount the plugin use the following command:
```
kdb umount user:/tests/hexnumber
#>
```

Chapter 46

Plugin: hosts

- infos = Information about hosts plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = storage/hosts
- infos/needs =
- infos/recommends = glob error network
- infos/placements = getstorage setstorage
- infos/status = maintained unittest nodep libc limited
- infos/metadata = order comment/# comment/#/start comment/#/space
- infos/description = This plugin reads and writes /etc/hosts files.

46.1 Introduction

The `/etc/hosts` file is a simple text file that associates IP addresses with hostnames, one line per IP address. The format is described in `hosts(5)`.

The `hosts` plugins transforms the information of this file to the following structure. The keys directly below `ipv4` or `ipv6` are host names of IPv4 or IPv6 addresses, respectively. The keys directly below these keys are aliases. The IP addresses themselves are stored as values.

46.2 Special values

46.2.1 Hostnames

Canonical hostnames are stored as key base names with their IP addresses as value.

46.2.2 Aliases

Aliases are stored as keys directly below canonical hostnames with a read-only duplicate of the associated IP address as value.

46.2.3 Comments

Comments are stored according to the comment metadata specification (see `/doc/METADATA.ini` for more information).

46.2.4 Ordering

The ordering of the hosts is stored in metakeys of type `order`. The value is an ascending number. Ordering of aliases is *not* preserved.

46.3 Examples

Mount the plugin:
```
sudo kdb mount --with-recommends /etc/hosts system:/hosts hosts
```

Print out all known hosts and their aliases:
```
kdb ls system:/hosts
```

Get IP address of ipv4 host "localhost":
```
kdb get system:/hosts/ipv4/localhost
```

Check if a comment is belonging to host "localhost":
```
kdb meta-ls system:/hosts/ipv4/localhost
```

Try to change the host "localhost", should fail because it is not an IPv4 address:
```
sudo kdb set system:/hosts/ipv4/localhost ::1
# Backup-and-Restore:/tests/hosts
sudo kdb mount --with-recommends hosts /tests/hosts hosts
mkdir -p $(dirname `kdb file user:/tests/hosts`)
# Create hosts file for testing
echo '127.0.0.1 localhost' >  `kdb file user:/tests/hosts`
echo '::1       localhost'          » `kdb file user:/tests/hosts`
# Check the file
cat `kdb file user:/tests/hosts`
#> 127.0.0.1    localhost
#> ::1  localhost
# Check if the values are read correctly
kdb get /tests/hosts/ipv4/localhost
#> 127.0.0.1
kdb get /tests/hosts/ipv6/localhost
#> ::1
# Should both fail with error C03200 and return 5
kdb set /tests/hosts/ipv4/localhost ::1
# RET:5
# ERROR:C03200
kdb set /tests/hosts/ipv6/localhost 127.0.0.1
# RET:5
# ERROR:C03200
# cleanup
kdb rm -r /tests/hosts
sudo kdb umount /tests/hosts
```

46.4 Limitations

- host names are not validated, see `https://issues.libelektra.org/2185`
- duplicates, where a host names is the same as an alias name, are not rejected, see `https://issues.libelektra.org/3461`
- keys that do not confirm to the hierarchy are ignored

Chapter 47

Plugin: iconv

- infos = Information about iconv plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = conv
- infos/needs =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest libc
- infos/description = Converts values of keys between charsets

47.1 Introduction

This plugin is a filter plugin that converts between different character encodings.

47.2 Installation

See installation. The package is called `libelektra5-extra`.

47.3 Purpose

Consider a user insisting on a `latin1` character encoding because of some old application. All other users already use, for example, `UTF-8`. For these users, the configuration files are encoded in `UTF-8`. So we need a solution for the user with `latin1` to access the key database with proper encoding.

On the other hand, contemplate an XML file which requires a specific encoding. But the other key databases work well with the users encoding. So a quick fix for that backend is needed to feed that XML file with a different encoding.

The iconv plugin provides a solution for both scenarios. It converts between many available character encodings. With the plugin's configuration the user can change the from and to encoding. The default values of the plugin configuration are: `from` encoding will be determined at run-time. `to` encoding is `UTF-8`.

Parameters:

- `to` is per default UTF-8, to have unicode configuration files
- `from` is per default the users locale

Note that for writing the configuration `from` and `to` is swapped. A key database that requires a specific encoding can make use of it. To sum up, every user can select a different encoding, but the key databases are still properly encoded for anyone.

47.4 Example

For example `iconv/iconv.ini` should be `latin1`, but all users have `UTF-8` settings:

```
# Mount the file `iconv/iconv.ini` using the `mini` plugin together with `iconv`
sudo kdb mount "$PWD/src/plugins/iconv/iconv/iconv.ini" system:/tests/iconv mini iconv
       from=UTF-8,to=ISO-8859-1
# Check the file type of the mounted file
file -b "`kdb file system:/tests/iconv`"
#> ISO-8859 text
kdb get system:/tests/iconv/a         # converts ISO-8859 to UTF-8
#> hellö
kdb set system:/tests/iconv/a öäß     # converts UTF-8 to ISO-8859
kdb get system:/tests/iconv/a
#> öäß
# Cleanup
kdb set system:/tests/iconv/a hellö
sudo kdb umount system:/tests/iconv
```

Chapter 48

Plugin: internalnotification

- infos = Plugin for internal notification
- infos/author = Thomas Wahringer `waht@libelektra.org`
- infos/maintainer = Maximilian Irlinger `max@maxirlinger.at`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = postgetstorage postcommit
- infos/status = unittest libc nodep configurable global experimental unfinished nodoc concept
- infos/metadata =
- infos/description = Plugin for internal notification

48.1 Usage

Allows applications to automatically update registered variables when the value of a specified key has changed.

Application developers should use the `notification API` instead of the functions exported by this plugin. The API is easier to use and decouples applications from this plugin.

48.2 Exported Functions

This plugin exports various functions starting with `register*` below `system:/elektra/modules/internalnotificati`
These functions should not be used directly. Instead the `notification API` should be used.

Chapter 49

Plugin: ipaddr

- infos = Information about the ipaddr plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = check
- infos/recommends =
- infos/placements = presetstorage
- infos/status = maintained unittest nodep
- infos/metadata = check/ipaddr
- infos/description = Validation for IP addresses

49.1 IP Address Validation

49.1.1 Introduction

This plugin validates IP addresses using regular expressions.

49.1.2 Installation

See installation. The package is called `libelektra5-experimental`.

49.1.3 Usage

```
# Mount `ipaddr` plugin to cascading namespace `/tests/ipaddr`
kdb mount config.dump /tests/ipaddr dump ipaddr
# Check the validity of the IP stored in `system:/tests/ipaddr/ipv4`
kdb meta-set spec:/tests/ipaddr/ipv4 check/ipaddr ipv4
# Try to set an incorrect IP address
kdb set system:/tests/ipaddr/ipv4 127.0.0.1337
# STDERR: .*Validation Semantic.*
# ERROR:  C03200
# RET:    5
# Set a correct IPv4 address
kdb set system:/tests/ipaddr/ipv4 127.0.0.1
kdb get system:/tests/ipaddr/ipv4
#> 127.0.0.1
# By default the plugin allows both IPv4 and IPv6 addresses
kdb meta-set spec:/tests/ipaddr/address check/ipaddr ""
# Set correct IP addresses
kdb set system:/tests/ipaddr/address 1.2.3.4
kdb set system:/tests/ipaddr/address ::1
# Try to set incorrect addresses
kdb set system:/tests/ipaddr/address bad::ip
# RET: 5
kdb set system:/tests/ipaddr/address 1.2.-3.4
# RET: 5
# Undo modifications to the database
kdb rm -r /tests/ipaddr
kdb umount /tests/ipaddr
```

49.1.4 Limitations

The plugin only checks IP addresses for validity. It is not able to resolve hostnames. If you are looking for a plugin that supports hostnames, check out the network plugin.

Chapter 50

Plugin: iterate

- infos = Information about the iterate plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/placements = presetstorage postgetstorage
- infos/status = unittest nodep libc experimental unfinished nodoc concept
- infos/description = conditionally calls exported functions

50.1 Installation

See installation. The package is called `libelektra5-experimental`.

50.2 Usage

Suppose you have a plugin bar that exports the function `foo(Key *k)`. Then you can mount:

```
kdb mount file.dump /example/iterate dump iterate when=bar foo Key
```

Which will execute `foo(k)` for every key that has the metadata `when`.

Chapter 51

Plugin: jni

- infos = Information about the jni plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest configurable global memleak experimental
- infos/description = generic Java plugin

51.1 Introduction

Allows you to write plugins in Java.

This plugin needs the JNA bindings to work. Furthermore, it requires Java 11 or later.

While the plugin internally uses JNI (thus the name), the Java binding for your Java plugin may use something different, e.g. JNA. The requirements for the Java bindings are:

- needs to have the classes `elektra/Key` and `elektra/KeySet` with
  - a constructor that takes a C-Pointer as long (J)
  - a method "release" that gives up ownership (set internal pointer to NULL)

The Java plugin itself needs to have the following methods:

- constructor without arguments (i.e. default constructor)
- open with argument `elektra/KeySet` (the plugin's conf) and `elektra/Key`
- close with argument `elektra/Key`
- get with arguments `elektra/KeySet` and `elektra/Key`
- set with arguments `elektra/KeySet` and `elektra/Key`
- error with arguments `elektra/KeySet` and `elektra/Key`

51.2 Installation

See installation. The package is called `libelektra5-java`. To actually mount plugins, you will additionally need `java-elektra`. Furthermore, at least JNA version 5.5 is required.

51.3 Plugin Config

You need to pass :

- classname the classname to use as plugin, e.g. `elektra/plugin/Echo`
- classpath the classpath where to find JNA, the package elektra and other classes needed

Additionally, you can set:

- option allows you to pass an option to the jvm, default: `-verbose:gc,class,jni`
- ignore allows you to ignore broken options, default: `false`
- print allows you to print java exceptions for debugging purposes

If Elektra and a recent jna.jar (adapt path below) is already installed, following should output some debug logs and this README:

```
kdb plugin-info -c
       classname=org/libelektra/plugin/Echo,classpath=.:/usr/share/java/jna.jar:/usr/share/java/libelektra.jar,print=
       jni
```

> Note: The Java implementation of the plugin can request any other additional plugin configuration, read about it in the end of the output of plugin-info. Plugins dynamically append text after the end of this page.

You can also mount plugins (see open issues):

```
kdb mount -c
       classname=org/libelektra/plugin/PropertiesStorage,classpath=.:/usr/share/java/jna.jar:/usr/share/java/libelektra.jar,p
       file.properties /jni jni
       classname=org/libelektra/plugin/PropertiesStorage,classpath=.:/usr/share/java/jna.jar:/usr/share/java/libelektra.jar,p
```

51.4 Compiling

If you do not want to use pre-compiled versions, you can compile the plugin yourself. Start by enabling the plugin using (`ALL;-EXPERIMENTAL` is default):

```
cmake -DPLUGINS="ALL;-EXPERIMENTAL;jni" /path/to/libelektra
```

51.4.1 on Debian 10 / Ubuntu 20.04 LTS

Install package `openjdk-11-jdk` and make sure that no older Java versions are present in `/usr/lib/jvm`.

If you have manually installed Java, you might get errors related to `Could NOT find JNI` during `cmake`. In this case, please consider setting your `JAVA_HOME` environment variable accordingly.

51.4.2 on macOS

Older macOS include an old apple specific version of Java, based on 1.6. However, for the jni plugin JDK version 11 is required, so either the openjdk or the oracle jdk has to be installed.

For example, install oracle's jdk11 via their provided installer. After that, you have to set the JAVA_HOME environment variable to the folder where the jdk is installed, usually like

```
export JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home/"
```

As macOS handles linked libraries differently, there is no ldconfig command. Instead you can export an environment variable to tell Elektra the location of Java's dynamic libraries.

```
export
       DYLD_FALLBACK_LIBRARY_PATH="/Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home/jre/lib:/Library/Java/JavaVirtua
```

Afterwards, the jni plugin should be included in the build and compile successfully.

51.4.3 Running the JNI test

Make sure to run the test after compiling the plugin. Change to your Elektra's build folder and execute the following command for running the JNI plugin test and verify it works:

```
ctest -V -R testmod_jni
```

51.4.4 Troubleshooting

If it should still not find the correct jni version, or says the jni version is not 11, then it most likely still searches in the wrong directory for the jni header file. It has been experienced that if the project has been built already without this environment variable set, the Java location is cached. As a result, it will be resolved wrong in future builds, even though the environment variable is set. To resolve this, it should be enough to delete the CMakeCache.txt file in the build directory and reconfigure the build.

51.5 Development

To know how the methods of your class are called, use:

```
javap -s YourClass
```

Also explained here

JNI Functions Invocation

51.6 Open Issues

- Only a single Java plugin can be loaded
- Java plugins cannot be used from an Java application
- If this plugin is enabled, valgrind detects memory problems even if the plugin is not mounted.

Chapter 52

Plugin: journald

- infos = Information about journald plugin is in keys below
- infos/author = Felix Berlakovich `elektra@berlakovich.net`
- infos/licence = BSD
- infos/provides = logging
- infos/needs =
- infos/placements = pregetstorage postcommit postrollback
- infos/status = maintained libc global nodoc
- infos/description = logging of committed and rolled back keys via systemd-journal

52.1 Introduction

The plugin logs successful and failed write attempts via the systemd journal daemon (systemd-journal). See the `systemd-journal man page` for more information about systemd-journal. Errors are reported with priority 3 (error priority) and use the message ID `fb3928ea453048649c61d62619847ef6`. Successful writes are reported with priority 5 (notice priority) and use the message ID `fc65eab25c18463f97e4f9b61ea31eae`.

Configure the plugin with `log/get=1` to enable logging when configuration is loaded. For example, `kdb gmount journald log/get=1`.

52.2 Installation

See installation. The package is called `libelektra5-journald`.

52.3 Dependencies

- `libsystemd-dev` (also called `libsystemd-journal-dev`)

Chapter 53

Plugin: kconfig

- infos = Information about the kconfig plugin is in keys below
- infos/author = Dardan Haxhimustafa `mail@dardan.im`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/kconfig
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = recommended maintained compatible specific unittest nodep experimental unfinished concept
- infos/metadata =
- infos/description = Reads and writes the KConfig INI format

53.1 Introduction

This plugin can be used to parse and serialize a `KConfig` INI file.

Information about the syntax:

- Files are expected to be encoded in `UTF-8`.
- Empty lines are ignored.
- Lines that start with a `#` character are considered comments. Comments are ignored too.
- Configurations consist of groups and keys. Only keys can have values.
- Key names can't start with a `[` character
- Keys can contain spaces and any special characters except `=`.
- If a key has a value, then it will be followed with an `=` symbol and then the value will be read until the end of the line. The white space characters around the `=` symbol are ignored.
- In values, the following escape sequences can be used:

  - \n and \r are mapped to newline
  - \t is mapped to tab
  - \\ is mapped to \

- Values can contain any character from the UTF-8 set except for newline and \ followed by an invalid escape sequence.
- Keys can be localized. The locale is surrounded with [and] and cannot start with $.
- Same key names can be used multiple times if it has different locales. The following example is valid:
```
greeting[en] = Hello
greeting[de] = Hallo
```
- Keys can have metadata. Those are one byte long, start with $ and are surrounded with [and].
- The same key name can't be used multiple times with different metadata (different to locales). The following example is invalid:
```
key.name[$a] = Something
key.name[$i] = Something else
```
- Group names begin have a [symbol at the beginning of a line and every key that follows them is part of this group (until the next group is declared)

An example of how a valid config file might look like:
```
[group][subgroup]
key.name[en][$i][$e]=Key Value
key.name[de]=Key Wert
```

And how it will be represented in kdb:
```
keyNew (PREFIX "group/subgroup/key.name[en]", KEY_VALUE, "Key Value", KEY_META, "kconfig", "ie", KEY_END)
keyNew (PREFIX "group/subgroup/key.name[de]", KEY_VALUE, "Key Wert", KEY_END)
```

53.2 Usage

The following example shows you how you can read data using this plugin.
```
# Mount the plugin to the cascading namespace `/tests/kconfig`
sudo kdb mount configrc /tests/kconfig kconfig
# Manually add a key-value pair to the database
mkdir -p "$(dirname "$(kdb file user:/tests/kconfig)")"
echo 'key=Value' > "$(kdb file user:/tests/kconfig)"
# Retrieve the new value
kdb get /tests/kconfig/key
#> Value
# Set the value to Example
kdb set /tests/kconfig/key Example
# Verify that the value has changed in the file too
cat `kdb file user:/tests/kconfig`
#> key=Example
# Manually add a gorup to the database
echo '[group][subgroup]' >> "$(kdb file user:/tests/kconfig)"
# Manually add a key that contains metas to that group
echo 'key.name[$a][$i]=New Value' >> "$(kdb file user:/tests/kconfig)"
# Retrieve the new value
kdb get /tests/kconfig/group/subgroup/key.name
#> New Value
# Retrieve the meta values
kdb meta-get /tests/kconfig/group/subgroup/key.name kconfig
#> ai
# Manually add a group and a localized key
echo '[localized keys]' >> `kdb file user:/tests/kconfig`
echo 'greeting[en]=Hello' >> `kdb file user:/tests/kconfig`
echo 'greeting[de]=Hallo' >> `kdb file user:/tests/kconfig`
# Retrieve the english greeting
kdb get '/tests/kconfig/localized keys/greeting[en]'
#> Hello
# Retrieve the german greeting
kdb get '/tests/kconfig/localized keys/greeting[de]'
#> Hallo
# Undo modifications to the database
sudo kdb umount /tests/kconfig
```

53.3 Limitations

- Comments from the file are discarded on save (same as the default KConfig functionality)
- No validation for meta values or locale codes

Chapter 54

Plugin: keytometa

- infos = Information about keytometa plugin is in keys below
- infos/author = Felix Berlakovich `elektra@berlakovich.net`
- infos/licence = BSD
- infos/provides = conv
- infos/needs =
- infos/placements = presetstorage postgetstorage
- infos/status = unittest tested nodep libc configurable discouraged
- infos/metadata =
- infos/description = conversion of keys to metakeys and vice versa

54.1 Introduction

> Note: This plugin uses a deprecated way to store comments.

This plugin converts keys into metakeys of other keys. The keys to be converted are tagged with special metadata. Converting keys into metakeys basically raises two questions:

- which keys should be converted
- which key to append the resulting metakeys to

The keys to be converted are identified by metakeys below `convert` (e.g. `convert/append`). The keys receiving the resulting metadata are identified by append strategies. The plugin currently supports the following metakeys for controlling the conversion:

- `convert/metaname` specifies the name of the resulting metakey. For example tagging the key `user:/config/key1` with `convert/metaname = comment` means that the key will be converted to a metakey with the name `comment`.
- `convert/append` specifies the append strategy (see below)

- `convert/append/samelevel` specifies that the key should only be written to the metadata of a key with the same hierarchy level (see below).

The keys converted to metadata are restored as soon as the keyset is written back. However, the plugin is stateful. This means that a keyset must be read and keys must be converted by the plugin in order to undo this conversion in the set direction. Modifications to the metadata which resulted from converted keys are propagated back to the corresponding key (see merging for more details).

The keys are ordered by the "order" metadata. If two keys are equal according to the order metadata, they are ordered by name instead.

54.2 Installation

See installation. The package is called `libelektra5-extra`.

54.3 Append Strategies

The append strategy specifies which key will receive the resulting metadata. Currently the plugin supports the following strategies:

54.3.1 Parent Strategy

The metadata is added to the first existing parent of the converted key. This does not necessarily have to be the parent of the keyset. If no such key is found, the first key in a sorted keyset will receive the metadata (this is usually the parent key of the keyset). For example consider the following keyset:

```
user:/config/key1
user:/config/key1/child1
user:/config/key2
user:/config/key2/deeper/child2
user:/config/child3
```

If child1, child2 and child3 were tagged with `convert/append = parent`, key1 would receive the metadata from child1 and child3. Key2 would receive the metadata from child2.

54.3.2 Next Strategy

The metadata is added to the key following the converted key in a sorted keyset. If no such key is found (for example because the key to be converted is the last one), the strategy is reverted to parent. For example consider the following keyset:

```
user:/config/deeper/key1
user:/config/key2
user:/config/key3
user:/config/key4
```

If key1 and key3 were tagged with `convert/append = next`, key2 would receive the metadata resulting from key1 and key4 would receive the metadata resulting from key3.

54.3.3 Previous Strategy

The metadata is added to the key preceding the converted key in a sorted keyset. If no such key is found (for example because the key to be converted is the first one), the strategy is reverted to parent. For example consider the following keyset:

```
user:/config/key1
user:/config/deeper/key2
user:/config/key3
user:/config/key4
```

If key2 and key4 were tagged with `convert/append = previous`, key1 would receive the metadata resulting from key2 and key3 would receive the metadata resulting from key4.

54.4 Merging

The metadata resulting from a converted key is never appended to another key which is going to be converted. This prevents that the data of converted keys is invisible after the conversion. Instead the metadata resulting from different converted keys with the same append strategy is merged together (separated by a newline). Keys with different append strategies are skipped, until either a key with the same strategy is found (which is simply merged as described above) or the target key is found. The keys are always processed in the order of an ordered keyset. For example consider the following keyset:

```
user:/config/key0
user:/config/key1 = value1
user:/config/key2 = value2
user:/config/key3 = value3
user:/config/key4 = value4
user:/config/key5
```

If key1 and key2 were tagged with `convert/append = next` and key3 and key4 were tagged with `convert/append = previous` the following would happen:

- the resulting metadata of key0 would contain `value3\nvalue4` (the values of key3 and key4 are merged together and key1 and key2 are skipped, as they have different append strategy)
- the resulting metadata of key5 would contain `value1\nvalue2` (the values of key1 and key2 are merged together and key3 and key4 are skipped, as they have different append strategy)

54.4.1 Same-Level Appending

The option `convert/append/samelevel` can be used to force that the metadata is only appended to a key on the same hierarchy level. If no such key is found, the strategy is reverted to parent. Note, that the value of the samelevel key does not matter. Only its existence is relevant. For example consider the following keyset:

```
user:/config/key0
user:/config/key1/child1
user:/config/key2
user:/config/key3/child2
user:/config/key4
user:/config/key5
user:/config/key6
```

If child1, child2 and key4 were each tagged with `convert/append = next` and child2 and key4 were tagged with `convert/append/samelevel`, key2 would receive the metadata resulting from child1. key0 would receive the metadata resulting from child2 (strategy reverted to parent, as the samelevel request cannot be fulfilled). key5 would receive the metadata resulting from key4.

54.5 Real World Example

The keytometa plugin was initially developed to aid the integration of the Augeas plugin. The Augeas plugin represents comments in configuration files as keys. However, in Elektra comments are usually represented within comment metakeys. Therefore it would be desirable to convert all comment keys to comment metakeys. This is achieved by adding the following to the Augeas plugin contract.

```
// ...
keyNew ("system:/elektra/modules/augeas/config/needs/glob/get/#1",
    KEY_VALUE, "*#comment*",
    KEY_META, "convert/metaname", "comment/#0",
    KEY_META, "convert/append", "next",
    KEY_END),
keyNew ("system:/elektra/modules/augeas/config/needs/glob/get/#1/flags",
    KEY_VALUE, "", /* disable the path matching mode */
    KEY_END)
// ...
;
```

Tagging the keys to be converted to comment metakeys happens via the glob plugin. The metadata set on the key `glob/get/#1` is copied to each key that matches the pattern `*#comment*`, i.e. each comment key generated by the Augeas plugin. `convert/metaname = comment` because we want the comment keys to be converted to the comment metadata. `convert/append = next` is chosen because usually comments occur before the key they describe.

Chapter 55

Plugin: length

- infos = Information about the len plugin is in keys below
- infos/author = Philipp Oppel `philipp.oppel@tuwien.ac.at`
- infos/maintainer = Florian Lindner `florian.lindner@student.tuwien.ac.at`
- infos/licence = BSD
- infos/needs =
- infos/provides = check
- infos/recommends =
- infos/placements = presetstorage postgetstorage
- infos/status = unittest nodep
- infos/metadata = check/length/max
- infos/description = validates if input is less or equal to length and throws error otherwise

55.1 Length Validation

55.1.1 Introduction

This plugins purpose is to check the maximum length of strings. For example if `check/length/max` is set to 3, Strings with more than 3 characters will not validate (e.g. "abcd"), whereas "abc" would validate.

55.1.2 Installation

See installation. The package is called `libelektra5-experimental`.

55.1.3 Usage

```
# Mount `length` plugin to cascading namespace `/tests/length`
kdb mount config.dump /tests/length length
# Check the validity of the string stored in `/tests/length/text`
kdb meta-set spec:/tests/length/text check/length/max 3
# Try to set a longer string
kdb set user:/tests/length/text abcd
# STDERR: .*Validation Semantic.*
# ERROR:  C03200
# RET:    5
# Set a correct string
kdb set user:/tests/length/text abc
kdb get user:/tests/length/text
# Undo modifications to the database
kdb rm -rf /tests/length
kdb umount /tests/length
```

55.1.4 Limitations

The plugin only checks that strings are not longer than a given number. It is not possible to set a minimum length.

Chapter 56

Plugin: line

- infos = Information about line plugin is in keys below
- infos/author = Ian Donnelly ian.s.donnelly@gmail.com
- infos/provides = storage/line
- infos/licence = BSD
- infos/needs = binary
- infos/placements = getstorage setstorage
- infos/status = maintained unittest nodep libc final limited
- infos/description = storage plugin which stores each line from a file

56.1 Introduction

This plugin is useful if you have a file in a format not supported by any other plugin and want to use the Elektra tools to edit individual lines.

This plugin is designed to save each line from a file as a key. The keys form an array. The key names are determined by the line number such as `#3` or `#_12` for lines 4 and 13. The plugin considers `#0` to be the first line. The plugin will automatically add _ to the beginning of key names in order to keep them in numerical order (like specified for Elektra arrays).

The value of each key hold the content of the actual file line-by-line.

56.2 Examples

For example, consider the following content of the file `~/.config/line` where the numbers on the left represent the line numbers:

```
1  setting1 true
2  setting2 false
3  setting3 1000
4  #comment
5
6
7  //some other comment
8
9  setting4 -1
```

We mount that file by:
```
sudo kdb mount line user:/line line
```

This file would result in the following keyset which is being displayed as `key: value`, e.g. with:
```
kdb export -c "format=%s: %s" user:/line simpleini
#> 0: setting1 true
#> 1: setting2 false
#> 2: setting3 1000
#> 3: #comment
#> 4:
#> 5:
#> 6: //some other comment
#> 7:
#> 8: setting4 -1
```

56.2.1 Creating Files

```
# Backup-and-Restore:/tests/line
sudo kdb mount line /tests/line base64 line
kdb set user:/tests/line/add something
kdb set user:/tests/line/ignored huhu
kdb set user:/tests/line ignored   # adding parent key does nothing
kdb set user:/tests/line/add here
cat `kdb file user:/tests/line`
#> something
#> huhu
#> here
kdb ls user:/tests/line
# STDOUT-REGEX: line.+line/#0.+line/#1.+line/#2
kdb set user:/tests/line/#1 huhu
# STDOUT-REGEX: Set string to "huhu"
kdb export user:/tests/line line
#> something
#> huhu
#> here
sudo kdb umount /tests/line
```

56.2.2 Other Tests

```
# Backup-and-Restore:/tests/line
sudo kdb mount line /tests/line base64 line
# create and initialize testfile
echo 'setting1 true'        >  `kdb file user:/tests/line`
echo 'setting2 false'       >> `kdb file user:/tests/line`
echo 'setting3 1000'        >> `kdb file user:/tests/line`
echo '#comment'             >> `kdb file user:/tests/line`
echo                        >> `kdb file user:/tests/line`
echo                        >> `kdb file user:/tests/line`
echo '//some other comment' >> `kdb file user:/tests/line`
echo                        >> `kdb file user:/tests/line`
echo 'setting4 -1'          >> `kdb file user:/tests/line`
# output filecontent and display line numbers
awk '{print NR-1 "-" $0}' < `kdb file user:/tests/line`
#> 0-setting1 true
#> 1-setting2 false
#> 2-setting3 1000
#> 3-#comment
#> 4-
#> 5-
#> 6-//some other comment
#> 7-
#> 8-setting4 -1
# cleanup
kdb rm -r user:/tests/line
sudo kdb umount /tests/line
```

Chapter 57

Plugin: lineendings

- infos = Information about the lineendings plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`, Michael Langhammer `e1125605@student.tuwien.ac.at`, Florian Lindner `florian.lindner@student.tuwien.ac.at`
- infos/maintainer = Florian Lindner `florian.lindner@student.tuwien.ac.at`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/placements = pregetstorage postgetstorage precommit
- infos/status = maintained unittest nodep configurable
- infos/description = verifies line endings of files

57.1 Introduction

The plugin provides validation for the line endings of a file.

57.2 Installation

See installation. The package is called `libelektra5-extra`.

57.3 Usage

The plugin checks the line endings of a file. It validates the line endings against consistency and if the key `/valid` is present also against the specified line ending. Inconsistent line endings or line endings that don't match `/valid` yield an error in case of `kdbSet`. In the case of `kdbGet` the plugin yields a warning instead.

57.4 Configuration

The key `/valid` tells the plugin to reject all line endings other than specified and can have the following options:

- `CRLF`: Carriage return followed by a line feed
- `LFCR`: Line feed followed by a carriage return
- `CR`: Carriage return only
- `LF`: Line feed only

If the key doesn't exist only inconsistent line endings are rejected.

Chapter 58

Plugin: logchange

- infos = Information about the logchange plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/maintainer = Maximilian Irlinger max@maxirlinger.at
- infos/licence = BSD
- infos/needs =
- infos/provides = tracing
- infos/placements = hook pregetstorage postgetstorage postcommit
- infos/status = maintained unittest nodep global nodoc
- infos/description = demonstrates notification of key changes

58.1 Purpose

The purpose of this plugin is to demonstrate how one can be notified of every removed, added or changed key easily.

58.2 Installation

See installation. The package is called `libelektra5-extra`.

58.3 Usage

Prints every added, changed or deleted key on the console. To use it, add it during mounting:

```
sudo kdb mount logchange.dump user:/tests/logchange dump logchange
# And to unmount it
sudo kdb umount user:/tests/logchange
```

Configure the plugin with `log/get=1` to enable printing when configuration is loaded. For example, `kdb gmount logchange log/get=1`.

Chapter 59

Plugin: lua

- infos = Information about the lua plugin is in keys below
- infos/author = Manuel Mausz manuel-elektra@mausz.at
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = unittest configurable global memleak
- infos/description = proxy that calls other plugins (scripts) written in lua

59.1 Introduction

The plugin uses Lua to do magic things. It basically allows to call plugins written in Lua.

What a Lua script can do is not really limited by design, so any kind of plugin may be implemented. The lua plugin is especially useful to write filter and logging scripts.

59.2 Installation

See installation. The package is called `libelektra5-lua`.

59.3 Usage

The lua plugin accepts only the **script** configuration parameter holding the path to a Lua script. The mount command would look like

```
kdb mount file.ini /lua ini lua script=/path/to/filter_script.lua
```

if the **ini** plugin should be used for storage and the lua plugin only serves to invoke the filter script.

For a Lua script that serves as (JSON) storage plugin itself, one could also use

```
kdb mount file.json /lua lua script=/path/to/json_plugin.lua
```

59.3.1 Lua Scripts

Lua scripts can implement the following functions

- elektraOpen(config, errorKey)
- elektraGet(returned, parentKey)
- elektraSet(returned, parentKey)
- elektraError(returned, parentKey)
- elektraClose(errorKey)

where *config* & *returned* are KeySets and *errorKey* & *parentKey* are Keys. For the return codes of the functions, the same rules as for normal plugins apply.

If a function is not available, it simply is not called. A script does not have to implement all functions therefore.

Access to **kdb** can be retrieved using the Lua import
```
require("kdb")
```

59.4 Example

An example script that prints some information for each method call would be:
```
function elektraOpen(config, errorKey)
  print("Lua script method 'elektraOpen' called")
  return 0
end
function elektraGet(returned, parentKey)
  print("Lua script method 'elektraGet' called")
  return 1
end
function elektraSet(returned, parentKey)
  print("Lua script method 'elektraSet' called")
  return 1
end
function elektraError(returned, parentKey)
  print("Lua script method 'elektraError' called")
  return 1
end
function elektraClose(errorKey)
  print("Lua script method 'elektraClose' called")
  return 0
end
```

Further examples can be found in the lua directory.

59.5 Disclaimer

Note, this is a technical preview. It might have severe bugs and the API might change in the future.

Be also aware that a Lua script will never be as performant as a native C/C++ plugin. Spinning up the interpreter takes additional time and resources.

Chapter 60

Plugin: macaddr

- infos = Information about the macaddr plugin is in keys below
- infos/author = Thomas Bretterbauer e01306821@student.tuwien.ac.at
- infos/licence = BSD
- infos/needs =
- infos/provides = check
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest nodep
- infos/metadata = check/macaddr
- infos/description = Validates MAC-addresses and returns them as integers

60.1 Introduction

This plugin validates MAC-addresses. The following MAC-address-formats are supported:

```
XX-XX-XX-XX-XX-XX
XX:XX:XX:XX:XX:XX
XXXXXX-XXXXXX
Integer values (0 - 281474976710655)
```

`kdbGet` returns an integer-representation of these values.

60.2 Installation

See installation. The package is called `libelektra5-extra`.

60.3 Usage

```
# Backup-and-Restore: user:/tests/mac
# Mount `macaddr` plugin
sudo kdb mount macconf.ecf user:/tests/mac macaddr
# Setting a MAC address using colons
kdb set user:/tests/mac/mac1 00:A0:C9:14:C8:29
# RET: 0
# Setting a MAC address using hyphens
kdb set user:/tests/mac/mac2 00-A0-C9-14-C8-29
# RET: 0
# Setting a MAC address using one hyphen
kdb set user:/tests/mac/mac3 00A0C9-14C829
# RET: 0
# Setting a MAC address using an integer value
kdb set user:/tests/mac/mac4 17661175009296
# RET: 0
# Marking written keys as MAC addresses
kdb meta-set user:/tests/mac/mac1 check/macaddr ""
kdb meta-set user:/tests/mac/mac2 check/macaddr ""
kdb meta-set user:/tests/mac/mac3 check/macaddr ""
kdb meta-set user:/tests/mac/mac4 check/macaddr ""
# Setting a MAC address using an invalid address
kdb set user:/tests/mac/mac1 00:G1:C9:14:C8:29
# RET: 5
# Setting a MAC address using an invalid address
kdb set user:/tests/mac/mac1 00:E1:C914:C8:29
# RET: 5
# Setting a MAC address using an invalid address
kdb set user:/tests/mac/mac4 281474976710656
# RET: 5
# Retrieving a MAC address with colons as integer
kdb get user:/tests/mac/mac1
#> 690568349737
# Retrieving a MAC address with hyphens as integer
kdb get user:/tests/mac/mac2
#> 690568349737
# Retrieving a MAC address with one hyphen as integer
kdb get user:/tests/mac/mac3
#> 690568349737
# Retrieving an integer MAC address
kdb get user:/tests/mac/mac4
#> 17661175009296
kdb rm -r user:/tests/mac
sudo kdb umount user:/tests/mac
```

60.4 Dependencies

None.

60.5 Limitations

None.

Chapter 61

Plugin: mathcheck

- infos = Information about the mathcheck plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/provides = check
- infos/needs =
- infos/placements = presetstorage
- infos/status = maintained unittest shelltest nodep discouraged
- infos/metadata = check/math
- infos/description = validates a set of keys through a mathematical expression

61.1 Introduction

Compares a key value to a mathematical expression using polish prefix notation defined in the `check/math` metakey. Operations are `+ - / *`. Operants are keys with names relative to the parent key. How the values are compared is specified at the beginning of the metakey using the conditions `<, <=, ==, !=, =>, >, :=` `:=` is used to set key values. All values are interpreted as `double` floating point values.

61.2 Installation

See installation. The package is called `libelektra5-extra`.

61.2.1 Keynames

Keynames are all either relative to to-be-tested key (starting with `./` or `../`), relative to the parentkey (starting with `@/`) or absolute (e.g. `system:/key`).

61.3 Examples

`check/math = "== + ../testval1 + ../testval2 ../testval3"` compares the keyvalue to the sum of testval1-3 and yields an error if the values are not equal. `check/math = "<= - @/testval1 * @/testval2 @/testval3"` tests if the keyvalue is less than or equal to `testval1 - (testval2 * testval3)` and yields an error if not.

Full example:

```
# Backup-and-Restore:user:/tests/mathcheck
sudo kdb mount mathcheck.dump user:/tests/mathcheck mathcheck
kdb set user:/tests/mathcheck/a 3.1
kdb set user:/tests/mathcheck/b 4.5
kdb set user:/tests/mathcheck/k 7.6
kdb meta-set user:/tests/mathcheck/k check/math "== + ../a ../b"
# should fail
kdb set user:/tests/mathcheck/k 7.7
# RET:5
# ERROR:C03200
# Set string to "7.7"
# Sorry, module mathcheck issued the error C03200:
# invalid value: 7.7 != 7.6
```

To calculate values on-demand you can use:

```
kdb meta-set user:/tests/mathcheck/k check/math ":= + @/a @/b"
kdb set user:/tests/mathcheck/a 8.0
kdb set user:/tests/mathcheck/b 4.5
kdb get user:/tests/mathcheck/k
#> 12.5
kdb set user:/tests/mathcheck/a 5.5
kdb get user:/tests/mathcheck/k
#> 10
```

It also works with constants:

```
kdb meta-set user:/tests/mathcheck/k check/math ":= + ../a '5'"
kdb set user:/tests/mathcheck/a 5.5
kdb get user:/tests/mathcheck/k
#> 10.5
kdb set user:/tests/mathcheck/a 8.0
kdb get user:/tests/mathcheck/k
#> 13
#cleanup
kdb rm -r user:/tests/mathcheck
sudo kdb umount user:/tests/mathcheck
```

Chapter 62

Plugin: mini

- infos = Information about the mini plugin is in keys below
- infos/author = René Schwaiger `sanssecours@me.com`
- infos/licence = BSD
- infos/needs = ccode
- infos/provides = storage/properties
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = shelltest unittest nodep limited
- infos/metadata =
- infos/description = A minimal plugin for properties files

62.1 mINI

The “maybe this is not INI” plugin (`mini`) is a very simple storage plugin loosely based on the `INI` file format. Since this plugin **does not support sections** it might be more appropriate to say that it is based on the `.properties` format, used in many Java applications.

62.1.1 Examples

62.1.1.1 Basic Usage

The following example shows basic usage of the `mini` plugin.

```
# Mount mini plugin to `user:/tests/mini`
sudo kdb mount mini.ini user:/tests/mini mini
# Add two key value pairs to the database
kdb set user:/tests/mini/key value
#> Create a new key user:/tests/mini/key with string "value"
kdb set user:/tests/mini/mi/mi/mr beaker
#> Create a new key user:/tests/mini/mi/mi/mr with string "beaker"
# Export our current configuration
kdb export user:/tests/mini mini
#> key=value
#> mi/mi/mr=beaker
# Manually add some values
```

```
echo "key = unicorn"           » `kdb file user:/tests/mini`
echo "level1/level2 = alien" » `kdb file user:/tests/mini`
# Now `user:/tests/mini` contains four key value pairs
kdb ls user:/tests/mini
#> user:/tests/mini/key
#> user:/tests/mini/level1/level2
#> user:/tests/mini/mi/mi/mr
#> user:/tests/mini/key
# Let us check if `user:/tests/mini/key` contains the correct value
kdb get "user:/tests/mini/key"
#> unicorn
# Undo modifications to the key database
kdb rm -r user:/tests/mini
sudo kdb umount user:/tests/mini
```

62.1.1.2 Escaping

As with most configuration file formats, some characters carry special meaning. In the case of the `mini` plugin that character are

1. the = sign, which separates keys from values and
2. # and **;** , the characters that denote a comment.

In case of **key values** you do not need to care about the special meaning of these characters most of the time, since the plugin handles escaping and unescaping of them for you. Since mINI use the backslash character (`\`) to escape values, the backspace character will be escaped too (`\\`). The following example shows the escaping behavior.

```
sudo kdb mount mini.ini user:/tests/mini mini
# Store a value containing special characters
kdb set user:/tests/mini/key ';#=\'
#> Create a new key user:/tests/mini/key with string ";#=\"
# The actual file contains escaped version of the characters
kdb file user:/tests/mini | xargs cat
#> key=\;\#\=\\
# However, if you retrieve the value you do not have to care
# about the escaped characters
kdb get user:/tests/mini/key
#> ;#=\
# If we do not escape the `;` and `#` characters, then they
# donate a comment.
echo 'background = \#0F0F0F ; Background color' » `kdb file user:/tests/mini`
echo 'foreground = \#FFFFFF # Foreground color' » `kdb file user:/tests/mini`
kdb get user:/tests/mini/background
#> #0F0F0F
kdb get user:/tests/mini/foreground
#> #FFFFFF
# Undo modifications to the key database
kdb rm -r user:/tests/mini
sudo kdb umount user:/tests/mini
```

In the case of **key names** you **must not use any of the characters mentioned above** (**;** , # and =) at all. Otherwise the behavior of the plugin will be **undefined**.

62.1.2 Limitations

This plugin only supports simple key-value based properties files without sections. mINI also does not support metadata. If you want a more feature complete plugin, then please take a look at the toml plugin. The example below shows some of the limitations of the plugin.

```
sudo kdb mount mini.ini user:/tests/mini mini
# The plugin does not support sections or multi-line values
echo   '[section]'         » `kdb file user:/tests/mini`
printf 'key="multi\nline"' » `kdb file user:/tests/mini`
# mINI only reads the first line of the value with the name `key`, since
# the plugin assigns no special meaning to double or single quotes.
kdb ls user:/tests/mini 2> stderr.txt
#> user:/tests/mini/key
# As we can see in the first two line of the standard error output below,
# mINI will inform us about lines it was unable to parse.
cat stderr.txt | grep -oE 'Line [[:digit:]]+.*' | sed 's/^[[:space:]]*//'
```

```
#> Line 1: '[section]' is not a valid key value pair
#> Line 3: 'line"' is not a valid key value pair
# Unlike the `ini` and `ni` plugin, mINI does not support metadata.
kdb meta-set user:/tests/mini foo bar
# RET: 5
kdb meta-ls user:/tests/mini
# RET: 1
# The value of `key` also contains the double quote symbol, since mINI does
# not assign special meaning to quote characters.
kdb get user:/tests/mini/key
#> "multi
# Undo modifications
rm stderr.txt
kdb rm -r user:/tests/mini
sudo kdb umount user:/tests/mini
```

Chapter 63

Plugin: missing

- infos = Information about the template plugin is in keys below
- infos/author = Vid Leskovar `vid.leskovar5@gmail.com`
- infos/maintainer = Florian Lindner `florian.lindner@student.tuwien.ac.at`
- infos/licence = BSD
- infos/needs =
- infos/provides = missing
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = nodep nodoc
- infos/metadata =
- infos/description = A plugin which informs the user about a missing backend.

63.1 Usage

Internal plugin, do not use directly.

Chapter 64

Plugin: mmapstorage

- infos = Information about the mmapstorage plugin is in keys below
- infos/author = Mihael Pranjić `mpranj@limun.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/mmapstorage
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = unittest experimental
- infos/metadata =
- infos/description = high performance storage using memory mapped files

64.1 Introduction

This is a high performance storage plugin that supports full Elektra semantics.

64.2 Format

The storage format uses Elektra's in-memory data layout and employs the `mmap()` system call to read/write data. The format is not portable across different architectures/platforms. The format can be seen as a memory dump of a keyset. Therefore, the files must not be edited by hand. Files written by mmapstorage are not intended to be human-readable.

64.3 Usage

Mount mmapstorage using `kdb mount`:

```
sudo kdb mount config.mmap user:/tests/mmapstorage mmapstorage
```

Unmount mmapstorage using `kdb umount`:

```
sudo kdb umount user:/tests/mmapstorage
```

64.4 Compiling

The mmapstorage has two compilation variants:

1. mmapstorage
2. mmapstorage_crc

The `mmapstorage` will always be compiled on a supported system (see Dependencies). When zlib is available, we will additionally compile the `mmapstorage_crc` variant. The first variant does not do a CRC32 checksum of the critical data, while the second variant always checks the CRC32 checksum for additional security.

64.5 Installation

See installation. The `mmapstorage` variant is part of the `libelektra5` package and the `mmapstorage_crc` is part of the `libelektra5-extra` package.

64.6 Dependencies

POSIX compliant system (including XSI extensions).

Additionally, zlib is needed for the `mmapstorage_crc` compilation variant: `zlib1g-dev` or `zlib-devel`.

64.7 Examples

```
# Mount mmapstorage to `user:/tests/mmapstorage`
sudo kdb mount config.mmap user:/tests/mmapstorage mmapstorage
# Add some values via `kdb set`
kdb set user:/tests/mmapstorage 'Some root key'
kdb set user:/tests/mmapstorage/dir 'Directory within the hierarchy.'
kdb set user:/tests/mmapstorage/dir/leaf 'A leaf node holding some valuable data.'
kdb meta-set  user:/tests/mmapstorage/dir/leaf superMetaKey 'Metadata is supported too.'
# List the configuration tree below `user:/tests/mmapstorage`
kdb ls user:/tests/mmapstorage
#> user:/tests/mmapstorage
#> user:/tests/mmapstorage/dir
#> user:/tests/mmapstorage/dir/leaf
# Retrieve the new values
kdb get user:/tests/mmapstorage
#> Some root key
kdb get user:/tests/mmapstorage/dir
#> Directory within the hierarchy.
kdb get user:/tests/mmapstorage/dir/leaf
#> A leaf node holding some valuable data.
kdb meta-get  user:/tests/mmapstorage/dir/leaf superMetaKey
#> Metadata is supported too.
# Undo modifications to the database
kdb rm -r user:/tests/mmapstorage
# Unmount mmapstorage
sudo kdb umount user:/tests/mmapstorage
```

64.8 Limitations

Mapped files shall not be altered, otherwise the behavior is undefined.

The `mmap()` system call only supports regular files and so does the mmapstorage plugin with one notable exception: The plugin detects when it is called with the files `/dev/stdin` and `/dev/stdout` and makes an internal copy. This makes the plugin compatible with `kdb import` and `kdb export`.

Chapter 65

Plugin: mmapstorage

65.1 Additional shell tests for mmapstorage

This file contains important shell tests for mmapstorage which do not fit well into the plugin README.

65.2 Test kdb export & import (stat: pipe size known)

```
# Make temp file
kdb set system:/tests/mmaptempfile $(mktemp)
# Mount mmapstorage to 'user:/tests/mmapstorage'
sudo kdb mount config.mmap user:/tests/mmapstorage mmapstorage
# Add some values via 'kdb set'
kdb set user:/tests/mmapstorage/test1 test1
kdb set user:/tests/mmapstorage/test2 test2
kdb set user:/tests/mmapstorage/test1/test3 test3
# List the configuration tree below 'user:/tests/mmapstorage'
kdb ls user:/tests/mmapstorage
#> user:/tests/mmapstorage/test1
#> user:/tests/mmapstorage/test1/test3
#> user:/tests/mmapstorage/test2
# Retrieve the new values
kdb get user:/tests/mmapstorage/test1
#> test1
kdb get user:/tests/mmapstorage/test2
#> test2
kdb get user:/tests/mmapstorage/test1/test3
#> test3
kdb export user:/tests/mmapstorage mmapstorage > $(kdb get system:/tests/mmaptempfile)
kdb rm -r user:/tests/mmapstorage
kdb ls user:/tests/mmapstorage
#>
kdb import user:/tests/mmapstorage mmapstorage < $(kdb get system:/tests/mmaptempfile)
# List the configuration tree below 'user:/tests/mmapstorage'
kdb ls user:/tests/mmapstorage
#> user:/tests/mmapstorage/test1
#> user:/tests/mmapstorage/test1/test3
#> user:/tests/mmapstorage/test2
# Retrieve the new values
kdb get user:/tests/mmapstorage/test1
#> test1
kdb get user:/tests/mmapstorage/test2
#> test2
kdb get user:/tests/mmapstorage/test1/test3
#> test3
# Undo modifications to the database
kdb rm -r user:/tests/mmapstorage
# Unmount mmapstorage
sudo kdb umount user:/tests/mmapstorage
# Remove temp file
rm $(kdb get system:/tests/mmaptempfile)
kdb rm -r system:/tests/mmaptempfile
```

65.3 Test kdb export & import (stat: pipe size unknown)

```
# Make temp file
kdb set system:/tests/mmaptempfile $(mktemp)
# Mount mmapstorage to 'user:/tests/mmapstorage'
sudo kdb mount config.mmap user:/tests/mmapstorage mmapstorage
# Add some values via 'kdb set'
kdb set user:/tests/mmapstorage/test1 test1
kdb set user:/tests/mmapstorage/test2 test2
kdb set user:/tests/mmapstorage/test1/test3 test3
# List the configuration tree below 'user:/tests/mmapstorage'
kdb ls user:/tests/mmapstorage
#> user:/tests/mmapstorage/test1
#> user:/tests/mmapstorage/test1/test3
#> user:/tests/mmapstorage/test2
# Retrieve the new values
kdb get user:/tests/mmapstorage/test1
#> test1
kdb get user:/tests/mmapstorage/test2
#> test2
kdb get user:/tests/mmapstorage/test1/test3
#> test3
kdb export user:/tests/mmapstorage mmapstorage > $(kdb get system:/tests/mmaptempfile)
kdb rm -r user:/tests/mmapstorage
kdb ls user:/tests/mmapstorage
#>
cat $(kdb get system:/tests/mmaptempfile) | kdb import user:/tests/mmapstorage mmapstorage
# List the configuration tree below 'user:/tests/mmapstorage'
kdb ls user:/tests/mmapstorage
#> user:/tests/mmapstorage/test1
#> user:/tests/mmapstorage/test1/test3
#> user:/tests/mmapstorage/test2
# Retrieve the new values
kdb get user:/tests/mmapstorage/test1
#> test1
kdb get user:/tests/mmapstorage/test2
#> test2
kdb get user:/tests/mmapstorage/test1/test3
#> test3
# Undo modifications to the database
kdb rm -r user:/tests/mmapstorage
# Unmount mmapstorage
sudo kdb umount user:/tests/mmapstorage
# Remove temp file
rm $(kdb get system:/tests/mmaptempfile)
kdb rm -r system:/tests/mmaptempfile
```

Chapter 66

Plugin: modules

- infos = Information about the modules plugin is in keys below
- infos/author = Klemens Böswirth `k.boeswirth+git@gmail.com`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = backend
- infos/status = unittest nodep nodoc
- infos/metadata =
- infos/description = For internal use only

66.1 Introduction

For internal use by `libelektra-kdb` only. Intentionally not documented here.

Chapter 67

Plugin: mozprefs

67.1 Introduction

This file explains how Firefox preferences can be manipulated at run-time using Elektra's intercept open and a custom autoconfig script.

67.2 Basics

`kdb configure-firefox -s` will configure everything needed to get started.

67.2.1 Files

- `/usr/lib/firefox/ /usr/lib/firefox-esr`
  - `defaults/pref/autoconfig.js` Loads `elektra.cfg`
    ``` pref("general.config.filename", "elektra.cfg"); pref("general.config.obscure_value", 0); ```
  - `elektra.cfg` Contains the autoconfig code
- `~/.mozilla/firefox/<profile>/prefs.js` Contains the user preferences. The configure-firefox script will append the config for our autoconfig script:

```
user_pref("elektra.config.file", "/tmp/imnotreal.js");
user_pref("elektra.config.reload_trigger_port", 65432);
```

Every time the string `reload` is send to localhost:`elektra.config.reload_trigger_port` the autoconfig script will import the preferences from `elektra.config.file`
`elektry.config.file` contains the name of the dummy file for `intercept open`. `open` calls to this file will be intercepted and the content generated by `intercept open`
```

67.2.2 Setting Preferences

67.2.2.1 Guided Setup

Running `kdb configure-firefox -a` provides a guided setup for adding http_proxy and homepage preferences.

```
% kdb configure-firefox -a
Add new preferences
Config Setup:
1) Proxy
2) Homepage
0) Exit
1
Setting up HTTP Proxy
1) lock
2) default
3) user
0) Exit
1
Proxy Type
0) No Proxy
1) Manual Setup
2) PAC
4) Auto-detect
5) System Settings
1
Set string to 1
Host/IP: 127.0.0.1
Port: 8080
Setting lockPref HTTP Proxy to 127.0.0.1:8080
```

67.2.2.2 Manual Setup

This example shows how to manually setup a preferences. It's equivalent to the example shown above in the `Guided setup`

```
kdb meta-set user:/prefs/lock/network/proxy/type type integer
kdb set user:/prefs/lock/network/proxy/type 1
kdb meta-set user:/prefs/lock/network/proxy/http type string
kdb set user:/prefs/lock/network/proxy/http 127.0.0.1
kdb meta-set user:/prefs/lock/network/proxy/http_port type integer
kdb set user:/prefs/lock/network/proxy/http_port 8080
```

67.2.3 Test Setup

Running `kdb configure-firefox -t` will set up some test values.

```
kdb export /preload
#> [open]
#> \/tmp\/imnotreal.js =
#> \/tmp\/imnotreal.js/generate = user:/prefs
#> \/tmp\/imnotreal.js/generate/plugin = mozprefs
kdb export user:/prefs
#> [lock/a/lock]
#> 1 = lock1
#> 2 = lock2
#> [pref/a/default]
#> 1 = 1
#> 2 = 2
#> [user:/a/user]
#> f = false
#> t = true
kdb export user:/prefs mozprefs
#> lockPref("a.lock.1", "lock1");
#> lockPref("a.lock.2", "lock2");
#> pref("a.default.1", 1);
#> pref("a.default.2", 2);
#> user_pref("a.user.f", false);
#> user_pref("a.user.t", true);
kdb elektrify-open firefox-esr "about:config"
kdb meta-set user:/prefs/lock/a/lock/3 type boolean
kdb set user:/prefs/lock/a/lock/3 true
kdb export user:/prefs
#> [lock/a/lock]
#> 1 = lock1
```

```
#> 2 = lock2
#> 3 = true
#> [pref/a/default]
#> 1 = 1
#> 2 = 2
#> [user:/a/user]
#> f = false
#> t = true
```

67.3 Limitations

Changing locked values doesn't work without restarting.

Chapter 68

Plugin: mozprefs

- infos = Information about the mozprefs plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained reviewed conformant compatible coverage specific unittest tested nodep libc preview experimental difficult limited unfinished concept
- infos/metadata =
- infos/description = storage plugin for mozilla preferences

68.1 Basics

This plugin works on Mozilla preference files and is used in Elektra's Firefox autoconfig script.

68.2 Installation

See installation. The package is called `libelektra5-extra`.

68.2.1 Preference Types

- Default preferences: `pref(...)`, keys below `mountpoint/pref/`.
- User preferences: `user_pref(...)`, keys below `mountpoint/user/`.
- Lock preferences: `lockPref(...)`, keys below `mountpoint/lock/`.
- Sticky preferences: `sticky_pref(...)`, keys below `mountpoint/sticky/`.

Only Keys below one of these points are valid, everything else will be dropped

68.2.2 Data Types

- `integer`
- `string`
- `boolean`

68.2.3 Hierarchy

In Mozilla preference files `.` is used to separate sections, while elektra uses `/`. For simplification, and because `/` isn't allowed in preference keys, the plugin treats `.` and `/` equally.

```
kdb set system:/prefs/lock/a/lock/key lock
kdb set system:/prefs/lock/a/lock.key lock
kdb set system:/prefs/lock/a.lock.key lock
```

will all result in `lockPref("a.lock.key", "lock");`

68.3 Example

```
# Backup-and-Restore:user:/tests/mozprefs
sudo kdb mount prefs.js user:/tests/mozprefs mozprefs
kdb meta-set user:/tests/mozprefs/lock/a/lock/key type boolean
kdb set user:/tests/mozprefs/lock/a/lock/key true
kdb meta-set user:/tests/mozprefs/pref/a/default/key type string
kdb set user:/tests/mozprefs/pref/a/default/key "i'm a default key"
kdb meta-set user:/tests/mozprefs/user/a/user/key type integer
kdb set user:/tests/mozprefs/user/a/user/key 123
cat `kdb file user:/tests/mozprefs`
#> lockPref("a.lock.key", true);
#> pref("a.default.key", "i'm a default key");
#> user_pref("a.user.key", 123);
# cleanup
kdb rm -r user:/tests/mozprefs
sudo kdb umount user:/tests/mozprefs
```

Chapter 69

Plugin: network

- infos = Information about network plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = check
- infos/needs =
- infos/placements = presetstorage
- infos/status = maintained unittest nodep libc
- infos/metadata = check/ipaddr check/port check/port/listen
- infos/description = Checks keys if they contain a valid ip address

69.1 Introduction

This plugin is a check plugin that checks if a key contains a valid ip address. It uses the `POSIX.1-2001` interface `getaddrinfo()` in order to check if an ip address is valid.

Furthermore `getaddrinfo()` is used in `check/port` to resolve a port by its service name which is defined under `/etc/services`. The portname is translated to the respective portnumber. The plugin can be used to check for valid port numbers and if the set port is free to use.

69.2 Purpose

While, in theory, a regular expression can express if a string is a network address, in practice, such an attempt does not work well. The reason is that an unmanageable number of valid shortenings for IPv6 addresses makes the regular expression hard to write and understand.

So the idea of building such a complicated regular expression was discarded, but instead a dedicated checker was introduced. The idea is to use the operating system facilities to resolve the network address. If this succeeds, it is guaranteed that this network address will be valid when it is resolved by the same interface afterwards.

Many network address translators coexist. In `POSIX.1-2001` a powerful address translator is provided with the interface `getaddrinfo()`. It is a common network address translation for both IPv4 and IPv6. We used it to implement this plugin.

69.3 Usage

Every key tagged with the metakey `check/ipaddr` will be checked using `getaddrinfo()`. If additionally the values `ipv4` or `ipv6` are supplied, the address family will be specified.

69.3.1 Example

```
# Mount Network plugin to 'user:/tests/network'
sudo kdb mount config.file user:/tests/network network
# Set valid IPv4 address
kdb set user:/tests/network/host 127.0.0.1
# Check for valid IPv4 address
kdb meta-set user:/tests/network/host check/ipaddr ipv4
# Try to set invalid IPv4 address
kdb set user:/tests/network/host 133.133.133.1337
# RET: 5
# STDERR:.*Validation Semantic: name:.*133.133.133.1337.*
kdb get user:/tests/network/host
#> 127.0.0.1
# Set valid IPv4 address
kdb set user:/tests/network/host 1.2.3.4
#> Set string to "1.2.3.4"
kdb get user:/tests/network/host
#> 1.2.3.4
# Check for any valid network address
kdb meta-set user:/tests/network/host check/ipaddr ”
# If identifier 'localhost' is not a valid network address it is not part of /etc/hosts
kdb set user:/tests/network/host localhost || ! grep -q localhost /etc/hosts
kdb get user:/tests/network/host
# STDOUT-REGEX: localhost|1.2.3.4
# Undo modifications to the key database
kdb rm -r user:/tests/network
sudo kdb umount user:/tests/network
```

If `check/port` is specified on a given key, the plugin will validate if the port is a correct number between 1 and 65535.

If `check/port/listen` is specified, the plugin will check if the application can be started and listen on the given port.

If `gethostbyname()` returns a "try again" error, two retries will be attempted. If all retries were consumed, the plugin will proceed in the same manner as with other errors.

69.4 Future Work

`check/port/connect` to check if the port can be pinged/reached (usually for clients). If not reachable, users receive a warning. A correct timeout setting will be problematic though.

Chapter 70

Plugin: ni

- infos = Information about ni plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = storage/ini
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest libc nodep
- infos/metadata =
- infos/description = Reads and writes the nickel ini format

70.1 Introduction

This plugin uses the nickel library in order to read/write metakeys in the nickel ini format. Its purpose is to be used in the `spec`-namespace or when any metadata should be stored.

For configuration itself you should prefer the toml plugin.

70.2 Usage

To mount the ni plugin you can simply use:
```
kdb mount file.ini spec:/ni ni
```

The strength of this plugin is that it supports arbitrary meta data and the file format is still human-readable. For example the following lines:
```
[key]
meta=foo
```

specify that `key` has a metadata key `meta` containing the metavalue `foo`:
```
kdb meta-get user:/ni/key meta
#> foo
```

For the metadata of the parent key use the following syntax:
```
[]
meta=foo
```

Line continuation works by ending the line with `\\` (a single backslash). If you want a line break at the end of the line, use `\\n\\`.

To export a `KeySet` in the nickel format use:
```
kdb export spec:/ni ni > example.ni
```

For in-detail explanation of the syntax (nested keys are not supported by the plugin) see /src/plugins/ni/nickel-1.1.0/include/bohr/ni.h

70.3 Examples

```
# Mount the `ni` plugin at `spec:/tests/ni`
sudo kdb mount file.ini spec:/tests/ni ni
# Add some metadata
kdb meta-set spec:/tests/ni/key metakey metavalue
kdb meta-set spec:/tests/ni/key check/type char
# Retrieve metadata
kdb meta-ls spec:/tests/ni/key
#> check/type
#> metakey
kdb meta-get spec:/tests/ni/key metakey
#> metavalue
# Add and retrieve key values
kdb get spec:/tests/ni/key
#>
kdb set spec:/tests/ni/key value
kdb set spec:/tests/ni/key/to nothing
kdb get spec:/tests/ni/key
#> value
kdb get spec:/tests/ni/key/to
#> nothing
# Undo modifications
kdb rm -r spec:/tests/ni
sudo kdb umount spec:/tests/ni
```

70.4 Limitations

- Supports most KeySets, but `kdb test` currently reports some errors (likely because of the UTF-8 handling happening within ni).
- Keys have a random order when written out.
- Comments are not preserved, they are simply removed.
- Parse errors simply result in ignoring (and removing) these parts.

70.5 Nickel

This plugin is based on the Nickel Library written by author: charles@chaoslizard.org

Nickel (Ni) has its strength in building up a hierarchical recursive Node structure which is perfect for parsing and generating ini files. With them arbitrary deep nested hierarchy are possible, but limited in a keyname of a fixed size.

The API of nickel is very suited for elektra, it can use `FILE*` pointers (using that elektra could open and lock files), the node-hierarchy can be transformed to keysets, but it lacks many features like comments and types.

The format is more general than the kde-ini format, it can handle their configuration well, when the section names do not exceed the specified length. Nesting is only required in the first depth, any deeper is not understood by kde config parser.

The memory footprint is for a 190.000 (reduced to 35.000 when rewrote first) line ini file with 1.1MB size is 16.88 MB. The sort order is not stable, even not with the same file rewritten again.

Chapter 71

Plugin: noresolver

- infos = Information about the noresolver plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = resolver
- infos/needs =
- infos/placements = rollback getresolver setresolver commit
- infos/status = maintained nodep libc configurable discouraged
- infos/description = resolver dummy that always succeeds

71.1 Introduction

`noresolver` is designed for non-file-based storage plugins like uname. It is a trivial resolver that:

- does not resolve file names but forwards them as given to the storage plugin
- does not provide any consistency guarantees but storage plugins directly write to the config files

71.2 Explanation

Returns success on every call and can be used as resolver.

It also exports a function checks if a filename is valid. It returns 1 for a relative path and 0 for an absolute path (always successfully).

The path passed to the storage plugin via the parent key is exactly the value that was set during mounting. This plugin *does not* resolve any paths. If a relative path was set while mounting, storage plugins may treat it as relative to the current working directory.

Chapter 72

Plugin: passwd

- infos = Information about the passwd plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/passwd
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained reviewed conformant compatible coverage specific unittest tested nodep libc configurable experimental limited
- infos/metadata =
- infos/description = storage plugin for passwd files

72.1 Introduction

This plugin parses `passwd` files, e.g. `/etc/passwd`.

72.2 Installation

See installation. The package is called `libelektra5-experimental`.

72.3 Implementation Details

The non-POSIX function `fgetpwent` (GNU_SOURCE) will be used to read the file supplied by the resolver. As a fallback we implemented our own version based on musls `fgetpwent`.

For writing `putpwent` (GNU_SOURCE) will be used. If it is not available the plugin will write straight to the config file.

72.4 Requirements

For the plugin to be build at least `POSIX_C_SOURCE >= 200809L` compatibility is required.

72.5 Configuration

If the config key `index` is set to `name` passwd entries will be sorted by name, if not set or set to `uid` passwd entries will be sorted by uid

72.6 Fields

- `gecos` contains the full name of the account
- `gid` contains the accounts primary group id
- `home` contains the path to the accounts home directory
- `shell` contains the accounts default shell
- `uid` contains the accounts uid
- `name` contains the account name

72.7 Usage

To mount the passwd file you can run
```
sudo kdb mount /etc/passwd system:/tests/passwd passwd index=name
```

To see which entries for the root user exist you can run
```
kdb ls system:/tests/passwd/root
#> system:/tests/passwd/root
#> system:/tests/passwd/root/gecos
#> system:/tests/passwd/root/gid
#> system:/tests/passwd/root/home
#> system:/tests/passwd/root/passwd
#> system:/tests/passwd/root/shell
#> system:/tests/passwd/root/uid
```

If you want to receive one specific value you can run for example
```
kdb get system:/tests/passwd/root/gecos
```

You can also export it as whole in any format you like, for example JSON
```
kdb export system:/tests/passwd/root json
# {
#     "gecos": "root",
#     "gid": "0",
#     "home": "/root",
#     "passwd": "x",
#     "shell": "/bin/bash",
#     "uid": "0"
# }
```

To unmount it, you can run
```
sudo kdb umount system:/tests/passwd
```

Chapter 73

Plugin: path

- infos = Information about path plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/placements = presetstorage
- infos/needs =
- infos/provides = check
- infos/status = maintained nodep libc
- infos/metadata = check/path check/path/mode check/path/user
- infos/description = Checks if keys enriched with appropriate metadata contain valid paths as values as well as correct permissions

73.1 Introduction

This plugin checks whether the value of a key is a valid file system path and optionally if correct permissions are set for a certain user.

73.2 Installation

See installation. The package is called `libelektra5-extra`.

73.3 Purpose

The motivation to write this plugin is given by the two paths that exist in /etc/fstab: the device file and the mountpoint. A missing file is not necessarily an error, because the device file may appear later when a device is plugged in and the mountpoint may be there when another subsequent mount was executed. So only warnings are yielded in that case. One situation, however, presents an error: Only an absolute path is allowed to occur for both device and mountpoint. When checking for relative files, it is not enough to look at the first character if it is a `/`, because remote file systems and some special names are valid, too.

If `check/path/mode = <permission>` is also present it will check for the correct permissions of the file/directory. Optionally, you can also add `check/path/user = <user>"` which then checks the permissions for the given user. When calling `kdb set` on the actual key, you have to run as `root` user or the file permissions cannot be checked (you will receive an error message). It is also possible to leave the `check/path/user` empty (just provide an empty string) which then takes the executing user as target to check. So for example `sudo kdb set ...` will check if `root` can access the target file/directory whereas `kdb set ...` will take the current executing process/user. If `check/path/user` is not given at all, the plugin will check accessibility for the `root` user only (which again requires `sudo`)

`check/path/mode = rw` and `check/path/user = tomcat` for example will check if the user `tomcat` has read and write access to the path which was set for the key. Please note that the file has to exist already and it is not checked if the user has the right to create a file in the directory.

Permissions available:

- `r`: **R**ead
- `w`: **W**rite
- `x`: e**X**ecute

73.4 Usage

If the metakey `check/path` is present, it is checked if the value is a valid absolute file system path. If a metavalue is present, an additional check will be done if it is a directory or device file.

73.5 Examples

An example on which the user should have no permission at all for the root directory.

```
sudo kdb mount test.dump user:/tests path dump
sudo kdb set user:/tests/path "$HOME"
sudo kdb meta-set user:/tests/path check/path ""
sudo kdb meta-set user:/tests/path check/path/user ""
sudo kdb meta-set user:/tests/path check/path/mode "rw"
# Standard users should not be able to read/write the root folder
[ $(id -u) = 0 ] && printf >&2 'User is root\n' || kdb set user:/tests/path "/root"
# STDERR: User is root|.*C03200.*
# Set something which the current user can access for sure
kdb set user:/tests/path "$HOME"
# STDOUT-REGEX: .*Set string to "/.*".*
#cleanup
sudo kdb rm -r user:/tests
sudo kdb umount user:/tests
```

An example where part of the permissions are missing for a tmp file

```
sudo kdb mount test.dump user:/tests path dump
sudo kdb set user:/tests/path "$HOME"
sudo kdb meta-set user:/tests/path check/path ""
sudo kdb meta-set user:/tests/path check/path/user ""
sudo kdb meta-set user:/tests/path check/path/mode "rwx"
```

```
# Standard users should not be able to read/write the root folder
kdb set user:/tests/path/tempfile $(mktemp)
chmod +rw `kdb get user:/tests/path/tempfile`
kdb set user:/tests/path `kdb get user:/tests/path/tempfile`
# ERROR:C03200
# Set something which the current user can access for sure
chmod +x `kdb get user:/tests/path/tempfile`
kdb set user:/tests/path `kdb get user:/tests/path/tempfile`
# STDOUT-REGEX: Set string to "/.*".*
#cleanup
sudo rm -rf `kdb get user:/tests/path/tempfile`
sudo kdb rm -r user:/tests
sudo kdb umount user:/tests
```

73.6 Future work

Add a check which ensures that the given path is a file/directory/symbolic link/hard link/etc.

Chapter 74

Plugin: process

- infos = Information about the process plugin is in keys below
- infos/author = Klemens Böswirth k.boeswirth+git@gmail.com
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest shelltest nodep configurable
- infos/metadata =
- infos/description = one-line description of process

74.1 Introduction

This plugin spawns a new process with a user-defined executable and delegates all operations to the new process.

74.2 Usage

Set the config key `executable` and the arrays `args/#`, `env/#` to the path of an executable, the arguments that shall be passed and the environment variables to be set.

```
kdb mount test.dump /tests/process process 'executable=/usr/bin/pluginproc' 'args=#1'
      'args/#0=--load-plugin' 'args/#1=myplugin'
```

During `elektraStdprocioOpen` the plugin will collect the `args/#` and `env/#` values into two arrays `argv` and `envp`. Additionally, the array `copyenv/#` is read as a list of environment variables. Each variable will be looked up via `getenv` and then added to `envp`.

The plugin then `fork`s a new process and the child calls `execve` with the path from `app` as well as `argv` and`envp`. The child process is expected to listen to `stdin` and write to `stdout` according to the protocol described below.

If communication can be established, the child process will be kept running until `elektraStdprocioClose`.

74.3 Protocol

The entire protocol is text-based (apart from binary key values) and request-response-based, and happens over `stdin`/`stdout`. The parent process sends request to the child process. The child then processes the request and sends a response back.

To encode keysets and keys we use the format of the `dump` plugin. It is important to note that the `dump` plugin is instructed to write the full keynames (normally it removes the parent prefix). There is, however, an exception during initialization, which is described in the appropriate section. The child process may also delegate directly to the `dump` plugin, or reimplement the encoding.

The communications protocol used by the plugin has 3 phases:

- Initialization: Initial handshake after first starting the child-process.
- Operation: The main phase, which implements all the operations of an Elektra plugin (open, get, set, close, ...).
- Termination: The plugin is being closed and the parent-process tells the child to shut down.

Note: In the sections below we use the following format to describe messages:

```
Parent > Child
HELLO WORLD
[users]
(user)
{ok|error}
```

This denotes a message sent from parent to child, containing

- the literal text `HELLO WORLD`
- followed by a newline
- followed by a keyset called `users`
- followed by a newline
- followed by dynamic text called `user`
- followed by either the literal text `ok` or the literal text `error`

The names `users` and `user` don't actually appear in the message. They are only used as a reference for the descriptions of the message.

The actual message sent could look like this:

```
HELLO WORLD
kdbOpen 2
$key string 3 5
joe
12345
$end
appleseed
```

74.3.1 Initialization

The parent writes the protocol header to `stdin` of the child.

```
Parent > Child
ELEKTRA_PROCESS INIT v1
```

The child must respond with an acknowledgement, followed by a `contract` keyset.

```
Child > Parent
ELEKTRA_PROCESS ACK v1
(name)
[contract]
```

Here (name) is the module name the child uses. This will be used to replace process in the contract of the plugin. The [contract] keyset must contain the infos keys, which will be used replace the ones at the top of this README and the ones in process.c. It must also contain exports/_ keys for every operation that is implemented by the child. All keys in [contract] should be below system:/elektra/modules/process. The parent will rename these keys appropriately.

A child that implements all operations should include these keys (in addition to the relevant infos keys):
```
system:/elektra/modules/process/exports/open = 1
system:/elektra/modules/process/exports/close = 1
system:/elektra/modules/process/exports/get = 1
system:/elektra/modules/process/exports/set = 1
```

After this initial handshake, the child should simply wait for further requests from the parent.

> **Note**: Under normal circumstances this handshake will always be followed by an open request immediately.

74.3.2 Operation

When the parent needs the child to process an operation (open, get, ...), it will send a request like this:
```
Parent > Child
{open|get|set|close}
[parent]
[data]
```

First we send the opname (one of open, get, set or close) of the operation that shall be performed by the child. The keyset [parent] always consists of a single key, namely the parentKey (or errorKey) that was passed to the plugin. Finally, [data] is the keyset that was passed to the plugin. The [data] keyset is not present in open and close operations, since those don't receive a KeySet in the C API. However, in the open operation [data] is replaced [config] which is the KeySet returned by elektraPluginGetConfig in the C API. This is needed, because the child process cannot request the config keyset otherwise.

The child should then perform the requested operation and respond with
```
Child > Parent
{success|noupdate|error}
[parent]
[returned]
```

Here (result) is one of success, noupdate and error, which correspond to ELEKTRA_PLUGIN_↩ STATUS_SUCCESS, ELEKTRA_PLUGIN_STATUS_NO_UPDATE and ELEKTRA_PLUGIN_STATUS_ERROR respectively. The keysets [parent] and [returned] are the modified versions of the one sent by the parent.

74.3.3 Termination

When the parent no longer needs the child process, it will send a final termination request.
```
Parent > Child
ELEKTRA_PROCESS TERMINATE
```

The child process should now exit (and thereby close its ends of the stdin/stdout pipes).

> **Note:** Under normal circumstances this only happens, when plugin is being closed, i.e. during a elektraPluginClose call for a process instance.

74.3.4 Errors

If an unexpected error occurs on either side of the protocol, the connection should be terminated and the child process shall exit.

74.4 Examples

```
# mount the Whitelist Java Plugin via process
# NOTE: the copyenv and copyenv/#0 are normal not needed, but we need them to make this script work as an
       automated test
sudo kdb mount config.file user:/tests/process dump process 'executable=/usr/bin/java' 'args=#3'
       'args/#0=-cp' "args/#1=$BUILD_DIR/src/bindings/jna/plugins/whitelist/build/libs/whitelist-$(kdb
       --version | sed -nE 's/KDB_VERSION: (.+)/\1/gp')-all.jar"
       'args/#2=org.libelektra.process.PluginProcess' 'args/#3=org.libelektra.plugin.WhitelistPlugin'
       'copyenv=#0' "copyenv/#0=LD_LIBRARY_PATH"
# Define whitelist
kdb meta-set user:/tests/process/key "check/whitelist/#0" ""
kdb meta-set user:/tests/process/key "check/whitelist/#1" allowed0
kdb meta-set user:/tests/process/key "check/whitelist/#2" allowed1
# Should be allowed
kdb set user:/tests/process/key allowed0
#> Set string to "allowed0"
kdb set user:/tests/process/key allowed1
#> Set string to "allowed1"
# Should cause error
kdb set user:/tests/process/key not_allowed
# RET: 5
# STDERR:.*Validation Semantic: .*'not_allowed' does not adhere to whitelist.*
# cleanup
kdb rm -r user:/tests/process
sudo kdb umount user:/tests/process
```

Note: The mount line of the snippet above can be simplified by using the `mount-java` helper:

```
sudo kdb mount-java config.file user:/tests/process dump
 java:org.libelektra.plugin.WhitelistPlugin
```

74.5 Limitations

- The `error` and `commit` functions are currently not supported. Therefore, implementing a resolver is not supported.
- Exporting additional functions (e.g. `checkconf`) is currently not supported.
- With the current backend system, `process` can only be used for plugins in the `postgetstorage` or `presetstorage` positions.
- The executable must be defined as an absolute path during mounting.

Chapter 75

Plugin: profile

- infos = Information about the profile plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = postgetstorage presetstorage
- infos/status = libc nodep global preview unfinished
- infos/metadata =
- infos/description = helps switching between configuration profiles

75.1 Usage

Following the elektra keyname convention application configurations are stored under `/sw/org/myapp/#` and `/sw/org/myapp/#0/current`is the profile to be used. The `profile` plugin provides an easy way to switch configuration profiles.

The key `/sw/org/myapp/#0/profile` defines what profile should be used as `current`, e.g. `/sw/org/myapp/#0/profile = myprofile`. If a key `/sw/org/myapp/#0/myprofile/key` is found and no key `/sw/org/myapp/#0/current/key` exists an override key will be created linking `/sw/org/myapp/#0/currrent/key` to `/sw/org/myapp/#0/myapp/key` If neither `/sw/org/myapp/#0/current/` nor `/sw/org/myapp/#0/myprofile/key` is found, but `/sw/org/myapp/#0/%/key`, `/sw/org/myapp/#0/current` will be linked to `/sw/org/myapp/#0/%/key`.

So a cascading lookup will automatically implement following preferences (next to the namespace preferences):

1. Usage of key in `current`
2. Usage of key in the profile set with `profile`
3. Usage of key in the `%` fallback profile

75.2 Installation

See installation. The package is called `libelektra5-extra`.

75.3 Example

Suppose we have the configuration file `profile.ini` in `~/.config`:

```
cat profile.ini
#> []
#> profile = myprofile
#>
#> [current]
#> key2 = will win
#>
#> [myprofile]
#> key1 = test1
#> key2 = test2
#>
#> [%]
#> key2 = failed?
#> key3 = test3
```

Then we simply mount it *without* the profile plugin:

```
kdb mount profile.ini /sw/org/myapp/#0 ini
```

But we have to make sure that the profile plugin is mounted globally:

```
kdb global-mount profile
```

Then we can access `/sw/org/myapp/#0` in a profile-aware way:

```
kdb ls /sw
#> spec:/sw/org/myapp/#0/current/key1
#> spec:/sw/org/myapp/#0/current/key3
#> user:/sw/org/myapp/#0
#> user:/sw/org/myapp/#0/%
#> user:/sw/org/myapp/#0/%/key2
#> user:/sw/org/myapp/#0/%/key3
#> user:/sw/org/myapp/#0/current
#> user:/sw/org/myapp/#0/current/key2
#> user:/sw/org/myapp/#0/profile
#> user:/sw/org/myapp/#0/myprofile
#> user:/sw/org/myapp/#0/myprofile/key1
#> user:/sw/org/myapp/#0/myprofile/key2
```

As we can see with the `-v` option, we will fetch keys from our `myprofile` even though we request `current`:

```
kdb get -v /sw/org/myapp/#0/current/key1
#> got 25 keys
#> searching spec:/sw/org/myapp/#0/current/key1, found: spec:/sw/org/myapp/#0/current/key1, options:
       KDB_O_CALLBACK
#> The resulting keyname is user:/sw/org/myapp/#0/myprofile/key1
#> test1
```

To switch profile we simply have to set one key:

```
kdb set user:/sw/org/myapp/#0/profile newprofile
```

Usually, this will be done via commandline by setting `proc:/sw/org/myapp/#0/profile`.

Chapter 76

Plugin: python

- infos = Information about the python plugin is in keys below
- infos/author = Lukas Hartl git@lukashartl.at, Leonard Guelmino e1503940@student.↩ tuwien.ac.at
- infos/licence = BSD
- infos/provides = check
- infos/status = maintained
- infos/placements = postgetstorage presetstorage
- infos/description = checks if name is resolvable

76.1 DNS Python plugin

This filter plugin checks if a Key is a valid domain name, i.e. if the name can be resolved into an IPv4 address. The validation takes place whenever the `kdb set` method is called.

76.2 Usage

The python plugin requires the configuration parameter **script** holding the file path to the python script. The mount command looks like

```
sudo kdb mount file.ini /python ni python script=/path/to/dns_plugin.py
```

For the plugin to actually check a certain key, the `check/dns` metakey must be set.

```
sudo kdb meta-set spec:/python/my_hostname check/dns ”
```

Now each time the key is set, the filter plugin validates the given value.

```
kdb set user:/python/my_hostname www.libelektra.org
```

Chapter 77

Plugin: python

- infos = Information about the python plugin is in keys below
- infos/author = Manuel Mausz manuel-elektra@mausz.at
- infos/licence = BSD
- infos/provides =
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest configurable global memleak
- infos/description = proxy that calls other plugins (scripts) written in python

77.1 Introduction

The plugin uses Python to do magic things. It allows plugins to be written in Python.

What a Python script can do is not really limited by design, so any kind of plugin may be implemented. The python plugin is especially useful to write filter and logging scripts.

77.2 Installation

See installation. The package is called `libelektra5-python`.

77.3 Usage

The python plugin requires the configuration parameter **script** holding the file path to a python script. The mount command would look like

```
sudo kdb mount file.ini /python python script=/path/to/filter_script.py
```

if the **ini** plugin should be used for storage and the python plugin only serves to invoke the filter script.

For a Python script that serves as INI storage plugin itself, one uses

```
sudo kdb mount file.json /python python script=python_configparser.py
```

77.3.1 Plugin Configuration

The python plugin supports following optional configuration values/flags:

- `script` (string): The script to be executed.
- `print` (flag): Make the plugin print engine errors, triggered by the calls of this plugin, to stderr. Mainly intended for diagnostic. Please note that the Python engine itself will print script errors to stderr regardless of this flag.
- `python/path` (string): Extends sys.path by this entry. Better then PYTHONPATH it is always available, regardless of the context where a binary is executed.

77.3.2 Python Scripts

Python scripts must implement a class called `ElektraPlugin` with one parameter. The class itself can implement the following functions

- `open(self, config, errorKey)`
- `get(self, returned, parentKey)`
- `set(self, returned, parentKey)`
- `error(self, returned, parentKey)`
- `close(self, errorKey)`

where *config* & *returned* are KeySets and *errorKey* & *parentKey* are Keys. For the return codes of the functions, the same rules as for normal plugins apply.

If a function is not available, it simply is not called. A script does not have to implement all functions therefore.

Access to **kdb** can be retrieved using the Python import
```
import kdb
```

77.4 Example

An example script that prints some information for each method call would be:
```
import kdb
class ElektraPlugin(object):
    def open(self, config, errorKey):
        """
        returns
          * 0: no error
          * -1: error during initialization
        """
        print("Python script method 'open' called")
        return 0
    def get(self, returned, parentKey):
        """
        returns
          * 1: on success
          * 0: nothing was to do
          * -1: failure
        """
        print("Python script method 'get' called")
        return 1
    def set(self, returned, parentKey):
        """
        returns
```

```
        * 1: on success
        * 0: nothing was to do
        * -1: failure
      """
      print("Python script method 'set' called")
      return 1
  def error(self, returned, parentKey):
      """
      returns
        * 1: on success
        * 0: on success with no action
        * -1: failure
      """
      print("Python script method 'error' called")
      return 1
  def close(self, errorKey):
      print("Python script method 'close' called")
      return 0
```

Further examples can be found in the python directory.

77.5 Disclaimer

Be aware that a Python script will never be as performant as a native C/C++ plugin. Spinning up the interpreter takes additional time and resources.

Chapter 78

Plugin: quickdump

78.1 Benchmarks

78.1.1 <tt>benchmark_storage</tt>

The following table shows a summary of the results of a `benchmark_storage` run: NOTE: `factor` is always `dump / quickdump` (bigger = `quickdump` is faster), this test was only run with version 1, see below for a comparison of with version 2

operation	dump (µs)	quickdump (µs)	factor
write keyset	81661	4741	17.224
read keyset	94782	44673	2.122
iterate keyset	125	135	0.926
delete keyset	1823	3518	0.518
re-read keyset	92939	43749	2.124
strcmp key name	609	1826	0.334
strcmp key value	684	761	0.899

78.1.2 <tt>benchmark_plugingetset</tt>

A script was used to generate KeySets with `2`, `200`, `2,000`, `200,000` and `2,000,000` Keys. For keynames we used increasing numbers (prefixed with _ to keep correct order), each key had a randomized value of 15 characters. Additionally we added 4 metakeys to each key. The names and values of these were created similarly.

The parent key used was `dir:/tests/bench`.

We then used `hyperfine` to compare the performance of `dump` and `quickdump`.

```
hyperfine --min-runs 25 'benchmark_plugingetset <path> <parent> dump' 'benchmark_plugingetset <path>
      <parent> quickdump'
```

This means we used the same key sets for each of the runs, but the key values shouldn't really matter. It also means that no shared metadata (`keyCopyMeta`) was involved.

We also used a modified version of `benchmark_plugingetset` that returned after the `get` call, to test the `get` performance by itself. Because `set` doesn't scale linearly because of the checks for `keyCopyMeta`, the largest KeySet was only used for the `get` only version. (The test was aborted after the initial run of `dump` took longer than all the runs for `quickdump` together.)

The complete results can be found in `this repository` and the files used for these benchmarks are quite big and can be found in another `separate repository`.

78.1.2.1 Raw data sizes

For reference in the file size tests below, here is a list of the raw data sizes of the keysets. These sizes are calculated as the sum of: 1) length of all keynames 2) length of all key values 3) length of all metakeys 4) length of all meta values All lengths include a null terminator, sizes are calculated once with (WP) and once without the parent key (WOP).

no. of keys	raw size WP (B)	raw size WOP (B)
2	252	220
200	25784	22584
2000	262786	229786
200000	26977790	23777790
20000000	273777792	241777792

78.1.2.2 Version 1

78.1.2.2.1 File sizes `factor` is `dump / quickdump` like above (bigger = `quickdump` has smaller files)

no. of keys	dump (B)	quickdump (B)	factor
2	430	412	1.04
200	41206	40988	1.01
2000	415807	413788	1.00
200000	42377809	42177788	1.00
2000000	427777810	425777788	1.00

78.1.2.2.2 get and set The values are mean ± standard deviation. `factor` is `dump / quickdump` like above

no. of keys	dump (s)	quickdump (s)	factor
2	0.0016 ± 0.0003	0.0006 ± 0.0001	2.67
200	0.0091 ± 0.0009	0.0018 ± 0.0000	5.06
2000	0.0770 ± 0.0030	0.0127 ± 0.0008	6.06
200000	10.7876 ± 4.2564	1.2640 ± 0.0061	8.53

78.1.2.2.3 get only The values are mean ± standard deviation. `factor` is `dump / quickdump` like above

no. of keys	dump (s)	quickdump (s)	factor
2	0.0014 ± 0.0000	0.0005 ± 0.0000	2.8
200	0.0032 ± 0.0005	0.0013 ± 0.0002	2.46
2000	0.0179 ± 0.0002	0.0086 ± 0.0008	2.08
200000	1.6830 ± 0.0222	0.8515 ± 0.0048	1.98
2000000	17.1814 ± 0.2201	8.8808 ± 0.0344	1.93

78.1.2.3 Version 2

Running the same `benchmark_plugingetset` tests with version 2 of the plugin yields:

78.1.2.3.1 File sizes `factor` is `v1 / v2` like above (bigger is better)

no. of keys	quickdump v1 (B)	quickdump v2 (B)	factor
2	412	380	1.08
200	40988	23788	1.72
2000	413788	381788	1.08
200000	42177788	38977788	1.08
2000000	425777788	393777788	1.08

78.1.2.3.2 get and set The values are mean ± standard deviation. `factor` is `v1 / v2`, i.e. bigger is better

no. of keys	quickdump v1 (s)	quickdump v2 (s)	factor
2	0.0006 ± 0.0001	0.0013 ± 0.0035	0.47
200	0.0018 ± 0.0000	0.0018 ± 0.0002	1.02
2000	0.0127 ± 0.0008	0.0112 ± 0.0021	1.13
200000	1.2640 ± 0.0061	1.2602 ± 0.0372	1.00
2000000	-	13.2104 ± 0.1185	-

78.1.2.3.3 get only The values are mean ± standard deviation. `factor` is `v1 / v2` like above

no. of keys	quickdump v1 (s)	quickdump v2 (s)	factor
2	0.0005 ± 0.0000	0.0006 ± 0.0001	0.88
200	0.0014 ± 0.0002	0.0010 ± 0.0001	1.33
2000	0.0086 ± 0.0008	0.0072 ± 0.0005	1.20
200000	0.8516 ± 0.0048	0.6413 ± 0.0082	1.33
2000000	8.8809 ± 0.0345	6.3756 ± 0.0443	1.39

78.1.2.4 Version 3

Running the same `benchmark_plugingetset` tests with version 3 of the plugin yields:

78.1.2.4.1 File sizes `factor` is `v1 / v2` like above (bigger is better)

no. of keys	quickdump v2 (B)	quickdump v3 (B)	factor
2	380	240	1.58
200	23788	23788	1.00
2000	381788	241788	1.58
200000	38977788	24977788	1.56
2000000	393777788	253777788	1.55

78.1.2.4.2 get and set The values are mean ± standard deviation. `factor` is `v2 / v3`, i.e. bigger is better

no. of keys	quickdump v2 (s)	quickdump v3 (s)	factor
2	0.0013 ± 0.0035	0.0012 ± 0.0029	1.08
200	0.0018 ± 0.0002	0.0017 ± 0.0002	1.05
2000	0.0112 ± 0.0021	0.0106 ± 0.0011	1.06
200000	1.2602 ± 0.0372	1.1473 ± 0.0287	1.10
2000000	13.2104 ± 0.1185	11.5082 ± 0.2624	1.15

78.1.2.4.3 get only The values are mean ± standard deviation. `factor` is `v2 / v3` like above

no. of keys	quickdump v2 (s)	quickdump v3 (s)	factor
2	0.0006 ± 0.0001	0.0006 ± 0.0001	0.99
200	0.0010 ± 0.0001	0.0012 ± 0.0001	0.84
2000	0.0072 ± 0.0005	0.0070 ± 0.0004	1.03
200000	0.6413 ± 0.0082	0.6288 ± 0.0218	1.02
2000000	6.3756 ± 0.0443	6.2309 ± 0.0462	1.02

Chapter 79

Plugin: quickdump

- infos = Information about the quickdump plugin is in keys below
- infos/author = Klemens Böswirth `k.boeswirth+git@gmail.com`
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/quickdump
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained compatible unittest tested nodep libc configurable preview
- infos/metadata =
- infos/description = much quicker version of dump (2x or more in most cases)

79.1 Introduction

`quickdump` is a storage plugin based on the `dump` format. It is a lot quicker (see benchmarks.md) than the old `dump` plugin, because it does not use commands and stores string lengths as binary data. Through these changes all string comparisons and integer-string conversions can be eliminated, which made up for a lot of the time spent by the `dump` plugin.

The format is also useful for IPC and streaming, because of this it is used by the `specload` plugin.

79.2 Format

A `quickdump` file starts with the magic number `0x454b444200000003`. The first 4 bytes are the ASCII codes for `EKDB` (for Elektra KDB), followed by a version number. This 64-bit is always stored as big-endian (i.e. the way it is written above).

After the magic number the file is just a list of Keys. Each Key consists of a name, a value and any number of metakey names and values. Each name and value is written as a 64-bit length `n` followed by exactly `n` bytes of data. For strings we do not store a null terminator. Therefore the length also does not account for that. When reading a string, the plugin allocates `n+1` bytes and sets the last one to `0`. Note that ALL lengths are stored in

little-endian format, because most modern machines are little-endian. To save disk space, we use a variable length encoding for integers. The exact format is described below.

We don't store the full name of the key. Instead we only store the name relative to the parent key.

The end of a key is marked by a null byte. This cannot be confused with null bytes embedded in binary key values, because of the length prefixes before each key and metavalue.

To distinguish between binary and string keys the (length of the) key value is prefixed with either a `b` or an `s`. Each metakey is prefixed with an `m`, unless we detect that the same metakey was already present on a previous key (e.g. through `keyCopyMeta`). In this case the prefix `c` is used and instead of the metakey name and value, we write the name of the previous key and the metakey name.

79.2.1 Variable Length Integer encoding

The basic idea of the format is to store integers in base 128. This means we only use 7 bits per byte and the 8th bit (marker bit) indicates whether or not there are more bytes to read. However, to make things more efficient we move all those marker bits to the first byte. Then we can read one byte and immediately know, how much bytes follow. This is similar to what UTF-8 does.

The table below shows how the encoding works. The first byte is shown in full (`x` is either `0` or `1`), then `[n]` indicates that `n` bytes of data follow.

```
xxxxxxx1     up to  7 bits in 1 byte
xxxxxx10 [1] up to 14 bits in 2 bytes
xxxxx100 [2] up to 21 bits in 3 bytes
xxxx1000 [3] up to 28 bits in 4 bytes
xxx10000 [4] up to 35 bits in 5 bytes
xx100000 [5] up to 42 bits in 6 bytes
x1000000 [6] up to 49 bits in 7 bytes
10000000 [7] up to 56 bits in 8 bytes
00000000 [8] up to 64 bits in 9 bytes
```

Thanks to https://github.com/stoklund/varint for listing various integer encodings.

79.3 Usage

Like any other storage plugin, you simply use `quickdump` during mounting, import or export.

```
sudo kdb mount quickdump.eqd user:/tests/quickdump quickdump
```

79.4 Dependencies

None.

79.5 Examples

```
# Mount a backend using quickdump
sudo kdb mount quickdump.eqd user:/tests/quickdump quickdump
# RET: 0
# Set some keys and metakeys
kdb set user:/tests/quickdump/key value
#> Create a new key user:/tests/quickdump/key with string "value"
kdb meta-set user:/tests/quickdump/key meta "metavalue"
kdb set user:/tests/quickdump/otherkey "other value"
#> Create a new key user:/tests/quickdump/otherkey with string "other value"
# Show resulting file (not part of test, because xxd is not available everywhere)
# xxd $(kdb file user:/tests/quickdump/key)
# 00000000: 454b 4442 0000 0003 076b 6579 730b 7661  EKDB.....keys.va
# 00000010: 6c75 656d 096d 6574 6113 6d65 7461 7661  luem.meta.metava
# 00000020: 6c75 6500 116f 7468 6572 6b65 7973 176f  lue..otherkeys.o
# 00000030: 7468 6572 2076 616c 7565 00              ther value.
# Change mounted file (in a very stupid way to enable shell-recorder testing):
cp $(kdb file user:/tests/quickdump/key) a.tmp
# 1. change key from 'value' to 'other value'
(head -c 13 a.tmp; printf "%bother value" '\0027'; tail -c 40 a.tmp) > b.tmp
rm a.tmp
# 2. add copy metadata instruction to otherkey
(head -c 64 b.tmp; printf "c%bkey%bmeta\0" '\0007' '\0011') > c.tmp
rm b.tmp
# test file name, so KDB isn't destroyed if mounting failed
[ "x$(kdb file user:/tests/quickdump/key)" != "x$(kdb file user:/)" ] && mv c.tmp $(kdb file
       user:/tests/quickdump/key)
kdb get user:/tests/quickdump/key
#> other value
kdb meta-get user:/tests/quickdump/key meta
#> metavalue
kdb get user:/tests/quickdump/otherkey
#> other value
kdb meta-get user:/tests/quickdump/otherkey meta
#> metavalue
# Cleanup
kdb rm -r user:/tests/quickdump
sudo kdb umount user:/tests/quickdump
```

79.6 Limitations

None.

Chapter 80

Plugin: range

- infos = Information about the range plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = presetstorage postgetstorage
- infos/status = maintained conformant compatible coverage specific unittest tested libc preview unfinished
- infos/metadata = check/range check/type type
- infos/description = tests if a value is within a given range

80.1 Introduction

The range plugin checks if a `Key`'s value is within a given range.

80.2 Installation

See installation. The package is called `libelektra5-experimental`.

80.3 Usage

- The plugin checks every `Key` in the `KeySet` having a metakey `check/range`.
- For these keys, it checks whether the key value is within the specified range.
- `check/range` can contain either:
  1. a single range with the syntax `[-]min-[-]max`
  2. or a list of ranges or values separated by `,`
- Metakey `check/type` can be used to specify the data type. If not specified, metakey `type` will be used as fallback. If both are unspecified or unsupported, the type is assumed to be `long long`.

Possible values:

- `short`, `long`, `long long`
  for signed integer values
- `unsigned short`, `unsigned long`, `unsigned long long`
  for unsigned integer values
- `float`, `double`, `long double`
  for floating point values
- `HEX`
  for hexadecimal values
- `char`
  for characters

80.4 Dependencies

None.

80.5 Examples

```
# Backup-and-Restore:/tests/range
sudo kdb mount range.ecf /tests/range range dump
# should succeed
kdb set user:/tests/range/value 5
kdb meta-set spec:/tests/range/value check/range "1-10"
# RET: 0
# should fail
kdb set user:/tests/range/value 11
# RET:5
# should also fail
kdb set user:/tests/range/value "\-1"
# RET:5
# we can also allow only individual values: (using the --force flag, as the current value of 5 would not be
       allowed under the new policy)
kdb meta-set -f spec:/tests/range/value check/range "1,2,4,8"
kdb set user:/tests/range/value 7
# RET:5
kdb set user:/tests/range/value 2
# RET:0
kdb rm -r user:/tests/range
sudo kdb umount /tests/range
```

80.6 Limitations

None.

Chapter 81

elektra-plugins(7) – plugins overview

Multiple plugins can be mounted into the key database (KDB). On every access to the key data base they are executed and thus can change the functionality and behavior.

Unlike Elektra's core the plugins have all kinds of dependencies. It is the responsibility of the plugin to find and check its dependencies using CMake. If a dependency cannot be found, the plugin will automatically disable itself.

81.1 Description

Elektra has a wide range of different plugins. The plugin folders should contain a README.md with further information. (Or follow links below.) The plugins are:

81.1.1 C-Interface

All plugins implement the same interface:

- `kdbOpen()` calls `elektraPluginOpen()` of every plugin to let them do their initialisation.
- `kdbGet()` requests `elektraPluginGet()` of every plugin in the queried backends to return a key set.
- `kdbSet()` usually calls `elektraPluginSet()` of every plugin in the queried backends to store the configuration.
- `kdbSet()` also calls `elektraPluginError()` for every plugin when an error happens. Because of `elektraPluginError()`, plugins are guaranteed to have their chance for necessary cleanups.
- `kdbClose()` makes sure that plugins can finally free their own resources in `elektraPluginClose()`.

Furthermore, plugins might export symbols:

- `checkconf` can be called during mounting to ensure a plugin has valid configuration.
- `genconf` can be called to produce all valid configurations of a plugin.

81.1.2 KDB-Interface

- To list all plugins use kdb-plugin-list(1).
- To check a plugin use kdb-plugin-check(1).
- For information on a plugin use kdb-plugin-info(1).
- For mount plugin(s) use kdb-mount(1).

81.2 Installation

See INSTALL. Many plugins are already part of the core package `libelektra5`. The package that includes a plugin which does not belong to the `libelektra5` package can be found in it's README.md.

81.3 See Also

For an easy introduction, see this tutorial how to write a storage plugin. For more background information of the plugins framework, continue here. Otherwise, you can visit the `the API documentation`.

81.4 Plugins

81.4.1 Backends

Backends provide access to different data sources (e.g. files, databases, network resources)

- backend is the default plugin implementing backend functionality for configuration files
- backend_odbc provides access to ODBC data sources for storing configuration data (keys, values, metadata)

81.4.2 Resolver

Before configuration is actually written, the file name needs to be determined (resolvers will be automatically added by kdb mount):

- resolver uses advanced POSIX APIs to handle conflicts gracefully
- noresolver does not resolve as no file name is needed (for non-file storage plugins)
- wresolver minimalistic resolver for non-POSIX systems

Furthermore, there are following experimental resolvers:

- blockresolver resolves tagged blocks inside config files
- curlget fetches configuration file from a remote host
- gitresolver checks out and commits files to a local Git repository

81.4.3 Storage

Are responsible for reading writing the configuration to configuration files.

Read and write everything a KeySet might contain:

- dump makes a dump of a KeySet in an Elektra-specific format
- quickdump uses binary portable format based on dump, but more efficient
- mmapstorage uses binary, not portable memory mapped file for a high performance storage

Read (and write) standard config files:

- toml reads and writes data using a parser generated by `Flex` and `Bison`
- hosts reads/writes hosts files
- kconfig reads/writes KConfig ini files
- line reads/writes any file line by line
- yajl reads/writes JSON.
- augeas reads/writes many different configuration files using the Augeas library
- xfconf reads/writes to arbitrary xfconf channels

Using semi-structured data for config files, mainly suitable for spec-namespace (put a focus on having nice syntax for metadata):

- ni parses INI files based on (including metadata)

Only suited for import/export:

- mini dependency free, line based key-value storage plugin.
- simpleini line-based key-value pairs with configurable format (without sections). Only works on glibc systems.
- xerces uses XML (without a specific schema).
- xmltool uses XML in the deprecated Elektra XML schema for importing configuration from Elektra 0.7.
- c writes Elektra C-structures (`ksNew(..` `keyNew(...)`
- ansible writes the given KeySet as Ansible Playbook that uses the `ansible-libelektra` module

Plugins that just show some functionality, (currently) not intended for productive use:

- csvstorage for csv files
- dpkg reads /var/lib/dpkg/{available,status}
- file reads and writes a file from/to a single key
- fstab for fstab files.
- mozprefs for Mozilla preference files
- passwd for passwd files
- specload calls an external application to request its specification, depends on quickdump
- yamlcpp reads and writes data in the `YAML` format using `yaml-cpp`

81.4.4 System Information

Information compiled in Elektra:

- version is a built-in plugin directly within the core so that it cannot give wrong version information
- constants various constants fixed when Elektra was compiled
- desktop contains information which desktop is currently running

Providing information found on the system not available in persistent files:

- uname information from the uname syscall.

81.4.5 Filter

Filter plugins process keys and their values in both directions. In one direction they undo what they do in the other direction. Most filter plugins available now encode and decode values. Storage plugins that use characters to separate key names, values or metadata will not work without them.

Rewrite unwanted characters within strings (**code**-plugins):

- ccode using the technique from arrays in the programming language C
- hexcode using hex codes

Rewrite unwanted characters within binary data (**binary**-plugins):

- base64 using the Base64 encoding scheme (RFC4648)

Other filters:

- crypto encrypts / decrypts confidential values
- fcrypt encrypts / decrypts entire files
- gpgme encrypts / decrypts confidential values (with GPGME)
- iconv makes sure the configuration will have correct character encoding

Experimental transformations (are **not** recommended to be used in production):

- directoryvalue converts directory values to leaf values
- hexnumber converts between hexadecimal and decimal
- keytometa transforms keys to metadata
- rename renames keys according to different rules
- profile renames keys according to current profile

81.4.6 Notification and Logging

Log/Send out all changes to configuration to:

- dbus sends notifications for every change via dbus `notification`
- journald logs key database changes to journald
- syslog logs key database changes to syslog
- zeromqsend sends notifications for every change via ZeroMQ sockets `notification`

Notification of key changes:

- internalnotification get updates automatically when registered keys were changed
- dbusrecv receives notifications via dbus `notification`
- zeromqrecv receives notifications via ZeroMQ sockets `notification`

81.4.7 Debug

Trace everything that happens within KDB:

- counter count and print how often a plugin is used
- timeofday prints timestamps
- tracer traces all calls
- iterate iterate over all keys and run exported functions on tagged keys
- logchange prints the change of every key on the console

81.4.8 Checker

Copies metadata to keys:

- glob using globbing techniques (needed by some plugins)
- spec copies metadata from spec namespace (the standard way)

Plugins that check if values are valid based on metadata (typically copied by the `spec` plugin just before) to validate values:

- type type checking (CORBA types) with enum functionality
- ipaddr checks IP addresses using regular expressions
- email checks email addresses using regular expressions
- network by using network APIs
- path by checking files on file system

- unit validates and normalizes units of memory (e.g. 20KB to 20000 Bytes)
- blacklist blacklist and reject values
- length validates that string length is less or equal to given value

The same but experimental:

- conditionals by using if-then-else like statements
- date validates date and time data
- mathcheck by mathematical expressions using key values as operands
- macaddr checks if MAC addresses are valid and normalizes them
- range checks if a value is within a given range
- reference checks if a value is a valid reference to another key
- rgbcolor validates and normalizes hexcolors
- validation by using regex

Other validation mechanisms not based on metadata:

- filecheck does sanity checks on a file
- lineendings tests file for consistent line endings

81.4.9 Interpreter

These plugins start an interpreter and allow you to execute a script in an interpreted language whenever Elektra's key database gets accessed. Note that they depend on the presence of the respective binding during run-time:

- jni java plugins started by jni, works with jna plugins
- lua Lua plugins
- python Python 3 plugins
- ruby Ruby plugins
- shell executes shell commandos

81.4.10 Other Important Plugins

- cache caches keysets from previous `kdbGet()` calls
- sync uses POSIX APIs to sync configuration files with the hard disk
- gopts global plugin to automatically call `elektraGetOpts`
- process proxy plugin that uses separate executables as plugin implementations
- recorder is the plugin used to implement session recording

81.4.11 Plugins for Development

- template to be copied for new plugins
- cpptemplate a template for C++ based plugins
- doc contains the documentation of the plugin interface
- error yields errors as described in metadata (handy for test purposes)

81.4.12 Internal Plugins

Internally used and hard coded into `libelektra-kdb`. Don't try to use manually.

- missing
- modules
- version

81.4.13 Deprecated Plugins

Please avoid, if possible, to use following plugins:

- (currently none party popper)

Chapter 82

Plugin: recorder

- infos = Information about the recorder plugin is in keys below
- infos/author = Maximilian Irlinger max@maxirlinger.at
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = hook
- infos/status = recommended productive maintained conformant unittest nodep configurable nodoc
- infos/metadata =
- infos/description = provides session recording functionality

82.1 Introduction

This plugin's sole responsibility is to act as a hook plugin for the `libelektra-record` library. For internal use by `libelektra-kdb` only. Intentionally not documented here.

82.2 Plugin Configuration

You can specify the path and name of the lockfile with the `lockfile` parameter. Default is `/tmp/elektra_↩ record.lock`.

82.3 Dependencies

This plugin depends on the `libelektra-record` library.

Chapter 83

Plugin: reference

83.1 Example: Alternative References

For this example the following configuration is used:

```
; this example uses the ni plugin syntax
; 1. metadata
[rootkey/ref]
check/reference = recursive
[otherkey/newref]
check/reference = alternative
; 2. configuration data
rootkey/ref = ../otherkey
otherkey/ref = ../yetanotherkey
otherkey/newref = ../otherotherkey
otherotherkey/ref = ../nonexistent
otherotherkey/newref = ../mergekey
yetanotherkey/ref = ../mergekey
yetanotherkey/newref = ../nonexistent
mergekey/ref = ../finalkey
mergekey/newref = ../finalkey
finalkey = ""
```

We mount it by executing:

```
cat « EOF > alternative.ini
; this example uses the ni plugin syntax
; 1. metadata
[rootkey/ref]
check/reference = recursive
[otherkey/newref]
check/reference = alternative
; 2. configuration data
rootkey/ref = ../otherkey
otherkey/ref = ../yetanotherkey
otherkey/newref = ../otherotherkey
otherotherkey/ref = ../nonexistent
otherotherkey/newref = ../mergekey
yetanotherkey/ref = ../mergekey
yetanotherkey/newref = ../nonexistent
mergekey/ref = ../finalkey
mergekey/newref = ../finalkey
finalkey = ""
EOF
kdb mount alternative.ini user:/tests/reference/alternative ni reference
```

The file contains a specification, which marks `user:/tests/reference/alternative/rootkey/ref` as our root reference key and `user:/tests/reference/alternative/otherkey/newref` as an alternative reference.

The actual configuration contains then a structure which is processed by the plugin like follows:

1. `user:/tests/reference/alternative/rootkey/ref` is read and its reference validated.

2. `user:/tests/reference/alternative/otherkey/ref` is read and its reference validated.
3. `user:/tests/reference/alternative/otherkey/newref` is discovered as an alternative reference and stored for later.
4. `user:/tests/reference/alternative/yetanotherkey/ref` is read and its reference validated.
5. `user:/tests/reference/alternative/mergekey/ref` is read and its reference validated.
6. `user:/tests/reference/alternative/finalkey` does not contain a reference, so we stop here.
7. Processing of the alternative reference chain from `user:/tests/reference/alternative/otherkey/newref` starts.
8. `user:/tests/reference/alternative/otherkey/newref` is read and its reference validated.
9. `user:/tests/reference/alternative/otherotherkey/newref` is read and its reference validated.
10. `user:/tests/reference/alternative/mergekey/newref` is read and its reference validated.
11. `user:/tests/reference/alternative/finalkey` does not contain a reference, so we stop here.

The resulting reference graph looks like this:

As you can see, the plugin completely ignores `user:/tests/reference/alternative/yetanotherkey/newref` as well as `user:/tests/reference/alternative/otherother/ref` and in turn does not throw an error because `user:/tests/reference/alternative/nonexistent` does not exist. However, the plugin does still exhaustively check both of the alternative reference chains. If you want to prohibit this, you can set the metakey `check/reference/restrict` to an empty value on whichever key you want to be a leaf node in the graph.

After we are done we can unmount the example with:
```
kdb umount user:/tests/reference/alternative
```

Chapter 84

Plugin: reference

84.1 Example: Validating Complex Recursive Structures

Suppose you have some mutually recursive `struct`s in C. You want to map this structure onto a hierarchy inside your KDB. This by itself was always possible, but the reference plugin in combination with the spec plugin, now allows for this in a way that can be validated and checked by Elektra.

The structure we will use for this example is easily defined in C code:

```
struct typeA {
  char *name;
  struct typeB *ref;
};
struct typeB {
  long int id;
  struct typeA *ref;
};
struct typeA *rootkey_ref;
```

Starting with a reference to an element of type `typeA`, we except an alternating chain of `typeA` and `typeB`. In this chain each element of type `typeA` shall have a string attached to it, while elements of type `typeB` shall contain an integer.

The specification used for such a structure is as follows (using the syntax of the `ni` plugin):

```
[rootkey/ref]
check/reference = recursive
check/reference/restrict = ../typeA/_
[typeA/_]
default = ""
[typeA/_/name]
check/type = string
[typeA/_/ref]
check/reference/restrict = ../../typeB/_
[typeB/_]
default = ""
[typeB/_/id]
check/type = long
[typeB/_/ref]
check/reference/restrict = ../../typeA/_
```

The basic idea is to use the keys `typeA` and `typeB` as a sort of 'directory', in which every 'file' represent an element of type `typeA` or type `typeB` respectively.

To achieve our goals of type validation, we specify `rootkey/ref` to be the root of our reference graph, but also restrict the possible reference to keys directly below `typeA`. That way we ensure that any reference set in `rootkey/ref` will refer to a key which is, validated to be compatible with our `struct typeA` by the spec and type plugins.

Using the spec plugin we then specify all direct children of `typeA` to have a default value of `""`. This ensures that these keys exist and therefore can be referenced. [1] Then we simply specify the `name` sub-key for all of these children to be a string, and restrict the possible reference to direct children of `typeB`. From there we proceed similarly for the children of `typeB` and we are done.

[1] It would actually be better to only give these keys a value, if the `name` sub-key exists, but there is currently no good way to do that in Elektra. An alternative would be to use the keys themselves, i.e. give them an actual value.

Chapter 85

Plugin: reference

- infos = Information about the reference plugin is in keys below
- infos/author = Klemens Böswirth `k.boeswirth+git@gmail.com`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = presetstorage
- infos/status = maintained unittest libc nodep writeonly
- infos/metadata = check/reference check/reference/restrict check/reference/restrict/#
- infos/description = Plugin for validating singular or recursive references

85.1 Introduction

This plugin serves one single purpose: the validation of references from key to another inside the KDB.
To specify a key as a reference add the metakey `check/reference`. Currently the metakey supports the three values `single`, `recursive` and `alternative`. While the value `alternative` only makes sense in a certain context inside the `recursive` mode, the other two values define to different modes of operation for the plugin.
If a key has the metakey `check/reference` with value `single`, the plugin reads the value of the key and produces an error, if the value is not a valid (i.e. resolvable) reference.
If a key has metakey `check/reference` with value `recursive`, the plugin constructs a reference graph starting with the marked key. (for the exact construction see below) In this reference graph each node corresponds to a key in the KDB, while each edge represents a reference between to keys. The plugin will produce an error, if this graph is not a directed acyclic graph or contains any invalid references.

85.2 Installation

See installation. The package is called `libelektra5-experimental`.

85.2.1 Resolution of References

The plugin will try to resolve all keys marked with the metakey `check/reference` as references. The only exception to this rule is, if the value of such a key is a valid array name (i.e. `#0`, `#_10`, ...). In that case, the plugin will try to resolve the values of the array elements, directly below the marked key. And treats the marked key as referencing multiple other keys.
The resolution of the references into key names goes as follows:

- If the reference value starts with `./` or `../`, it will be interpreted as relative to the current key or the parent of the current key.
- If the reference value starts with `@/`, it will be relative to the parent-key used in the call to `kdbGet`.
- All other values are treated as absolute and interpreted as a keyname directly.
- `.` and `..` will be treated the same way as in Unix paths. However, the plugin will emit warnings, if `.` is used anywhere other than before the first slash. Equally using `..` after a path segment other than `..` itself, will result in a warning. This is because these use cases are redundant and might be (or result in) mistakes. Here are a few examples of what results in warnings, and what doesn't:
  - `./system:/key` no warning
  - `system:/key` no warning
  - `system:/./key` warning, redundant use of `.`
  - `../../../key` no warning
  - `../key/../otherkey`warning, redundant use of `.`

85.2.2 Construction of the Reference Graph

The process starts with a key `X`, which has the metakey `check/reference` set to the value `recursive`. This key `X` must be an array key. The basename of `X` will be called the `refname`. For each element in the array run the following process:

1. Interpret the current key `K`s value as a reference and throw an error, if the reference is not valid.
2. If there is no node for `K` in the reference graph create one. If there is no node for `R` in the reference graph create one. Insert an edge from `K` to `R` into the graph.
3. Find a key `K1` directly below `R` that has the `refname` as its basename. Interpret this key as an array if found. For each array element, start the process recursively from 1 and then continue with 4.
4. Find any key `K2` directly below `R` that has the metakey `check/reference` set to `alternative`. Interpret any such key as an array. For each array element, start the process recursively from 1, but this time using the basename of `K2` (the of the array) as the `refname`.

Once the whole process is finished, we will have a graph of the whole reference structure stemming from `X`. This graph can then be check for acyclicity.

85.2.3 Restriction of References

Without any additional intervention, the plugin accepts the name of any existing key as valid reference value. Existing in this context means, the key either has a value or metadata associated with it. In recursive mode a key is also said to exist, if a key with the current `refname` as a basename exists (has value or metadata) directly below the key in question. If, however, the key containing the reference value has the `check/reference/restrict` metakey set (even if the value is empty), its value has to be taken into account.
If the value `check/reference/restrict` is anything but a valid array-element-name, i.e. a `#` followed by `n` underscores (_) and `n+1` digits (`0-9`), its value will be used directly. Otherwise `check/reference/restrict` has to be a valid array and its elements will be treated as alternative restriction. Only one of these restriction has to be fulfilled for a reference to be valid.
Each restriction is first resolved as if it was a reference itself (see `Resolution of references`). The resolved value is then used as an Elektra-style globbing pattern. For the supported see elektra-globbing.
If a keyname matches the globbing expression it will be considered a valid reference (as long as the key actually exists).
Note: If the restriction is the empty string **""** **no** keyname will match, meaning the reference key annotated with this metadata has to be a leaf node in the reference graph, and therefore cannot have a value (other than the empty string **""**).

85.3 Usage

```
# Mount the plugin
sudo kdb mount referencetest.dump user:/tests/reference dump reference
# Mark a key as a single reference
kdb meta-set user:/tests/reference/singleref check/reference single
# Try setting an invalid reference
kdb set user:/tests/reference/singleref user:/tests/reference/referred1
# RET: 5
# STDERR: .*Reference 'user:/tests/reference/referred1', set in key 'user:/tests/reference/singleref', does
       not reference an existing key.*
# Create referred key ...
kdb set user:/tests/reference/referred1 ""
#> Create a new key user:/tests/reference/referred1 with string ""
# ... and try again
kdb set user:/tests/reference/singleref user:/tests/reference/referred1
#> Set string to "user:/tests/reference/referred1"
# Cleanup
kdb rm user:/tests/reference/singleref
kdb rm user:/tests/reference/referred1
# Unmount the plugin
sudo kdb umount user:/tests/reference
```

85.4 Examples

The examples directory contains a few examples:

- alternative shows how the `alternative` value of `check/reference` gets processed.
- complex shows how the plugin can be used together with the spec plugin, to validate complex recursive structures.

Chapter 86

Plugin: rename

- infos = Information about rename plugin is in keys below
- infos/author = Felix Berlakovich `elektra@berlakovich.net`
- infos/licence = BSD
- infos/provides = filter
- infos/needs =
- infos/placements = presetstorage postgetstorage
- infos/status = maintained unittest nodep libc configurable
- infos/metadata = rename/to rename/toupper rename/tolower rename/cut origname
- infos/description = renaming of keys

86.1 Introduction

This plugin can be used to perform rename operations on keys. This is useful if a backend does not provide keys with the required names or case.
If keys are renamed, their original name is stored in the `origname` MetaKey.
There are 2 types of transformations:

- basic
- advanced

86.2 Basic Transformations

are applied before and after the advanced transformations.

86.2.1 Get

first transformation to be applied to all keys on kdbGet.
`get/case`

- toupper
- tolower
- unchanged // this is the default value if no configuration is given

converts the whole keyname below the parentKey to upper- or lowercase. if no configuration or `unchanged` is used no transformation is done here.

86.2.2 Set

last transformation to be applied to all keys on kdbSet.
`set/case`

- toupper
- tolower
- keyname
- unchanged // this is the default value if no configuration is given

`toupper` or `tolower` tells the rename plugin to convert the whole keyname below the parentKey to lower or uppercase.
`unchanged` returns the key to its original name
`keyname` tells the plugin to keep the name of the advanced transformation

86.3 Advanced Transformations

86.3.1 Cut

86.3.1.1 Operation

The cut operation can be used to strip parts of a key's name. The cut operation is able to cut anything below the path of the parent key. A renamed key may even replace the parent key. For example consider a KeySet with the parent key `user:/config`. If the KeySet contained a key with the name `user:/config/with/long/path/key1`, the cut operation would be able to strip the following key name parts:

- with
- with/long
- with/long/path
- with/long/path/key1

86.3.1.2 Configuration

The cut operation takes as its only configuration parameter the key name part to strip. This configuration can be supplied in two different ways. First, the global configuration key `cut` can be used. Second, keys to be stripped can be tagged with the MetaKey `rename/cut`. If both options are given, the MetaKey takes precedence. For example, consider the following setup:

```
config/cut = will/be
parent key = user:/config
user:/config/will/be/stripped/key1              <- meta rename/cut = will/be/stripped
user:/config/will/be/stripped/key2              <- meta rename/cut = will/be/stripped
user:/config/will/be/stripped/key3
user:/config/will/not/be/stripped/key4
```

The result of the cut operation would be the following KeySet:

```
user:/config/key1
user:/config/key2
user:/config/stripped/key3
user:/config/will/not/be/stripped/key4
```

The cut operation is agnostic to a single trailing slash in the configuration. This means that it makes no difference whether `cut = will/be/stripped` or `cut = will/be/stripped/`. However, the cut operation refuses cut paths with leading slash. This is to clarify that key name parts can only be stripped after the parent key path.
If an invalid configuration is given or the cut operation would cause a parent key duplicate, the affected keys are simply skipped and not renamed.

86.3.2 Replace

Using the `/replacewith` global key or `rename/to` MetaKey the rename plugin will replace the part removed by `cut` with the supplied String

86.3.2.1 To upper/lower

Using the `/toupper` or `/tolower` global configuration key, or the `rename/toupper` or `rename/tolower` metakey the rename plugin will convert the keynames to uppercase or lowercase. The supplied values tell the plugin how many levels starting from the right will be converted. `toupper` and `tolower` can be combined. When no value or "0" is supplied with the keys the whole name below the parentkey will be converted.
The toupper/tolower conversions are applied after cut/replace.
Note that the names refer to the representation as KeySet. For example, if you have a configuration file where every name is uppercase:

```
KEY=value
OTHER/KEY=otherval
```

you can use `tolower=0` to get the keys `key` and `other/key`.

86.3.2.2 Examples

```
sudo kdb mount caseconversion.ini user:/tests/rename mini rename toupper=1,tolower=3
kdb set user:/tests/rename/MIXED/CASE/conversion 1
kdb ls user:/tests/rename
#> user:/tests/rename/mixed/case/CONVERSION
# Undo modifications
kdb rm -r user:/tests/rename
sudo kdb umount user:/tests/rename
sudo kdb mount renameTest.ini user:/tests/rename mini rename get/case=toupper,set/case=keyname,/cut=REMOVED
printf 'removed/key=test' > `kdb file user:/tests/rename`
kdb ls user:/tests/rename
#> user:/tests/rename/KEY
kdb set user:/tests/rename ""
cat "`kdb file user:/tests/rename`" | tail -n1
#> KEY=test
# Undo modifications
kdb rm -r user:/tests/rename
sudo kdb umount user:/tests/rename
# If you always want the keys in the configuration file upper case,
# but for your application lower case you would use:
sudo kdb mount caseconversion.ini user:/tests/rename mini rename get/case=tolower,set/case=toupper
kdb set user:/tests/rename/section/key value
kdb get user:/tests/rename/section/key
#> value
cat "`kdb file user:/tests/rename`"
#> SECTION/KEY=value
# Undo modifications
kdb rm -r user:/tests/rename
sudo kdb umount user:/tests/rename
```

86.4 Planned Operations

Additional rename operations are planned for future versions of the rename plugin:

- trim: remove spaces in the name (that are not part of parentKey)
- ranges for toupper, tolower
- allow one to specify the case in configuration files (#485)

Chapter 87

Plugin: resolver

- infos = All information you want to know is in keys below
- infos/author = Markus Raab elektra@markus-raab.org
- infos/licence = BSD
- infos/provides = resolver
- infos/needs =
- infos/placements = rollback getresolver setresolver commit
- infos/status = productive maintained specific unittest tested libc nodep configurable
- infos/description = system independent resolver

87.1 Introduction

The `@PLUGIN_SHORT_NAME@` handles operating system dependent tasks. One task is the resolving of the filenames for user and system (hence its name).
Use following command to see to which file is resolved to:

```
kdb file <Elektra path you are interested in>
```

See the constants of this plugin for further information, they are:

```
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/ELEKTRA_VARIANT_SYSTEM
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/ELEKTRA_VARIANT_USER
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/KDB_DB_HOME
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/KDB_DB_SYSTEM
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/KDB_DB_USER
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/KDB_DB_SPEC
system:/elektra/modules/@PLUGIN_SHORT_NAME@/constants/KDB_DB_DIR
```

The built-in resolving considers following cases:

- for spec with absolute path: path
- for spec with relative path: `KDB_DB_SPEC` + path
- for dir with absolute path: `pwd` + path (or above when path is found)
- for dir with relative path: `pwd` + `KDB_DB_DIR` + path (or above when path is found)
- for user with absolute path: ~ + path
- for user with relative path: ~ + `KDB_DB_USER` + path
- for system with absolute path: path
- for system with relative path: `KDB_DB_SYSTEM` + path

(~ and `pwd` are resolved from system)

87.2 Example

For an absolute path `/example.ini`, you might get following values:

- for spec: `/example.ini`
- for dir: `$PWD/example.ini`
- for user: `~/example.ini`
- for system: `/example.ini`

For an relative path example.ini, you might get following values:

- for spec: `/usr/share/elektra/specification/example.ini`
- for dir: `$PWD/.dir/example.ini`
- for user: `~/.config/example.ini`
- for system: `/etc/kdb/example.ini`

See the mount tutorial for more examples.

87.3 Variants

Many variants exist that additionally influence the resolving process, typically by using environment variables.
Environment variables are very simple for one-time usage but their maintenance in start-up scripts is problematic.
Additionally, they are controlled by the user, so they cannot be trusted. So it is not recommended to use environment variables.
Note that the file permissions apply, so it might be possible for non-root to modify keys in `system`.
See COMPILE.md for a documentation of possible variants.

87.4 Installation

See installation. The default variant of this plugin `resolver_fm_hpu_b` is part of the `libelektra5` package.
All other variants are part of the `libelektra5-extra` package.

87.4.1 XDG Compatibility

The resolver is fully `XDG compatible` if configured with the variant:

- `xp`, `xh` or `xu` for user (either using `passwd`, `HOME` or `USER` as fallback or any combination of these fallbacks)
- `x` for system, no fallback necessary

Additionally `KDB_DB_USER` needs to be left unchanged as `.config`.
`XDG_CONFIG_DIRS` will be used to resolve system paths the following way:

- if unset or empty `/etc/xdg` will be used instead
- all elements are searched in order of importance
- if a file was found, the search process is stopped
- if no file was found, the least important element will be used for potential write attempts.

87.5 Reading Configuration

1. If no update needed (unchanged modification time): quit successfully
2. Otherwise call (storage) plugin(s) to read configuration
3. remember the last stat time (last update)

87.6 Writing Configuration

1. On unchanged configuration: quit successfully
2. On empty configuration: remove the configuration file and quit successfully
3. Otherwise, open the configuration file If not available recursively create directories and retry.

#ifdef ELEKTRA_LOCK_MUTEX

1. Try to lock a global mutex, if not possible -> conflict

#endif
#ifdef ELEKTRA_LOCK_FILE

1. Try to lock the configuration file, if not possible -> conflict

#endif

1. Check the update time -> might lead to conflict
2. Update the update time (in order to not self-conflict)

We have an optimistic approach. Locking is only used to detect concurrent cooperative processes in the short moment between prepare and commit. A conflict will be raised in that situation. When processes do not lock the file it might be overwritten. This is, however, very unlikely on file systems with nanosecond precision.

87.7 Exported Functions and Data

The resolver provides 2 functions for other plugins to use.

87.7.1 filename

resolves `path` in with namespace `namespace`, creates a temporary file to work on and returns a struct containing the resolved data.
Signature: `ElektraResolved * filename (elektraNamespace namespace, const char * path, ElektraResolveTempfile tempFile, Key * warningsKey)`
`ElektraResolved` and `ElektraResolveTempfile` are both defined in `shared.h`.
`ElektraResolved` is a struct with the following members:

- `relPath`: contains the path relative to the namespace.
- `dirname`: contains the parent directory of the resolved file.
- `fullPath`: contains the full path of the resolved file.
- `tmpFile`: contains the full path of the created temporary file.

`ElektraResolveTempfile` dictates if and where a temporary file should be created. Possible values:

- `ELEKTRA_RESOLVER_TEMPFILE_NONE`: don't create a temporary file.
- `ELEKTRA_RESOLVER_TEMPFILE_SAMEDIR`: create a temporary file in the same directory as the resolved file.
- `ELEKTRA_RESOLVER_TEMPFILE_TMPDIR`: create a temporary file in `/tmp`.

87.7.2 freeHandle

frees the handle returned by `filename`.
Signature: `void * freeHandle (ElektraResolved * handle)`
where `handle` is the handle returned by `filename`.

87.8 Limitations

- If none of the resolving techniques work, the resolver will fail during `kdbOpen`. This happens, for example, with the default resolver (ELEKTRA_VARIANT_USER `hpu`) if neither: `$HOME`, `$USER`, nor any home directory in `/etc/passwd` is set.
- Conflicts with removed files are not handled.
- Links are not handled.
- uid/gid from files are not restored.

Chapter 88

Plugin: rgbcolor

- infos = Information about the rgbcolor plugin is in keys below
- infos/author = Philipp Gackstatter philipp.gackstatter@student.tuwien.ac.at
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/ordering = type
- infos/placements = postgetstorage presetstorage
- infos/status = maintained unittest nodep
- infos/metadata = check/rgbcolor
- infos/description = Validation and normalization of rgbcolors

88.1 Introduction

This plugin validates hex-formatted rgb color strings and normalizes them to decimal rgba format. It also accepts `named colors` and normalizes them.

88.2 Usage

Add the metakey `check/rgbcolor` with an arbitrary value (e.g. `""`) to the key that you want to check and normalize.

88.3 Installation

See installation. The package is called `libelektra5-extra`.

88.4 Examples

```
# Mount a config file with the rgbcolor plugin
sudo kdb mount color.ecf user:/tests/color dump rgbcolor
# Suceeds, since the value is a valid rgbcolor. Quotes are important!
kdb set user:/tests/color/hex "#a1C2b3"
# Tell the plugin to validate the key and normalize if necessary
kdb meta-set user:/tests/color/hex check/rgbcolor ""
# Colors are normalized to 32-bit unsigned integers
# This one is normalized to 0xa1C2b3ff
kdb get user:/tests/color/hex
```

```
#> 2713891839
# Color names are supported (https://www.w3.org/TR/css-color-3/#svg-color)
kdb set user:/tests/color/hex "yellowgreen"
# yellowgreen is 0x9acd32ff
kdb get user:/tests/color/hex
#> 2597139199
kdb set user:/tests/color/hex/subcolor "#abc"
kdb meta-set user:/tests/color/hex/subcolor check/rgbcolor ""
# Expanded to rgba: #aabbccff
kdb get user:/tests/color/hex/subcolor
#> 2864434431
kdb set user:/tests/color/hex/subcolor "#abcd"
# Expanded to rgba: #aabbccdd
kdb get user:/tests/color/hex/subcolor
#> 2864434397
kdb set user:/tests/color/hex/subcolor "#aabbcc"
# Expanded to rgba: #aabbccff
kdb get user:/tests/color/hex/subcolor
#> 2864434431
# Try to set incorrect value
kdb set user:/tests/color/hex fff
# RET: 5
# Try to set incorrect value
kdb set user:/tests/color/hex/subcolor "not a named color"
# RET: 5
# Try to set incorrect value
kdb set user:/tests/color/hex "fff"
# RET: 5
# Try to set incorrect value
kdb set user:/tests/color/hex "#12345"
# RET: 5
# Old values are still there
kdb get user:/tests/color/hex
#> 2597139199
# Undo modifications to the key database
kdb rm -r user:/tests/color
sudo kdb umount user:/tests/color
```

Chapter 89

Plugin: ruby

- infos = Information about RUBY plugin is in keys below
- infos/author = Bernhard Denner bernhard.denner@gmail.com
- infos/licence = BSD
- infos/needs =
- infos/provides = storage
- infos/placements = getstorage setstorage
- infos/recommends =
- infos/status = maintained reviewed unittest tested configurable global preview memleak experimental
- infos/description = Ruby plugin bridge, implement Elektra plugins with Ruby

89.1 Introduction

This plugin is an Elektra to Ruby bridge and makes is possible to implement Elektra plugins with Ruby. This plugin requires the `ruby` binding.

89.2 Installation

See installation. The package is called `libelektra5-ruby`.

89.3 Configuration Options

The plugin has the following configuration options:

- `script`: path to the Ruby plugin to use (mandatory)

89.4 Usage

Mount a configuration file using this plugin, which specifying the concrete Ruby Elektra plugin:

```
kdb mount /.ssh/authorized_keys user:/ssh/authorized_keys ruby script=<path to elektra
source>/src/plugins/ruby/ruby/ssh_authorized_keys.rb
kdb ls user:/ssh/authorized_keys
```

89.5 Implementing Ruby Plugins

The following example illustrates how to implement a Elektra plugin with Ruby:

```
require 'kdb'
# call the static function 'define' to define a new Plugin. This
# function expects one argument and a block, which implements the plugin
Kdb::Plugin.define :somename do
  # this is automatically defined
  #@plugin_name = "somename"
  # 'Open' method, called during Elektra plugin initialization
  # parameter:
  #  - errorKey: Kdb::Key, add error information if required
  # returns:
  #  - nil or 0: no error
  #  - -1      : error during initialization
  def open(errorKey)
    # perform plugin initialization
    # if an error occurs it is save to simply throw an exception. This has the same
    # semantic as returning -1
  end
  # the close method is currently not supported, since this will crash the
  # Ruby-VM if this method should be called during the VM.finalization !!!
  #def close(errorKey)
  #
  #end
  # 'check_conf' method, called before this plugin is used for mounting to check
  # if the supplied config is valid. Missing or invalid config settings can be
  # 'corrected' or added.
  # Parameter:
  #  - errorKey: Kdb::Key, to add error information
  #  - config: Kdb::KeySet, holds all plugin configuration setting
  # returns:
  #  - nil or 1 : success and the plugin configuration was NOT changed
  #  -        0 : the plugin configuration was changed and is now OK
  #  -       -1 : the configuration is NOT OK
  def check_conf(errorKey, config)
  end
  # 'get' method, is called during a get cycle, to query all keys
  #
  # Parameter:
  #  - returned : Kdb::KeySet, already holding keys to be manipulated or to be
  #               filled by a storage plugin. All added keys have to have the same
  #               key name prefix as given by parentKey.
  #  - parentKey: Kdb::Key, defining the current Elektra path. The value of this key
  #               holds the configuration file name to read from
  # Returns:
  #  - nil or 1 : on success
  #  -        0 : OK but nothing was to do
  #  -       -1 : failure
  def get(returned, parentKey)
  end
  # 'set' method, is called when writing a configuration file
  #
  # Parameter:
  #  - returned : Kdb::KeySet, holding all keys which should be written to the
  #               config file.
  #  - parentKey: Kdb::Key, defining the current Elektra path. The value of this key
  #               holds the configuration file name to write to
  # Returns:
  #  - nil or 1 : on success
  #           0 : on success with no changed keys in database
  #          -1 : failure
  def set(returned, parentKey)
  end
  # 'error' method, is called when some plugin had a problem during a write cycle
  #
  # Parameter:
  #  - returned : Kdb::KeySet, holding all keys which should be written to the
  #               config file.
  #  - parentKey: Kdb::Key, defining the current Elektra path. The value of this key
  #               holds the configuration file name to write to
  # Returns:
  #  - nil or 1 : on success
  #           0 : on success with no action
  #          -1 : failure
  def error(returned, parentKey)
  end
end
```

89.6 Known Issues

- The plugin might cause a SIGSEGV crash on application shutdown. This is a result of missing RubyVM de initialization, which is silently ignored as we can't determine if the Ruby VM is still needed or not (libelektra

might be used by an Ruby application).

- The plugin does not work correctly with Ruby version managers like `rbenv` or `rvm` and is intended to be used with the Ruby ecosystem installed in system context.

Chapter 90

Plugin: shell

- infos = Information about the shell plugin is in keys below
- infos/author = Thomas Waser `thomas.waser@libelektra.org`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/placements = postgetstorage postcommit postrollback
- infos/status = nodep configurable preview unfinished
- infos/description = executes shell commands

90.1 Usage

The shell plugin executes shell commandos after set, get or error.
The configuration keys

- `execute/set`
- `execute/get`
- `execute/error`

are used to store the shell commands.
The configuration keys

- `execute/set/return`
- `execute/get/return`
- `execute/error/return`

can be compared against the return values of the shell commandos.

90.2 Example

```
# Create temporary file
kdb set system:/tests/tempfile $(mktemp)
# Mount plugin and specify plugin configuration
sudo kdb mount shell.ini system:/tests/shell ni array= shell execute/set="echo set » $(kdb get
       system:/tests/tempfile)"
cat $(kdb get system:/tests/tempfile)
#>
# Execute `set` command
kdb set system:/tests/shell ""
#> Create a new key system:/tests/shell with string ""
cat $(kdb get system:/tests/tempfile)
```

```
#> set
# Undo modifications
rm $(kdb get system:/tests/tempfile)
kdb rm -r system:/tests/shell
sudo kdb umount system:/tests/shell
```

Chapter 91

Plugin: simpleini

- infos = Information about simpleini plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = storage/properties
- infos/needs = code binary
- infos/placements = getstorage setstorage
- infos/status = maintained unittest nodep concept obsolete 3000
- infos/description = Very simple storage plugin which stores data in a basic properties file format

91.1 Introduction

This plugin reads and writes files written in a basic line-oriented ini-like format. It is very simplistic without sections, the toml plugin and for specifications the ni plugin should be preferred. Since the `simpleini` plugin requires the GNU C library it **will not work** on operating systems that use another C library such as macOS.

91.2 Usage

It is quite suitable to export configuration if you want line-by-line key, value pairs without sections or metadata. (Thus +3000 in status)

```
kdb export system:/samba simpleini
```

91.3 Configuration

The only parameter simpleini supports is `format` which allows you to change the syntax of individual lines. The `format` is a string with any characters where only `%` has special meaning:

- `%` in an even number N are escaped and represent N/2 `%`, e.g. `%%%%` are actually `%%` in the resulting format.
- The first unescaped `%` represents the key.
- The second unescaped `%` represents the value.

The default is `% = %`.
For example, if you want every key to be marked `%:key value` you would use:

```
kdb export -c "format=%%:% %" system:/samba simpleini
#> %:key value
#> %:key2 value2
```

91.4 Restrictions

- Lines in a different format (e.g. comments) are discarded.
- The order per line must be key and then value: the plugin cannot be used if the value is first
- Whitespace before and after keynames are trimmed (but not for value)
- Delimiting symbols cannot be part of the key.
- The last occurrence of the same key wins (others are discarded).
- The parent Key cannot be used.
- This plugin needs the code and binary plugins. A code plugin is used for the escape character for some symbols (but does not respect user-defined `format`) and the binary plugin is used to handle binary values.

91.5 Examples

Mount the plugin:

```
kdb mount -d /etc/samba/smb.conf system:/samba ccode simpleini
```

91.6 Limitations

- will be excluded in macOS
- cannot parse entries where key or value is missing

Chapter 92

Plugin: spec

- infos = Information about the spec plugin is in keys below
- infos/author = Tomislav Makar tmakar23@gmail.com
- infos/licence = BSD
- infos/needs =
- infos/provides = check apply
- infos/placements = hook
- infos/status = recommended productive unittest nodep global
- infos/description = allows to give specifications for keys

92.1 Introduction

The spec plugin is a global plugin that copies metadata from the `spec`-namespace to other namespaces using their key names as globbing expressions. Globbing resembles regular expressions. They do not have the same expressive power, but are easier to use. The semantics are more suitable to match path names:

- $*$ matches any key name of just one hierarchy. This means it complies with any character except slash or null.
- `#` matches Elektra array elements. (e.g. `#0`, `#_10`, `#__987`).
- _ matches anything that $*$ matches, except array elements.

The plugin copies the metadata of the corresponding `spec` key to every matching (see below) key in the other namespaces. The copied metadata is removed again during `kdbSet` (if remained unchanged).
The spec plugin also provides basic validation and structural checking. Specifically it supports:

- detection of invalid array key names
- detection of missing keys
- validation of array ranges (min/max array size)
- validating the number of subkeys of _

92.2 Matching Algorithm

The matching of the spec (globbing) keys to the keys in the other namespaces is based on `elektraKeyGlob()`, which in turn is based on the well known `fnmatch(3)`. However, there is special handling for array specifications (`#`) and wildcard specifications (_).

92.2.1 Default Values

If a spec key has the metakey `default` set and the key does not exist in other namespaces, we create a key in the `default:/` namespace. This key has the `default` value as its value. We also copy over metadata as always.

92.2.2 Array Specifications

Keys which contain a part that is exactly `#` (e.g. `my/#/key` or `my/#`) are called array specifications. These keys are instantiated in order to support `default` values. If the key does not exist and `default` is specified in the spec namespace, the key is created in the `default` namespace. We also lookup the array size (defined by the `array` metakey) using a cascading `ksLookup`. This only looks at non-spec namespaces, if we don't find an array size there, we check the array parent in the spec namespace. If we still have no array size, the array is deemed to be empty. For empty arrays, we will simply validate that they are indeed empty.

92.2.3 Wildcard Specifications

Keys which contain a part that is exactly _ (e.g. `my/_/key` or `my/_`) are called wildcard specifications. It would be nice to have an "instantiation" procedure for _ similar to the one for arrays. However, this is not possible with the current implementation, since there is no way of knowing in advance, which keys matching the globbing expression may be requested via `ksLookup`.
Instead _ is simply treated like $*$ during matching. Afterwards we check that no array elements were matched.

92.3 Specification and Validation

The basic functionality of the plugin is to just copy (using `keyCopyMeta()` so we don't waste memory) the metadata of spec keys to all matching (as described above) keys in other namespaces. This ensures that other plugins can do their work as expected, without manually setting metadata on every key. If a metakey on a target key already exists with different value, it gets overridden.
In addition to the basic functionality, the plugin does some validation itself.

92.3.1 Required Keys

If a spec key has the metakey `require` (the value of this metakey is irrelevant and ignored), we ensure that this key exists in at least one other namespace, i.e. it can be found using a cascading `ksLookup`. If the key cannot be found, this causes an `error`.

92.3.2 Array Size Validation

As hinted to above, we validate array sizes. If a spec key `x/#` is given, and the spec key `x` has the metakey `array/min` or `array/max` set, we validate the array size (given as metakey `array` on `x`) is within the limits of `array/min` and `array/max`. Both `array/min` and `array/max` have to be valid array-elements similar to `array`. If the array size is out of bounds, this causes an `error` for `kdbSet` or a warning for `kdbGet`.
Note: We don't actually validate that the array doesn't contain elements above the given array size. This is because it doesn't have anything to do with the specification, whether the array contains additional elements. Note also that we only copy metadata onto elements within the bounds of the array size.

92.3.3 Array Specification and Wildcard Specification (Collision)

A collision is when two specification keys exist, one as wildcard specification, the other as array specification, and it is not clear in this case what the correct specification is.

```
spec:/server/_/name => meta:/description = value1
spec:/server/#/name => meta:/description = value2
```

The spec plugin does not know what specification to take in this case, so it appends an `error` or `warning` (if kdbGet).

92.4 Error Handling

In case there is an error or warning, it is appended to the `parent key`.

92.4.1 Example

```
parentKey = "system:/sw/org"
# in case there is an error
system:/sw/org/error/...
# in case there is a warning
system:/sw/org/warning/...
```

If there is an error, the `spec` plugin returns `ELEKTRA_PLUGIN_STATUS_ERROR`, otherwise `ELEKTRA_PLUGIN_STATUS_SUCCESS`.

92.4.2 Cases

- Key is required but does not exist
  - In this case an `error` is appended to the `parent key`
- Key has default but does not exist
  - In this case the key is created

92.5 Examples (with shell_recorder)

This sample is creating keys and specifications for an application named `webserver`. The `webserver` application has a `name` and a `port`. In case a port is already in use, there is also an array key `alternative_ports` which is used to find another port for binding `webserver`.
A specification could look like this (`yaml`):

```
- elektra:
    mountpoint: user:/tests/sw/org/webserver
    keys:
      name:
        value: web1
        meta:
          require: true
      port:
        value: 5000
        meta:
          default: 5000
      alternative_ports:
        keys:
          0:
            value: 5001
            meta:
              description: This is an alternative port if any other is already bound
          1:
            value: 5002
            meta:
              description: This is an alternative port if any other is already bound
        meta:
          array: 2
```

NOTE: 0 and 1 should be #0 and #1, the array elements.

Below is an explanation of each command.

92.5.1 kdb meta-set spec:/tests/sw/org/webserver/name require true

Adding a specification key with a require metakey. This `meta-set` command should throw error as no key with the name `/tests/sw/org/webserver/name` exists.

92.5.2 kdb set user:/tests/sw/org/webserver/name web1

Set the value for `user:/tests/sw/org/webserver/name` to `web1`.

92.5.3 kdb meta-set spec:/tests/sw/org/webserver/port default 5000

Adding a specification key with a default metakey. After this `meta-set` command a `default:/tests/sw/org/webserver/port` with value `5000` should exist.

92.5.4 kdb set user:/tests/sw/org/webserver/alternative_ports/#0 5001

Adding an array key `user:/tests/sw/org/webserver/alternative_ports/#0` which value is set to `5001`.

92.5.5 kdb set user:/tests/sw/org/webserver/alternative_ports/#1 5002

Adding an array key `user:/tests/sw/org/webserver/alternative_ports/#1` which value is set to `5002`.

92.5.6 kdb meta-set user:/tests/sw/org/webserver/alternative_ports array '2'

Adding a metakey `array` with value `2` at `user:/tests/sw/org/webserver/alternative_ports`.

92.5.7 kdb meta-set spec:/tests/sw/org/webserver/alternative_ports/# description 'This is an alternative port if any other is already bound'

Adding a specification metakey `description`. After this `meta-set` the all the array entries (`#0` and `#1`) should contain the `description`.

92.5.8 kdb meta-get user:/tests/sw/org/webserver/alternative_ports/#0 description

Check if the `description` metakey was copied successfully.

92.5.9 kdb meta-get user:/tests/sw/org/webserver/alternative_ports/#1 description

Check if the `description` metakey was copied successfully.

```
kdb meta-set spec:/tests/sw/org/webserver/name require true
# RET: 5
kdb set user:/tests/sw/org/webserver/name web1
# RET: 0
kdb meta-set spec:/tests/sw/org/webserver/port default 5000
# RET: 0
kdb set user:/tests/sw/org/webserver/alternative_ports/#0 5001
#> Create a new key user:/tests/sw/org/webserver/alternative_ports/#0 with string "5001"
kdb set user:/tests/sw/org/webserver/alternative_ports/#1 5002
#> Create a new key user:/tests/sw/org/webserver/alternative_ports/#1 with string "5002"
kdb meta-set user:/tests/sw/org/webserver/alternative_ports array '2'
# RET: 0
kdb meta-set spec:/tests/sw/org/webserver/alternative_ports/# description 'This is an alternative port if
       any other is already bound'
# RET: 0
kdb meta-get user:/tests/sw/org/webserver/alternative_ports/#0 description
# STDOUT-REGEX: This is an alternative port if any other is already bound
kdb meta-get user:/tests/sw/org/webserver/alternative_ports/#1 description
# STDOUT-REGEX: This is an alternative port if any other is already bound
```

92.5.10 Known limitations

- # and _ keys do not work on MINGW
- No defaults for _ globbing character

Chapter 93

Plugin: specload

- infos = Information about the specload plugin is in keys below
- infos/author = Klemens Böswirth k.boeswirth+git@gmail.com
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/specload
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest nodep libc configurable unfinished
- infos/metadata =
- infos/description = loads a specification from an external application

93.1 Introduction

In order to optimally use Elektra an application has to provide a specification. This specification has to be mounted in the `spec` namespace by the user (or automatically during the install process). However, this specification should in most cases not be modified by the user. If it is modified the user has to be sure it is still compatible to the original specification. Otherwise the application might crash because it e.g. expected a default value to be provided by the specification that the user changed or removed.
This is were this plugin comes in. Unlike other storage plugins `specload` doesn't use the file given during mounting for most of the configuration. Instead `specload` requests the specification with which an application was compiled from the application itself. On top of this base specifications the user can make some modifications, which are the only keys stored in the mounted file. All modifications made by the user are verified by the plugin. The user can never modify the specification in a way to would break application compatibility.
NOTE: currently the modifications a user can make are very limited/not possible. See `Limitations` below.

93.2 Dependencies

The plugin relies heavily on the `quickdump` plugin. It is used for storing the overridden values as well as the data transfer between `specload` and an application.
To check whether `quickdump` is available you can use:

```
kdb plugin-list quickdump
#> quickdump
```

93.3 Usage

To mount specload use (note: only the spec namespace is supported):

```
# specload will call '/usr/bin/exampleapp --elektra-spec'
kdb mount specload.eqd spec:/tests/specload/example specload 'app=/usr/bin/exampleapp'
```

You always have to specify the app that shall be called. This application will be called with the single argument --elektra-spec. This can be configured by using app/args/# during mounting:

```
# specload will call '/usr/bin/exampleapp -o spec'
kdb mount specload.eqd spec:/tests/specload/example specload 'app=/usr/bin/exampleapp' 'app/args=#1'
      'app/args/#0=-o' 'app/args/#1=spec'
```

To inhibit the default --elektra-spec argument and call an application without any arguments use "app/args=".
The app must only output the expected base specification to stdout and then exit, when called by specload.
We use the quickdump plugin for communication, so the application should call elektraQuickdumpSet.
Because this dependency may change in future, it is recommended you use the function elektraSpecload↩
SendSpec exported as "system:/elektra/modules/specload/exports/sendspec":

```
Key * errorKey = keyNew ("/", KEY_END);
// add 'system:/sendspec' key to suppress checking the 'app' key in elektraSpecloadOpen
KeySet * specloadConf = ksNew (1, keyNew ("system:/sendspec", KEY_END), KS_END);
ElektraInvokeHandle * specload = elektraInvokeOpen ("specload", specloadConf, errorKey);
int result = elektraInvoke2Args (specload, "sendspec", ks, NULL);
elektraInvokeClose (specload, errorKey);
keyDel (errorKey);
ksDel (specloadConf);
```

The code above will automatically print the KeySet ks to stdout in exactly the way specload expects it.
Once the mounting is done you can inspect you specification like you would with any other mounted configuration. If you call kdb set (or kdb meta-set or anything else that calls kdbSet()), however, specload will verify that the modifications you made are compatible with the original specification. Currently this verification is very restrictive and doesn't allow a lot of changes that would be safe. This is because the necessary verification becomes very complex very quickly. For example adding opt/arg is only safe, if opt was also added by the user, because the application might rely on the default opt/arg=none. See also Limitations.

93.3.1 Direct File Mode

Instead of loading the specification via stdin/stdout from another app, you can also instruct specload to directly load a quickdump file. This can be done by setting the file config key instead of app. The provided path must be absolute, same as an app path.
Note: If file is specified, app will be ignored.

93.4 Examples

This assumes you compiled the file testapp.c and it is available as the executable testapp in the current folder.

```
sudo kdb mount -R noresolver specload.eqd spec:/tests/specload specload "app=$(pwd)/testapp"
kdb meta-ls spec:/tests/specload/mykey
#> default
kdb get /tests/specload/mykey
#> 7
sudo kdb umount spec:/tests/specload
```

Or in direct file mode:

```
# This assumes that `$PWD` is the root of the Elektra source tree.
sudo kdb mount -R noresolver specload.eqd spec:/tests/specload specload
      "file=$(pwd)/src/plugins/specload/specload/spec.quickdump"
kdb meta-ls spec:/tests/specload/mykey
#> default
kdb get /tests/specload/mykey
#> 7
sudo kdb umount spec:/tests/specload
```

93.5 Limitations

- The plugin only allows the following modifications right now:
  - add/edit/remove description or opt/help
  - add/edit default
  - add type

- The plugin must be mounted using `noresolver` or another resolver that always calls the storage plugin. To work around this limitation, `specload` prefixes relative paths with `KDB_DB_SPEC` before passing the path on to `quickdump`. This means that your overlay files will be stored in the same directory as any other `spec` configurations.

Chapter 94

Plugin: sync

- infos = Information about the sync plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = sync
- infos/needs =
- infos/placements = precommit
- infos/status = recommended productive maintained tested nodep libc final
- infos/description = Makes sure that config file is written to disc

94.1 Introduction

Storage plugins usually do not use fsync (2) for writing their file to disc. That means that the content of the file could be in some buffer and not on the hard disc.
This is good for power saving on laptops, but not a good idea on production use otherwise. So it is strongly recommended to always add this plugin.

94.2 Usage

Just add this plugin to the list of plugins during mounting, e.g.

```
kdb mount file.dump /important dump sync
```

Then when you observe the change of a value, e.g.

```
strace kdb set user:/important/key value
```

you can see, done by storage:

```
open("/home/markus/.kdb/file.dump.16874:1409592592.95084.tmp",
        O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
write(4, "kdbOpen 1\n", 10)             = 10
write(4, "ksNew 1\n", 8)                = 8
write(4, "keyNew 19 6\n", 12)           = 12
write(4, "user:/important/key\0value\0\n", 26) = 26
write(4, "keyEnd\n", 7)                 = 7
write(4, "ksEnd\n", 6)                  = 6
close(4)                                = 0
```

then done by sync:

```
open("/home/markus/.kdb/file.dump.16874:1409592592.95084.tmp",
        O_RDWR) = 4
fsync(4)                                = 0
close(4)                                = 0
```

and finally commit + sync of directory by resolver:

```
rename("/home/markus/.kdb/file.dump.16874:1409592592.95084.tmp",
        "/home/markus/.kdb/file.dump") = 0
stat("/home/markus/.kdb/file.dump", {st_mode=S_IFREG|0644,
        st_size=69, ...}) = 0
fcntl(3, F_SETLK, {type=F_UNLCK, whence=SEEK_SET, start=0,
        len=0}) = 0
close(3)                                = 0
```

```
open("/home/markus/.kdb",
        O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
fsync(3)                                = 0
```

Chapter 95

Plugin: syslog

- infos = Information about the syslog plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = logging
- infos/needs =
- infos/placements = hook pregetstorage postcommit postrollback
- infos/status = maintained tested nodep global nodoc
- infos/description = Logs set and error calls to syslog.

95.1 Introduction

This plugin is a logging plugin which adds a log entry to syslog on commit and rollback of the configuration. Configure the plugin with `log/get=1` to enable logging when configuration is loaded. For example, `kdb gmount syslog log/get=1`.

95.2 Installation

See installation. The package is called `libelektra5-extra`.

Chapter 96

Plugin: template

- infos = Information about the template plugin is in keys below
- infos/author = Author Name elektra@libelektra.org
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = prerollback rollback postrollback getresolver pregetstorage getstorage procgetstorage postgetstorage setresolver presetstorage setstorage precommit commit postcommit
- infos/status = recommended productive maintained reviewed conformant compatible coverage specific unittest shelltest tested nodep libc configurable final preview memleak experimental difficult unfinished old nodoc concept orphan obsolete discouraged -1000000
- infos/metadata =
- infos/description = one-line description of template

96.1 Introduction

Copy this template if you want to start a new plugin written in C.

96.2 Installation

See installation. The package is called `libelektra5-experimental`.

96.3 Usage

You can use `scripts/copy-template` to automatically rename everything to your plugin name:

```
cd src/plugins
../../scripts/copy-template yourplugin
```

Then update the `README.md` of your newly created plugin:

- enter your full name+email in `infos/author`
- make sure `status`, `placements`, and other clauses conform to descriptions in `doc/CONTRACT.ini`
- update the one-line description above
- write an introduction what your plugin does
- add your plugin in `src/plugins/README.md`
- and rewrite the rest of this `README.md` to give a great explanation of what your plugin does

96.4 Plugin Configuration

None.

96.5 Dependencies

None.

96.6 Examples

```
# Backup-and-Restore: user:/tests/template
kdb set user:/tests/template/key value
#> Create a new key user:/tests/template/key with string "value"
kdb get /tests/template/key
#> value
```

96.7 Limitations

None.

Chapter 97

Plugin: timeofday

- infos = Information about the timeofday plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = tracing
- infos/needs =
- infos/placements = pregetstorage postgetstorage presetstorage precommit postcommit prerollback postrollback
- infos/status = maintained tested nodep configurable global
- infos/description = Prints timestamps during execution of backend

97.1 Introduction

This plugin is a logging plugin which prints a timestamp during all placements of backend.

97.2 Installation

See installation. The package is called `libelektra5-extra`.

97.3 Usage

If you want to measure how long your storage plugin needs to do the read and write you mount the plugin using:

```
kdb mount file.ysp user:/trace_point your_storage_plugin timeofday
```

Then you can benchmark your storage plugin in the get path:

```
kdb get user:/benchmark
#> get     0000000356      di      0000000356      pos     pregetstorage
#> get     0000000530      di      0000000174      pos     postgetstorage
#> hello
```

and in the set path:

```
kdb set user:/benchmark
#> get     0000000342      di      0000000342      pos     pregetstorage
#> get     0000000532      di      0000000190      pos     postgetstorage
#> Set null value
#> set     0000000766      di      0000000234      pos     presetstorage
#> set     0000001002      di      0000000236      pos     precommit
#> set     0000008944      di      0000007942      pos     postcommit
```

The first digit column shows the complete time passed, the second column shows the time from invocation to invocation.

97.4 Module Loading

Will not log when loaded as module (config `/module` present), unless `/logmodule` is set:

```
kdb plugin-check -c "logmodule=" timeofday
```

Chapter 98

Plugin: toml

- infos = Information about the toml plugin is in keys below
- infos/author = Jakob Fischer jakobfischer93@gmail.com
- infos/licence = BSD
- infos/provides = storage/toml
- infos/needs = base64
- infos/recommends = type
- infos/placements = getstorage setstorage
- infos/status = unittest experimental unfinished
- infos/metadata = order comment/# comment/#/start comment/#/space type tomltype origvalue
- infos/description = This storage plugin reads and writes TOML files using Flex and Bison.

98.1 Introduction

This plugin is a storage plugin for reading and writing TOML files. The plugin retains most of the file structure of a read TOML file such as comments, empty lines and TOML tables. It supports all kinds of TOML specific types and tables, including nested inline tables and multiline strings.

98.2 Requirements

The plugin needs Flex (>=2.6.2) and Bison (>=3) for parsing TOML files.

98.3 Types

98.3.1 Reading

On reading, the plugin will set the `type` metakey for strings, integers, floats and boolean values, if applicable. For decimal integers, the metakey is set to `long_long`. For binary/octal/hexadecimal integers, the metakey is set to `unsigned_long_long`. For floats, the metakey will be set to `double`. These types are chosen to conform with the TOML format as stated on the offical TOML page.
On reading, non-decimal integers will get converted to decimal. The non-decimal representation will be stored in the `origvalue` metakey of the key. When writing a key, the value of that metakey will be written instead, in order to retain the original format. Note that the `origvalue` metakey gets removed if the value of the key changes.

```
sudo kdb mount test.toml user:/tests/storage/types toml type
# Create a TOML file with 4 keys
mkdir -p $(dirname $(kdb file user:/tests/storage/types))
echo 'plain_decimal = 1000' > `kdb file user:/tests/storage/types`
echo 'file_permissions = 0o777' > `kdb file user:/tests/storage/types`
echo 'pi = 3.1415' > `kdb file user:/tests/storage/types`
```

```
echo 'division_gone_wrong = -inf' » `kdb file user:/tests/storage/types`
# Print the content of the toml file
cat `kdb file user:/tests/storage/types`
# > plain_decimal = 1000
# > file_permissions = 0o777
# > pi = 3.1415
# > division_gone_wrong = -inf
# Print types and values of the keys with `kdb`
kdb meta-get 'user:/tests/storage/types/plain_decimal' 'type'
# > long_long
kdb get 'user:/tests/storage/types/plain_decimal'
# > 1000
kdb meta-get 'user:/tests/storage/types/file_permissions' 'type'
# > unsigned_long_long
# The octal value will be converted to decimal
kdb get 'user:/tests/storage/types/file_permissions'
# > 511
kdb meta-get 'user:/tests/storage/types/pi' 'type'
# > double
kdb get 'user:/tests/storage/types/pi'
# > 3.1415
kdb meta-get 'user:/tests/storage/types/division_gone_wrong' 'type'
# > double
kdb get 'user:/tests/storage/types/division_gone_wrong'
# > -inf
# Cleanup
kdb rm -r user:/tests/storage/types
sudo kdb umount user:/tests/storage/types
```

98.3.2 Writing

On writing, for most values, the plugin will infer the appropriate type and will write them accordingly. This means, values, that match a TOML float, integer or date will be written without any quotes around them. If a value does not match any of these types, it will be written as a string.
The plugin uses an existing `type` metakey only to check if it should write a value as a `string` or a `boolean`. With this functionality, numbers can be written as a string to the file, if wanted.
To write a boolean value, the `type` metakey must be set to `boolean` for the key. Otherwise, no conversion to the TOML boolean values will take place. Per default, Elektra uses 0/1 to represent boolean values.

```
sudo kdb mount test.toml user:/tests/storage/types toml type
# Create a key, which may be a integer, boolean or string
kdb set 'user:/tests/storage/types/value' '1'
# The plugin infers `long_long` for this value
kdb meta-get 'user:/tests/storage/types/value' 'type'
# > long_long
# The value is written as an integer
cat `kdb file user:/tests/storage/types`
# > value = 1
# Manually set the `type` metakey to boolean
kdb meta-set 'user:/tests/storage/types/value' 'type' 'boolean'
# The value is written as a boolean
cat `kdb file user:/tests/storage/types`
# > value = true
# Manually set the `type` metakey to string
kdb meta-set 'user:/tests/storage/types/value' 'type' 'string'
# The value is written as a string
cat `kdb file user:/tests/storage/types`
# > value = "1"
# Cleanup
kdb rm -r user:/tests/storage/types
sudo kdb umount user:/tests/storage/types
```

98.4 Numbers

The plugin supports reading and writing of any kind of number supported by the TOML format, such as floating point numbers and binary/octal/decimal/hexadecimal integers. To write a non-decimal integer, add the corresponding prefix to the number (`0b` for binary, `0o` for octal, `0x` for hexadecimal). The value will be written in the given base to the file, but converted to decimal within Elektra (see `Reading`). Note that the plugin doesn't warn about invalid prefix/digit combinations. If the combination is not valid, it will be written as a string instead.
If the `type` plugin is enabled and you want to change the value of an existing number key which needs conversion (all keys which have a `origvalue`), you have to change the value of `origvalue` instead of the key value. Otherwise the `type` plugin will give an error.

```
# Mount a new TOML file
sudo kdb mount test.toml user:/tests/storage/numbers toml type
# Write an octal value
kdb set 'user:/tests/storage/numbers/a' '0o777'
# Get the converted decimal value
```

```
kdb get 'user:/tests/storage/numbers/a'
# > 511
# Get the original octal value
kdb meta-get 'user:/tests/storage/numbers/a' 'origvalue'
# > 0o777
# Get the type of the number; since it's originally octal, we get `unsigned_long_long`
kdb meta-get 'user:/tests/storage/numbers/a' 'type'
# > unsigned_long_long
# Change the value by changing the `origvalue` metakey
kdb meta-set 'user:/tests/storage/numbers/a' 'origvalue' '0o666'
# Get the new value as decimal
kdb get 'user:/tests/storage/numbers/a'
# > 438
# Change the value to an invalid octal value.
kdb meta-set 'user:/tests/storage/numbers/a' 'origvalue' '0o888'
# The key value is no longer considered a number
kdb meta-get 'user:/tests/storage/numbers/a' 'type'
# > string
# Cleanup
kdb rm -r user:/tests/storage/numbers
sudo kdb umount user:/tests/storage/numbers
```

98.5 Strings

The plugin can read any kind of TOML string: basic, literal, basic multiline and literal multiline.
Similar to the TOML-specific values (table, array) discussed below, we use the `tomltype` metakey to store the kind of string. The values used are: `string_basic`, `string_literal`, `string_ml_basic` and `string_ml_literal`.
When writing a file, the value of `tomltype` will be respected. The plugin will therefore preserve the kind of string throughout a kdbGet/kdbSet cycle. You can also set `tomltype` manually to transform the string into a different type. If `tomltype` is set to a kind of string that is incompatible with the current value of the key, the plugin will report an error.
If `tomltype` is not set, the plugin defaults to a basic string, or if there are at least two `\n` to a basic multiline string. This way a `kdb set` will always work correctly, as long as the given string is valid UTF-8.

```
# Mount TOML file
sudo kdb mount test_strings.toml user:/tests/storage toml type
# setting a string containing a newline escape sequence
kdb set 'user:/tests/storage/string' 'I am a basic string\not a literal one.'
kdb get 'user:/tests/storage/string'
# > I am a basic string
# > ot a literal one
# setting the string again, but escape the backslash with another backslash
kdb set 'user:/tests/storage/string' 'I am a basic string\\not a literal one.'
kdb get 'user:/tests/storage/string'
# > I am a basic string\not a literal one
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

The plugin supports all kinds of escape sequences used by TOML in basic and basic multiline strings, like `\n`, `\r`, `\t` and even `\u`/`\U` for Unicode escape sequences. When reading a file, all these escape sequences will be translated into valid UTF-8 byte sequences that will be used as the key values.
When writing a file, the plugin uses the escape sequences, whenever a part of the key value cannot be placed literally into the file. If a key contains a non-UTF-8 string value, the plugin replaces any invalid sections with the Replacement Character U+FFFD and reports a warning.
Finally, the plugin also uses `origvalue` to remember the original formatting of the string. This is particularly useful for escape sequences. Without this mechanism, a `\u` escape sequence would always be written literally (not as an escape sequence), because when reading the file it is translated into UTF-8 bytes. For multiline strings `origvalue` also includes any leading or escaped newlines.

98.6 Binary/NULL values

The plugin handles binary values by using the base64 plugin. As a result, binary values get written as base64 encoded strings, which start with the special prefix `@BASE64`. `NULL` key values are written as special strings of value `@NULL`.

```
# Mount TOML file
sudo kdb mount test_binary.toml user:/tests/storage toml type
# Creating a key with a NULL value
kdb set 'user:/tests/storage/nullkey'
# > Create a new key user:/test/nullkey with null value
# Print file content
```

```
cat `kdb file user:/tests/storage`
# > nullkey = '@NULL'
# Write base64 encoded data to the file
echo "base64 = '@BASE64SSBhbSBiYXN1IDY0IGVuY29kZWQgZm9yIG5vIHJlYXNvbi4='" > `kdb file user:/test`
# Print the value of the key, which is a binary value
kdb get 'user:/test/base64'
#>
       \x49\x20\x61\x6d\x20\x62\x61\x73\x65\x20\x36\x34\x20\x65\x6e\x63\x6f\x64\x65\x64\x20\x66\x6f\x72\x20\x6e\x6f\x20\x72\x65\x
# Print the value again, but apply the escape codes
echo -e `kdb get 'user:/test/base64'`
#> I am base 64 encoded for no reason.
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.7 TOML specific structures

TOML specific structures are represented by the metakey `tomltype` on a certain key. It will be set when the TOML plugin reads a TOML structure from a file. Additionally, this metakey can be set by the user, if they want a certain TOML structure to be written. No automatic inference of this metakey is done on writing.

98.7.1 Simple Tables

TOML's simple tables are represented by setting the `tomltype` metakey to `simpletable`.

```
# Mount TOML file
sudo kdb mount test_table.toml user:/tests/storage toml type
# Create three keys, which are all a subkey of 'common',
# but we have no 'common' simple table key yet
kdb set 'user:/tests/storage/common/a' '0'
kdb set 'user:/tests/storage/common/b' '1'
kdb set 'user:/tests/storage/common/c' '2'
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> common.a = 0
#> common.b = 1
#> common.c = 2
# Create a simple table key
kdb meta-set 'user:/tests/storage/common' 'tomltype' 'simpletable'
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> [common]
#> a = 0
#> b = 1
#> c = 2
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.7.2 Table Arrays

Table arrays are represented by setting the `tomltype` metakey to `tablearray`. It is not required to also set the array metakey, since the plugin will set the metakey, if it is missing.

```
# Mount TOML file
sudo kdb mount test_table_array.toml user:/tests/storage toml type
# Create a table array containing two entries, each with a key 'a' and 'b'
kdb meta-set 'user:/tests/storage/tablearray' 'tomltype' 'tablearray'
kdb set 'user:/tests/storage/tablearray/#0/a' '1'
kdb set 'user:/tests/storage/tablearray/#0/b' '2'
kdb set 'user:/tests/storage/tablearray/#1/a' '3'
kdb set 'user:/tests/storage/tablearray/#1/b' '4'
# Print the highest index of the table array
kdb meta-get 'user:/tests/storage/tablearray' 'array'
#> #1
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> [[tablearray]]
#> a = 1
#> b = 2
#> [[tablearray]]
#> a = 3
#> b = 4
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.7.3 Inline Tables

Inline tables are represented by setting the `tomltype` metakey to `inlinetable`. The plugin also supports reading/writing nested inline tables.

```
# Mount TOML file
sudo kdb mount test_inline_table.toml user:/tests/storage toml type
# Create a table array containing two entries, each with a key 'a' and 'b'
kdb meta-set 'user:/tests/storage/inlinetable' 'tomltype' 'inlinetable'
kdb set 'user:/tests/storage/inlinetable/a' '1'
kdb set 'user:/tests/storage/inlinetable/b' '2'
kdb meta-set 'user:/tests/storage/inlinetable/nested' 'tomltype' 'inlinetable'
kdb set 'user:/tests/storage/inlinetable/nested/x' '3'
kdb set 'user:/tests/storage/inlinetable/nested/y' '4'
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> inlinetable = { a = 1, b = 2, nested = { x = 3, y = 4 } }
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.7.4 Arrays

Arrays are recognized by the `array` metakey. On writing, the plugin will detect arrays automatically and set the appropriate metakey if it is missing.

```
# Mount TOML file
sudo kdb mount test_array.toml user:/tests/storage toml type
# Create array elements
kdb set 'user:/tests/storage/array/#0' '1'
kdb set 'user:/tests/storage/array/#1' '2'
kdb set 'user:/tests/storage/array/#2' '3'
# Print the highest index of the array
kdb meta-get 'user:/tests/storage/array' 'array'
#> #2
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> array = [1, 2, 3]
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.8 Comments and Empty Lines

The plugin preserves all comments with only one limitation for arrays. The whitespace in front of a comment is also saved.

Comments can also be created by assigning meta keys to a key. The meta keys must be of the form `comment/#n`, where `n` is a positive number, indicating the position of the comment relative to the key. An index of 0 is always the inline comment of the key. Indices greater than zero are for comments preceding the given key, where 1 is the top-most comment and the highest index comment is right above the key.

Preceding whitespace can be added to a comment by creating a `comment/#n/space` metakey. This metakey can contain any number of space or tab characters, which will be placed before the `#` starting the comment. Any whitespace *after* the `#` is considered part of the comment and will therefore be part of `comment/#n`.

File ending comments must be assigned to the file root key.

Empty lines in front of a key can be created by adding an empty `comment/#n/start` entry to it. In this case, no `comment/#n` key is needed.

```
# Mount TOML file
sudo kdb mount test_comments.toml user:/tests/storage toml type
# create a key-value pair, ready for comment decoration
kdb set 'user:/tests/storage/key' '1'
# add an inline comment with 4 leading spaces
kdb meta-set 'user:/tests/storage/key' 'comment/#0' ' This value is very interesting'
kdb meta-set 'user:/tests/storage/key' 'comment/#0/space' '    '
# add some comments preceding the key
kdb meta-set 'user:/tests/storage/key' 'comment/#1' ' I am the top-most comment relative to my key.'
kdb meta-set 'user:/tests/storage/key' 'comment/#2' ' I am in the middle. Just boring.'
kdb meta-set 'user:/tests/storage/key' 'comment/#3' ' I am in the line right above my key.'
# add file ending comments and empty lines
kdb meta-set 'user:/tests/storage' 'comment/#1' ' First file-ending comment'
kdb meta-set 'user:/tests/storage' 'comment/#2/start' ''
kdb meta-set 'user:/tests/storage' 'comment/#3' ' Second file-ending comment. I am the last line of the
      file.'
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> # I am the top-most comment relative to my key.
#> # I am in the middle. Just boring.
#> # I am in the line right above my key.
```

```
#> key = 1    # This value is very interesting
#> # First file-ending comment
#>
#> # Second file-ending comment. I am the last line of the file.
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.8.1 Comments in Arrays

Any amount of comments can be placed between array elements or between the first element and the opening brackets.
However, only one comment - an inline comment - can be placed after the last element and the closing brackets. On reading, the plugin discards any non-inline comments between the last element and the closing brackets.

```
# Mount TOML file
sudo kdb mount test_array_comments.toml user:/tests/storage toml type
# Create array elements
kdb set 'user:/tests/storage/array/#0' '1'
kdb set 'user:/tests/storage/array/#1' '2'
kdb set 'user:/tests/storage/array/#2' '3'
# Add inline comment after the array
kdb meta-set 'user:/tests/storage/array' 'comment/#0' ' Inline comment after the array'
kdb meta-set 'user:/tests/storage/array' 'comment/#0/start' '#'
kdb meta-set 'user:/tests/storage/array' 'comment/#0/space' '     '
# Add comments for array elements
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#0' ' Inline comment of first element'
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#0/start' '#'
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#0/space' '    '
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#1' ' Comment preceding the first element'
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#1/space' '    '
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#2' ' Another comment preceding the first element'
kdb meta-set 'user:/tests/storage/array/#0' 'comment/#2/space' '      '
kdb meta-set 'user:/tests/storage/array/#1' 'comment/#0' ' Inline comment of second element'
kdb meta-set 'user:/tests/storage/array/#1' 'comment/#0/space' '    '
kdb meta-set 'user:/tests/storage/array/#1' 'comment/#1' ' Comment preceding the second element'
kdb meta-set 'user:/tests/storage/array/#1' 'comment/#1/space' '      '
kdb meta-set 'user:/tests/storage/array/#2' 'comment/#0' ' Inline comment of the last element'
kdb meta-set 'user:/tests/storage/array/#2' 'comment/#0/space' '     '
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> array = [    # Comment preceding the first element
#>        # Another comment preceding the first element
#> 1,     # Inline comment of first element
#>        # Comment preceding the second element
#> 2,     # Inline comment of second element
#> 3      # Inline comment of the last element
#> ]      # Inline comment after the array
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

98.9 Order

The plugin preserves the file order by the usage of the metakey `order`. When reading a file, the order metakey will be set according to the order as read in the file. If new keys are added, eg. via `kdb set`, the order of the set key will be set to the next-to-highest order value present in the existing key set.
However, the order is only relevant between elements with the same TOML-parent. For example keys of a simple table are only sorted with respect to each other, not with any keys outside that table. If that table has it's order changed and moves to another position in the file, so will it's subkeys.
When sorting elements under the same TOML-parent, tables (simple and array) will always be sorted after non-table elements, regardless of their order. With this limitation, we prevent that a newly set key, that is not part of a certain table array/simple table, would be placed after the table declaration, making it a member of that table on a subsequent read.

```
# Mount TOML file
sudo kdb mount test_order.toml user:/tests/storage toml type
# Create three keys in reverse alphabetical order under the subkey common
# Additionally, create one key not in the common subkey space
kdb set 'user:/tests/storage/common/c' '0'
kdb set 'user:/tests/storage/common/b' '1'
kdb set 'user:/tests/storage/common/a' '2'
kdb set 'user:/tests/storage/d' '3'
# Print the content of the resulting TOML file
# The keys are ordered as they were set
cat `kdb file user:/tests/storage`
#> common.c = 0
#> common.b = 1
```

```
#> common.a = 2
#> d = 3
# Create a simple table for the three keys under `common`
kdb meta-set 'user:/tests/storage/common' 'tomltype' 'simpletable'
# Print the content of the resulting TOML file
cat `kdb file user:/tests/storage`
#> d = 3
#> [common]
#> c = 0
#> b = 1
#> a = 2
# Cleanup
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

In this example, `d` and `common` have the same parent, the file root. This means, they need to be sorted with each other. `d` would be placed before `common` by it's order, since it was set before, and thus, has lesser order. However, their order never gets compared, since `common` is a simple table and `d` is not, so `d` will get sorted before the table regardless of order.

98.10 Limitations

While the plugin has good capabilities in handling the TOML file format, it currently lacks some features possible with Elektra:

- Sparse arrays are not preserved on writing, they get a continuous array without index holes.
- Values on non-leaf keys are currently not supported, they get discarded. This applies especially to the parent key of the mountpoint.
- Custom metakeys cannot be written by the plugin, so they get discarded.

Additionally, there are some minor limitations related to the TOML file format, mostly related to the preservation of the original file structure:

- On writing, each string is written as a TOML basic string (single or multiline). See `Strings`
- Comments and newlines between the last array element and closing brackets are discarded.
- Trailing commas in arrays and inline tables are discarded
- Only spaces in front of comments are preserved.

Chapter 99

Plugin: tracer

- infos = Information about the tracer plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = tracing
- infos/needs =
- infos/placements = pregetstorage procgetstorage postgetstorage presetstorage precommit postcommit prerollback postrollback
- infos/status = maintained tested nodep configurable global
- infos/description = Traces the execution path of a backend

99.1 Introduction

This plugin is added on every possible position within a backend. It allows you to trace when the backend is executed.

99.2 Installation

See installation. The package is called `libelektra5-extra`.

99.3 Usage

If you want to trace how and if the backend is called:

```
kdb mount file.ysp user:/trace_point your_storage_plugin tracer
```

So now we can trace whats below your trace point.

```
kdb ls user:/trace_point
```

Ok, no tracer is called because resolver immediately told that there is no file.

```
kdb get user:/trace_point
#> Did not find key
```

Ok, same conclusion.

```
kdb set user:/trace_point hello
#> create a new key user:/trace_point with string "hello"
#> tracer: set(0xd34cc0, user:/trace_point): user:/trace_point 1
#> tracer: set(0xd34cc0, user:/trace_point): user:/trace_point 1
#> tracer: set(0xd34cc0, user:/trace_point): user:/trace_point 1
```

Now the 3 placements in set are called.

```
kdb get user:/trace_point
#> tracer: get(0x22e1cc0, user:/trace_point): 0
#> tracer: get(0x22e1cc0, user:/trace_point): 0
#> hello
```

Now the 2 placements in get are called.

99.4 Module Loading

Will not log when loaded as module (config `/module` present), unless `/logmodule` is set:

```
kdb plugin-check -c "logmodule=" tracer
```

Chapter 100

Plugin: type

- infos = Information about type plugin is in keys below
- infos/author = Klemens Böswirth `k.boeswirth+git@gmail.com`
- infos/licence = BSD
- infos/provides = check
- infos/needs =
- infos/placements = postgetstorage presetstorage
- infos/status = recommended maintained unittest tested nodep libc configurable
- infos/metadata = check/type type check/enum check/enum/# check/enum/delimiter check/boolean/true check/boolean/false
- infos/description = type checker using COBRA data types

100.1 Introduction

This plugin is a type checker plugin using the `CORBA` data types.
A common and successful type system happens to be CORBA. The system is well suited because of the many well-defined mappings it provides to other programming languages.
The type checker plugin supports these types: `short`, `unsigned_short`, `long`, `unsigned_long`, `long_long`, `unsigned_long_long`, `float`, `double`, `char`, `wchar`, `boolean`, `any`, `enum`, `string`, `wstring` and `octet`.

- Checking `any` will always be successful, regardless of the content.
- `string` matches any string key value.
- `octet` and `char` are equivalent to each other.
- `enum` will do enum checking as described below.
- `boolean` only allows the values `1` and `0`. See also `Normalization`.
- To use `wchar` and `wstring` the function `mbstowcs(3)` must be able convert the key value into a wide character string. `wstring`s can be of any non-zero length, `wchar` must have exactly length 1.

100.2 Enums

If a key is set to the type `enum` the plugin will look for the metadata array `check/enum/#`.
For example:

```
check/enum = #3
check/enum/#0 = small
check/enum/#1 = middle
```

```
check/enum/#2 = large
check/enum/#3 = huge
```

Only the values listed in this array will be accepted. The array indices don't have to be continuous, using e.g. only `#1`, `#2` and `#4` is also allowed. Just make sure `check/enum` is set to the largest index in the array.
Furthermore `check/enum/delimiter` may contain a separator character, that separates multiple allowed occurrences. If `check/enum/delimiter` contain more than a single character validation will fail.
For example:

```
check/enum/delimiter = _
```

Then the value `middle_small` would validate. `middle_small_small` would be allowed as well, because multi-values are treated like bitfields.

100.3 Normalization

Some types support normalization in addition to the standard type checking. This means that an extended set of allowed values is normalized into a different (in most cases standardized) set before type checking. The normalized values will be passed on from `kdbGet` to the rest of Elektra and your application. During `kdbSet` changed values are also normalized before type checking and at the end of `kdbSet`, if everything was successful, the values are restored to the ones originally provided by the user (no matter if they were changed before `kdbSet` or already present in `kdbGet`).
The full normalization/restore procedure can be described by the following cases:

- **Case 1:** The Key existed in kdbGet and is unchanged between `kdbGet` and `kdbSet`

  This is the easiest and most obvious case. The value is normalized in `kdbGet` and the original value is restored in `kdbSet`, so that the underlying storage file remains unchanged (w.r.t. the key in question).

- **Case 2:** The Key didn't exist in `kdbGet`, i.e. it was added

  Here the value is normalized to verify the type and then restored immediately (all inside of `kdbSet`).

- **Case 3:** The Key existed in `kdbGet`, but its value was changed between kdbGet and kdbSet

  This is essential the same as Case 2. `keySetString` removes the `origvalue` metadata used to store the original value, so as far as this plugin is concerned, the key didn't exist in `kdbGet`.

Note: If normalization is used, often times you will get a normalization error instead of a type checking error.

100.3.1 Booleans

The values that are accepted as `boolean`s are configured during mounting:

```
sudo kdb mount typetest.dump user:/tests/type dump type booleans=#1 booleans/#0/true=a booleans/#0/false=b
       booleans/#1/true=t booleans/#1/false=f
```

The above line defines that the array of allowed boolean pairs. `booleans=#1` defines the last element of the array as `#1`. For each element `#` the keys `booleans/#/true` and `booleans/#/false` define the true and false value respectively. True values are normalized to `1`, false values to `0`.
Even though we didn't specify them the values `1` and `0` are still accepted. The normalized values are always okay to use. If no configuration is given, the allowed values default to `1`, `yes`, `on`, `true`, `enabled` and `enable` as true values and `0`, `no`, `off`, `false`, `disabled` and `disable` as false values.
The accepted values can also be overridden on a per-key-basis. Simply add the metakey `check/boolean/true` and `check/boolean/false` to set the accepted true and false values. Only a single true/false value can be chosen. This is intended for use cases, where normally you prefer to use only e.g. `1`, `0` and `true`, `false`, but what to override that for a key where e.g. `enabled` and `disabled` make more sense contextually (e.g. for something like `/log/debug`). Because of this intention restoring also works differently, when `check/boolean/true` and `check/boolean/false` are used. In this case we will always restore to the chosen override values.
Note: The values `1` and `0` are accepted, even if overrides are used. This means you can set a key with overrides to `0` (or `1`) and during `kdbSet` it will be restored to the false (or true) override value. (This is useful for the high-level API.)
It is an error to specify only one of `booleans/#/true` and `booleans/#/false` or `check/boolean/true` and `check/boolean/false`. *Boolean always come in pairs!*
You can also change how values shall be restored in `kdbSet`:

```
sudo kdb mount typetest.dump user:/tests/type dump type booleans=#0 booleans/#0/true=true
       booleans/#0/false=false boolean/restoreas=#0
```

The config key boolean/restoreas must be a valid index of the booleans array or the special value none. If boolean/restoreas was set to an index, the chosen boolean pair will be used when values are restored in kdbSet. So in the above example the plugin accepts 1, true, 0 and false as boolean values, on kdbGet it turns true into 1 and false into 0 and on kdbSet it turns 1 into true and 0 into false.
If no booleans array was given the allowed values for boolean/restoreas are:

- #0 for yes/no
- #1 for true/false
- #2 for on/off
- #3 for enabled/disabled
- #4 for enable/disable

The special value boolean/restoreas=none completely disables the restore procedure. In other words, kdbSet will always return either 0 or 1 for boolean values. This is useful, if a storage format with built-in support for boolean values is used.

100.3.2 Enums

Enums also support normalization. Contrary to boolean normalization, enum normalization is always configured on a per-key-basis.
Simply set the metakey check/enum/normalize to 1 in order to normalize the string values to there indexes. Any other value is ignored.
Take for example a key with the following enum configuration:

```
check/enum = #3
check/enum/#0 = small
check/enum/#1 = medium
check/enum/#3 = huge
```

The value small will be normalized to 0, medium to 1 and huge to 3. During restore the values 0, 1 and 3 will be restored to small, medium and huge.
If you use normalization, you can pass string values or indices to kdbGet and kdbSet, but you will always get back indices from kdbGet and string values from kdbSet. (Therefore you can seamlessly use elektraGet↩ UnsignedLongLong from high-level API for normalized enums.)
The plugin also supports normalizing enums that use check/enum/delimiter, however be careful which indexes you use in this case. The indexes of all values are simple bitwise or-ed (using |). In the above example small_medium would be normalized to 1 (0 | 1 == 1), the same value as medium. This means during restore the value emitted will be medium.
A version that would work with delimiter and normalization is:

```
check/enum = #4
check/enum/#0 = none
check/enum/#1 = small
check/enum/#2 = medium
check/enum/#4 = huge
```

Here small_medium is normalized to 3, which is a unique value. During restore with delimiters the values might not be restored to there original form, but may be restored to an equivalent representation. e.g. small_none may be restored to just small or small_medium may be restored to medium_small This has technical reasons and we do not guarantee any restriction on what representation is produced during restore, other than the normalized value being the same as for the user provided representation.
IMPORTANT: Do **not** use normalization together with enums, whose string values start with digits (e.g. check/enum/#0 = 1abc). This breaks normalization! Indices are differentiated from string value by whether they start with a digit.

100.4 Example

```
#Mount the plugin
sudo kdb mount typetest.dump user:/tests/type dump type
#Store a character value
kdb set user:/tests/type/key a
#Only allow character values
kdb meta-set user:/tests/type/key type char
kdb get user:/tests/type/key
#> a
```

```
#If we store another character everything works fine
kdb set user:/tests/type/key b
kdb get user:/tests/type/key
#> b
#If we try to store a string Elektra will not change the value
kdb set user:/tests/type/key 'Not a char'
# RET:5
# ERROR:C03200
kdb get user:/tests/type/key
#> b
#Undo modifications to the database
kdb rm user:/tests/type/key
sudo kdb umount user:/tests/type
```

For enums:

```
# Backup-and-Restore:/tests/enum
sudo kdb mount typeenum.ecf user:/tests/type dump type
# valid initial value + setup valid enum list
kdb set user:/tests/type/value middle
kdb meta-set user:/tests/type/value check/enum '#2'
kdb meta-set user:/tests/type/value 'check/enum/#0' 'low'
kdb meta-set user:/tests/type/value 'check/enum/#1' 'middle'
kdb meta-set user:/tests/type/value 'check/enum/#2' 'high'
kdb meta-set user:/tests/type/value type enum
# should succeed
kdb set user:/tests/type/value low
# should fail with error C03200
kdb set user:/tests/type/value no
# RET:5
# ERROR:C03200
```

Or with multi-enums:

```
# valid initial value + setup array with valid enums
kdb set user:/tests/type/multivalue middle_small
kdb meta-set user:/tests/type/multivalue check/enum/#0 small
kdb meta-set user:/tests/type/multivalue check/enum/#1 middle
kdb meta-set user:/tests/type/multivalue check/enum/#2 large
kdb meta-set user:/tests/type/multivalue check/enum/#3 huge
kdb meta-set user:/tests/type/multivalue check/enum/delimiter _
kdb meta-set user:/tests/type/multivalue check/enum "#3"
kdb meta-set user:/tests/type/multivalue type enum
# should succeed
kdb set user:/tests/type/multivalue small_middle
# should fail with error C03200
kdb set user:/tests/type/multivalue all_small
# RET:5
# ERROR:C03200
# cleanup
kdb rm -r user:/tests/type
sudo kdb umount user:/tests/type
```

For booleans:

```
# Mount plugin
sudo kdb mount config.ecf user:/tests/type dump type
# By default the plugin uses `1` (true) and `0` (false) to represent boolean values
kdb set user:/tests/type/truthiness false
kdb meta-set user:/tests/type/truthiness type boolean
kdb get user:/tests/type/truthiness
#> 0
# The plugin does not change ordinary values
kdb set user:/tests/type/key value
kdb get user:/tests/type/key
#> value
# Undo changes
kdb rm -r user:/tests/type
sudo kdb umount user:/tests/type
sudo kdb mount config.ecf user:/tests/type dump type
kdb set user:/tests/type/truthiness 0
kdb meta-set user:/tests/type/truthiness type boolean
 kdb set user:/tests/type/truthiness yes
# RET: 0
 # Undo changes
kdb rm -r user:/tests/type
sudo kdb umount user:/tests/type
```

100.5 Limitations

Records are part of other plugins.

The `CORBA` type system also has its limits. The types `string` and `enum` can be unsatisfactory. While string is too general and makes no limit on how the sequence of characters is structured, the enumeration is too finite. For example, it is not possible to say that a string is not allowed to have a specific symbol in it. Combine this plugin with other type checker plugins to circumvent such limitations.

Chapter 101

Plugin: uname

- infos = Information about the uname plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = storage/info
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest shelltest nodep readonly limited concept
- infos/description = Includes uname information into the key database.

101.1 Introduction

This plugin is a storage plugin that will use the syscall `uname (2)`. No resolver is needed for that plugin to work.

101.2 Installation

See installation. The package is called `libelektra5-extra`.

101.3 Special Values

This plugin defines following keynames below its mount point:

- sysname
- nodename
- release
- version
- machine

101.4 Errors

The only documented error in `uname(2)` is when an invalid buffer is passed to it. As this is an implementation error only, this plugin should not run into errors.

101.5 Restrictions

This plugin is read-only.

101.6 Example

```
# To mount uname information using this plugin:
kdb mount -R noresolver none user:/tests/uname uname
# List available data
kdb ls user:/tests/uname/
#> user:/tests/uname/machine
#> user:/tests/uname/nodename
#> user:/tests/uname/release
#> user:/tests/uname/sysname
#> user:/tests/uname/version
# Read the OS name
kdb get user:/tests/uname/sysname
# STDOUT-REGEX: CYGWIN_NT.*|Darwin|DragonFly|FreeBSD|Linux|OpenBSD
# Read the OS version number
kdb get user:/tests/uname/release
# STDOUT-REGEX: [0-9]+(\.[0-9]+)*[[:alnum:][:punct:]]*
# Unmount the plugin
kdb umount user:/tests/uname
```

Chapter 102

Plugin: unit

- infos = Information about the unit plugin is in keys below
- infos/author = Marcel Hauri `e1355940@student.tuwien.ac.at`
- infos/maintainer = Florian Lindner `florian.lindner@student.tuwien.ac.at`
- infos/licence = BSD
- infos/needs =
- infos/provides =
- infos/recommends =
- infos/placements = presetstorage postgetstorage
- infos/status = maintained reviewed conformant compatible coverage specific unittest tested libc nodep final
- infos/metadata = check/unit
- infos/description = validates units of memory and normalizes to bytes

102.1 Installation

See installation. The package is called `libelektra5-extra`.

102.2 Validation options

The following representation standards of units are currently supported and can be used by setting `check/unit`:

- `<numeric value><one or more spaces><memory unit>`
  e.g. 20 MB, 256 KB

102.3 Normalization

The following representation standards of units are currently supported and can be use by setting `check/unit` to:

- `<numeric value><one or more spaces><memory unit>`
  e.g. 20 MB, 256 KB, the value then will be normalized to bytes

102.4 Examples

e.g. 20 KB will become 20000 Bytes

102.5 Limitations

Only basic units are supported (everything from Bytes to Petabytes, but no KiB, MiB or Gib) The max value is limited by the capacity of unsigned Longs. Using greater values will result in an error.

Chapter 103

Plugin: validation

- infos = Information about validation plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = check
- infos/needs =
- infos/placements = presetstorage
- infos/status = maintained unittest nodep libc
- infos/metadata = check/validation check/validation/message check/validation/ignorecase check/validation/match check/validation/invert check/validation/type
- infos/description = Validates key values using regular expressions

103.1 Introduction

This plugin is a check plugin which checks string values of Keys using regular expressions.

103.2 Usage

The validation plugin looks for two metakeys. `check/validation` gives a regular expression to check against. If it is present, `check/validation/message` may contain an optional humanly readable message that will be passed with the error information.
Important: To validate against the whole string, you have to start the regular expression with ^ and end it with `$`. Otherwise expressions that e.g. match the empty string, always return true. Alternatively, you can use `check/validation/match=LINE`.

103.3 Configuration

Metadata can be supplied to configure the validation:

- `check/validation/match`: You can check against `LINE`, `WORD` or `ANY`
- `check/validation/ignorecase`: If you want to ignore case.
- `check/validation/invert`: If you want to invert match.

103.4 Implementation

The implementation consists of a loop checking for every key if it has the mentioned metakey. The check itself is done by the POSIX regular expression library with the interface `regcomp`, `regexec`, `regerror` and `regfree`. The flag `REG_EXTENDED` is passed so that the regular expression will be compiled as an extended regular expression. `REG_NOSUB` gives a better performance and subexpressions cannot be used in this setup anyway.

103.5 Exported Methods

The plugin also exports the function `ksLookupRE()` that does a lookup in a KeySet using a regular expression. It starts from the current cursor of the KeySet and stops when the first value matches. Finally, this key is returned.

Chapter 104

Plugin: version

- infos = Information about the version plugin is in keys below
- infos/author = Vid Leskovar vid.leskovar5@gmail.com, Klemens Böswirth k.boeswirth+git@gmail.com
- infos/licence = BSD
- infos/needs =
- infos/provides = version
- infos/recommends =
- infos/placements = backend
- infos/status = unittest nodep
- infos/metadata =
- infos/description = Provides version information for this Elektra installation.

104.1 Introduction

This plugin is very simple. It implements the minimum functionality required to qualify as a read-only backend plugin. The data this plugin returns during `kdbGet()` is simple the data returned by `elektraVersionKeySet()` with each `Key` additionally marked with the metadata `restrict/write` and `restrict/remove`.
For version `0.9.4` of Elektra, this plugin returned this data for example:

```
system:/elektra/version = "Below are version information of the Elektra Library you are currently using"
system:/elektra/version/constants =
system:/elektra/version/constants/KDB_VERSION = "0.9.4"
system:/elektra/version/constants/KDB_VERSION_MAJOR = "0"
system:/elektra/version/constants/KDB_VERSION_MINOR = "9"
system:/elektra/version/constants/KDB_VERSION_PATCH = "4"
system:/elektra/version/constants/SO_VERSION = "5"
system:/elektra/version/infos = "All information you want to know"
system:/elektra/version/infos/author = "Markus Raab <elektra@markus-raab.org>"
system:/elektra/version/infos/licence = "BSD"
system:/elektra/version/infos/description = "Information of your Elektra Installation"
system:/elektra/version/infos/version = "1"
```

Chapter 105

Plugin: wresolver

- infos = Information about the wresolver plugin is in keys below
- infos/author = Markus Raab elektra@libelektra.org
- infos/licence = BSD
- infos/provides = resolver
- infos/needs =
- infos/placements = rollback getresolver setresolver
- infos/status = recommended maintained nodep unfinished nodoc
- infos/description = Returns success on every call and can be used as resolver.

105.1 Introduction

Resolver for non-POSIX, e.g. w32/w64 systems.
Uses SHGetFolderPath for w32/w64 to get the "home directory".

105.2 Limitation

- Does not remove config files on empty configuration (https://issues.libelektra.org/2531)

Chapter 106

Plugin: xerces

- infos = Information about the xerces plugin is in keys below
- infos/author = Armin Wurzinger e1528532@libelektra.org
- infos/licence = BSD
- infos/provides = storage/xml
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = recommended maintained unittest memleak
- infos/metadata = xerces/rootname
- infos/description = Storage in the XML format.

106.1 Introduction

This plugin is a storage plugin allowing Elektra to read and write XML formatted files. It uses a general format which:

- Maps key names to XML elements
- Maps key values to textual content of XML elements
- Maps metakeys to XML attributes. Metakey name = attribute name, Metakey value = attribute value
- Ignores XML comments

106.2 Installation

See installation. The package is called `libelektra5-xerces`.
To include this plugin in a homebrew installation run `brew tap elektrainitiative/elektra` followed by `brew install elektrainitiative/elektra/elektra --with-xerces`

106.3 Usage

To mount an XML file we use:

```
kdb mount file.xml user:/test/file xerces
```

The strength and usage of this plugin is that it supports arbitrary XML files and does not require a specific format. Given the following example of an XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<xerces>foo
  <bar meta="da_ta">bar</bar>
</xerces>
```

Please note that if the key name does not correspond to the root element of the xml file, the original name gets stored in a metakey called `xerces/rootname`. The content of the root element gets mapped to the mount point. We can observe the following result after mounting:

```
kdb get user:/test/file
#> foo
kdb get user:/test/file/bar
#> bar
kdb meta-get user:/test/file/bar meta
#> da_ta
```

To export an existing keyset to the XML format:

```
kdb export user:/test/xerces xerces > example.xml
```

The root element of the resulting XML file will be "xerces" again, restored via the metadata. If you don't want this behavior, delete the metadata `xerces/rootname` on the mount point, then it uses the mount point's name instead.

106.4 Dependencies

- `Xerces-C++ 3.0.0` or newer (`apt-get install libxerces-c-dev` or `brew install xerces-c` on macOS)
- CMake 3.6 or a copy of `FindXercesC.cmake` in `/usr/share/cmake-3.0/Modules/`

106.5 Limitations

This plugin is not able to handle key names which contain characters that are not allowed to appear as an XML element name. Consider using the rename plugin to take care about proper escaping.
The main rules of an XML element name are:

- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc.)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

The root key is not allowed to be an array, as this would correspond to multiple root elements in XML (see the `GitHub issue`).
XSD transformations, schemas or DTDs are not supported yet.

106.6 Examples

106.6.1 Mounting, setting a key and exporting

```
# Backup-and-Restore:user:/tests/xercesfile
sudo kdb mount xerces.xml user:/tests/xercesfile xerces
kdb set user:/tests/xercesfile foo
kdb meta-set user:/tests/xercesfile xerces/rootname xerces
kdb set user:/tests/xercesfile/bar bar
kdb meta-set user:/tests/xercesfile/bar meta "da_ta"
kdb meta-get user:/tests/xercesfile xerces/rootname
#> xerces
kdb get user:/tests/xercesfile/bar
#> bar
kdb export user:/tests/xercesfile xerces
# STDOUT-REGEX: <bar meta="da_ta">bar</bar>
sudo kdb rm -r user:/tests/xercesfile
sudo kdb umount user:/tests/xercesfile
```

Chapter 107

Plugin: xfconf

- infos = Information about the xfconf plugin is in keys below
- infos/author = Richard Stöckl e11908080@student.tuwien.ac.at
- infos/licence = BSD
- infos/needs =
- infos/provides = storage/xml
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = maintained unittest libc configurable memleak experimental limited unfinished concept
- infos/metadata =
- infos/description = storage plugin for xfconf

107.1 Introduction

The xfconf plugin is a storage plugin to mount the xfconf configuration settings. This allows to operate on the XFCE configuration with libelektra. As usual, this allows the plugin to read and write to the XFCE configuration.
For further understanding how xfconf is structured, please refer to the Xfconf binding.
The list of all channels is stored in the `system:/elektra/modules/xfconf/channels` which is an array of all channel names. Channels cannot be created manually with libelektra. Instead, when mounting a channel (see `Plugin Configuration` how to specify a channel) and setting a key value below the mount, it will be automatically created.
You may also refer to the `official XFCE documentation of xfconf` to learn more about it.

107.2 Dependencies

The xfconf library from the XFCE project is the main dependency of this plugin. Usually, this library is called something such as `xfconf` (Arch, Fedora, `xfconf-devel` for compiling), `libxfconf-0` (Debian, `libxfconf-0-dev` for compmiling) or `xfce4-conf` (FreeBSD) in the package manager. As xfconf itself depends on dbus and glib, these are dependecies too but should be installed with the package manager automatically.
This plugin is tested with the xfconf versions `4.16` and above.
Before you can start working with the plugin, you have to make sure dbus is running. If you use a common desktop environment this should already be fulfilled. Otherwise, `export $(dbus-launch)` can be used to start it. You may also need to use `systemd-machine-id-setup` to create `/etc/machine-id`.
Note that this plugin does not support macOS since Xfce is not commonly used there, Mac Ports only provides a very old version (`4.12`) and Brew does not provide Xfce at all.

107.3 Usage

The usage is identical to most storage plugins except that the channel option during mount must be defined in order to tell xfconf which channel to mount.

107.4 Plugin Configuration

The required `channel` configuration option is used to tell xfconf which channel to mount.

107.5 Examples

```
# Backup-and-Restore: user:/tests/xfconf
# mount the xfwm channel
kdb mount -R noresolver none /test/xfwm xfconf "channel=xfwm4"
# store old button layout
set "OLD_LAYOUT=$(kdb get /test/xfwm/general/button_layout)"
# set only a close button
kdb set system:/test/xfwm/general/button_layout "C|"
# read the new layout
kdb get /test/xfwm/general/button_layout
#> C|
# restore old layout
kdb set /test/xfwm/general/button_layout "$OLD_LAYOUT"
# umount the channel
kdb umount /test/xfwm
```

107.6 Locks

Xfconf uses so-called locks to prevent users or applications to set properties to a new value. They can be referred as a type of constant. As of now, locks cannot be set or unset in xfconf which makes locked properties read-only forever.

107.7 Limitations

- xfconf locks can only be read but not set as this is not possible in xfconf
- comments and sorting are not implemented due the lack of both in xfconf
- due to memory leaks in xfconf upstream, valgrind reports errors when running the tests

Chapter 108

Plugin: xmltool

- infos = Information about xmltool plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = storage/xml
- infos/needs =
- infos/placements = getstorage setstorage
- infos/status = unittest final memleak unfinished old discouraged
- infos/description = Storage using legacy XML format.

108.1 Introduction

This plugin is a storage plugin allowing Elektra to read and write XML formatted files. It uses the legacy Elektra 0.7 XML format.
This plugin can be used for migration of Key Databases from 0.7 -> 0.8. It should not be used otherwise.

108.2 Installation

See installation. The package is called `libelektra5-xmltool`.

108.3 Dependencies

- `libxml2-dev`

108.4 Restrictions

- only supports metadata as defined in Elektra 0.7
- null and empty values are not distinguished
- exported relative to first key found, not to parent key (ksGetCommonParentName)
- error messages vague (no difference between error opening file and validation errors)

108.5 Examples

After you have upgraded Elektra, you can import XML files from Elektra 0.7:

```
kdb import system:/ xmltool < system.xml
kdb import user:/ xmltool < user.xml
```

Or you can also mount an XML file using `xmltool` (not recommended):

```
kdb mount /etc/example.xml system:/example xmltool
```

Chapter 109

Plugin: yajl

- infos = Information about YAJL plugin is in keys below
- infos/author = Markus Raab `elektra@libelektra.org`
- infos/licence = BSD
- infos/provides = storage/json
- infos/needs = directoryvalue type
- infos/recommends = rebase comment
- infos/placements = getstorage setstorage
- infos/status = maintained coverage unittest
- infos/description = JSON using YAJL

109.1 Introduction

This is a plugin reading and writing JSON files using the library `yail`
The plugin was tested with yajl version 1.0.8-1 from Debian 6 and yajl version 2.0.4-2 from Debian 7.
Examples of files which are used for testing can be found below the folder in "src/plugins/yajl/yajl".
The JSON grammar can be found `here`.
A validator can be found `here`.
Supports every KeySet except when arrays are intermixed with other keys. Has only limited support for metadata.

109.2 Installation

See installation. The package is called `libelektra5-yajl`.

109.3 Dependencies

- `libyajl-dev` (version 1 and 2 should work)

109.4 Types

The type of the data is available via the metadata `type`:

- `string`: The JSON string type.
- `boolean`: The JSON boolean type (true or false)
- `double`: For JSON numbers.

If no metadata `type` is given, the type is either:

- `null` on binary null-key
- `string` otherwise

Any other type/value will still be treated as string, but the warning `C03200` will be added because of the potential data loss.

109.5 Special values

In JSON it is possible to have empty arrays and objects. In Elektra this is mapped using the special names
`###empty_array`
and
`___empty_map`
Arrays are mapped to Elektra's array convention #0, #1,..

109.6 Restrictions

- Only UTF-8 is supported. Use the `iconv` plugin if your locale are not UTF-8. When using non-UTF-8 the plugin will be able to write the file, but cannot parse it back again. You will error C03100, invalid bytes in UTF8 string.
- Everything is string if not tagged by metakey "type" Only valid JSON types can be used in type, otherwise there are some fall backs to string but warnings are produced.
- Arrays will be normalized (to #0, #1, ..)
- Comments of various JSON-dialects are discarded.
- Mixing of arrays and maps is not detected and leads to corrupted JSON files. Please specify arrays to avoid such situations.
- The plugin creates adds an empty root key to the database, even if you did not add this key (see http://issues.libelektra.org/2132).

Because of these potential problems a type checker and comments filter are highly recommended.

109.7 Usage

The following example shows you how you can read and write data using this plugin.

```
# Mount the plugin to the cascading namespace `/tests/yajl`
sudo kdb mount config.json /tests/yajl yajl
# Manually add a key-value pair to the database
printf '{ "number": 1337 }' > `kdb file user:/tests/yajl`
# Retrieve the new value
kdb get user:/tests/yajl/number
#> 1337
# Determine the data type of the value
kdb meta-get user:/tests/yajl/number type
#> double
# Add another key-value pair
kdb set user:/tests/yajl/key value
# STDOUT-REGEX: .*Create a new key user:/tests/yajl/key with string "value"
# Retrieve the new value
kdb get user:/tests/yajl/key
#> value
# Check the format of the configuration file
kdb file user:/tests/yajl/ | xargs cat
#> {
#>     "key": "value",
#>     "number": 1337
#> }
# Add an array
kdb set user:/tests/yajl/piggy/#0 straw
kdb set user:/tests/yajl/piggy/#1 sticks
kdb set user:/tests/yajl/piggy/#2 bricks
# Retrieve an array key
kdb get user:/tests/yajl/piggy/#2
```

```
#> bricks
# Check the format of the configuration file
kdb file user:/tests/yajl | xargs cat
#> {
#>     "key": "value",
#>     "number": 1337,
#>     "piggy": [
#>         "straw",
#>         "sticks",
#>         "bricks"
#>     ]
#> }
# Undo modifications to the database
kdb rm -r user:/tests/yajl
sudo kdb umount /tests/yajl
```

109.7.1 Directory Values

The YAJL plugin support values in directory keys via the Directory Value plugin.

```
# Mount the plugin to `user:/tests/yajl`
sudo kdb mount config.json user:/tests/yajl yajl
# Add two directory keys and one leaf key
kdb set user:/tests/yajl/roots 'Things Fall Apart'
kdb set user:/tests/yajl/roots/bloody 'Radical Face'
kdb set user:/tests/yajl/roots/bloody/roots 'No Roots'
# Add an array containing two elements
kdb set user:/tests/yajl/now ', Now'
# Elektra arrays require the metakey `array` to the parent.
# Otherwise the keys below `user:/tests/yajl/now` would be
# interpreted as normal key-value pairs.
kdb meta-set user:/tests/yajl/now array ''
kdb set user:/tests/yajl/now/#0 'Neighbors'
kdb set user:/tests/yajl/now/#1 'Threads'
kdb ls user:/tests/yajl
#> user:/tests/yajl/now
#> user:/tests/yajl/now/#0
#> user:/tests/yajl/now/#1
#> user:/tests/yajl/roots
#> user:/tests/yajl/roots/bloody
#> user:/tests/yajl/roots/bloody/roots
# Retrieve directory values
kdb get user:/tests/yajl/roots
#> Things Fall Apart
kdb get user:/tests/yajl/roots/bloody
#> Radical Face
# Retrieve leaf value
kdb get user:/tests/yajl/roots/bloody/roots
#> No Roots
# Check array
kdb get user:/tests/yajl/now
#> , Now
kdb meta-get user:/tests/yajl/now array
#> #1
kdb get user:/tests/yajl/now/#0
#> Neighbors
kdb get user:/tests/yajl/now/#1
#> Threads
# Undo modifications to the database
kdb rm -r user:/tests/yajl
sudo kdb umount user:/tests/yajl
```

109.7.2 Booleans

The YAJL plugin maps "1" and "true" to its true bool type, and "0" and "false" to its false bool type. However, it always returns 1 or 0.

You can take advantage of the type plugin to map arbitrary values to true and false.

```
# Type plugin is automatically mounted since yajl depends on it
sudo kdb mount conf.json user:/tests/yajl yajl
kdb set user:/tests/yajl 1
kdb get user:/tests/yajl
#> 1
kdb meta-set user:/tests/yajl type boolean
kdb set user:/tests/yajl on
kdb get user:/tests/yajl
#> 1
kdb set user:/tests/yajl/subkey disable
kdb meta-set user:/tests/yajl/subkey type boolean
kdb get user:/tests/yajl/subkey
#> 0
# Undo modifications to the database
kdb rm -r user:/tests/yajl
sudo kdb umount user:/tests/yajl
```

109.8 OpenICC Device Config

This plugin was specifically designed and tested for the `OpenICC_device_config_DB` although it is of course not limited to it.

Mount the plugin:

```
kdb mount --resolver=resolver_fm_xhp_x color/settings/openicc-devices.json \
  /org/freedesktop/openicc yajl rename cut=org/freedesktop/openicc
```

or:

```
kdb mount-openicc
```

Then you can copy the `OpenICC_device_config_DB.json` to systemwide or user config, e.g.

```
cp src/plugins/yajl/examples/OpenICC_device_config_DB.json /etc/xdg
cp src/plugins/yajl/examples/OpenICC_device_config_DB.json ~/.config
kdb ls system:/org/freedesktop/openicc
```

prints out then all device entries available in the config

```
kdb get system:/org/freedesktop/openicc/device/camera/0/EXIF_manufacturer
```

prints out "Glasshuette" with the example config in source

You can export the whole system openicc config to ini with:

```
kdb export system:/org/freedesktop/openicc simpleini > dump.ini
```

or import it:

```
kdb import system:/org/freedesktop/openicc ini < dump.ini
```

Chapter 110

Plugin: yamlcpp

- infos = Information about the yamlcpp plugin is in keys below
- infos/author = René Schwaiger `sanssecours@me.com`
- infos/licence = BSD
- infos/needs = base64 directoryvalue
- infos/provides = storage/yaml
- infos/recommends =
- infos/placements = getstorage setstorage
- infos/status = unittest preview unfinished concept discouraged
- infos/metadata =
- infos/description = This storage plugin reads and writes data in the YAML format

110.1 YAML CPP

110.1.1 Introduction

The YAML CPP plugin reads and writes configuration data via the `yaml-cpp` library.

110.1.2 Installation

See installation. The package is called `libelektra5-yamlcpp`.

110.1.3 Usage

You can mount this plugin via `kdb mount`:
```
sudo kdb mount config.yaml /tests/yamlcpp yamlcpp
```
. To unmount the plugin use `kdb umount`:
```
sudo kdb umount /tests/yamlcpp
```
. The following examples show how you can store and retrieve data via `yamlcpp`.
```
# Mount yamlcpp plugin to cascading namespace `/tests/yamlcpp`
sudo kdb mount config.yaml /tests/yamlcpp yamlcpp
# Manually add a mapping to the database
echo "key : whale"               > `kdb file user:/tests/yamlcpp`
# Retrieve the value of the manually added key
kdb get user:/tests/yamlcpp/key
#> whale
# Manually add syntactically incorrect data
echo "some key: @some  value" » `kdb file user:/tests/yamlcpp`
kdb get "user:/tests/yamlcpp/some key"
# STDERR: .*yaml-cpp: error at line 2, column 11: unknown token.*
# ERROR: C03100
# RET: 5
# Overwrite incorrect data
echo "key: value" >  `kdb file user:/tests/yamlcpp`
```

```
# Add some values via `kdb set`
kdb set user:/tests/yamlcpp musical note
kdb set user:/tests/yamlcpp/fleetwood mac
kdb set user:/tests/yamlcpp/the chain
# Retrieve the new values
kdb get user:/tests/yamlcpp
#> musical note
kdb get user:/tests/yamlcpp/the
#> chain
kdb get user:/tests/yamlcpp/fleetwood
#> mac
# Undo modifications
kdb rm -r /tests/yamlcpp
sudo kdb umount /tests/yamlcpp
```

110.1.4 Arrays

YAML CPP provides support for Elektra's array data type.

```
# Mount yamlcpp plugin to `user:/tests/yamlcpp`
sudo kdb mount config.yaml user:/tests/yamlcpp yamlcpp
# Manually add an array to the database
echo 'sunny:'       >  `kdb file user:/tests/yamlcpp`
echo '  - Charlie'  » `kdb file user:/tests/yamlcpp`
echo '  - Dee'      » `kdb file user:/tests/yamlcpp`
# List the array entries
kdb ls user:/tests/yamlcpp
#> user:/tests/yamlcpp/sunny
#> user:/tests/yamlcpp/sunny/#0
#> user:/tests/yamlcpp/sunny/#1
# Read an array entry
kdb get user:/tests/yamlcpp/sunny/#1
#> Dee
# You can retrieve the last index of an array by reading the metakey `array`
kdb meta-get user:/tests/yamlcpp/sunny array
# 1
# Extend the array
kdb set user:/tests/yamlcpp/sunny/#2 Dennis
kdb set user:/tests/yamlcpp/sunny/#3 Frank
kdb set user:/tests/yamlcpp/sunny/#4 Mac
# The plugin supports empty array fields
kdb set user:/tests/yamlcpp/sunny/#_10 'The Waitress'
kdb meta-get user:/tests/yamlcpp/sunny array
#> #_10
kdb get user:/tests/yamlcpp/sunny/#_9
# RET: 11
# Retrieve the last array entry
kdb get user:/tests/yamlcpp/sunny/$(kdb meta-get user:/tests/yamlcpp/sunny array)
#> The Waitress
# The plugin also supports empty arrays (arrays without any elements)
kdb meta-set user:/tests/yamlcpp/empty array ''
kdb export user:/tests/yamlcpp/empty yamlcpp
#> []
# Arrays in Elektra always require the `array` metakey.
# Otherwise the keys will be interpreted as normal key-value mappings.
kdb set user:/tests/yamlcpp/movies ""
kdb set user:/tests/yamlcpp/movies/#0 'A Silent Voice'
kdb export user:/tests/yamlcpp/movies yamlcpp
#> "#0": A Silent Voice
kdb meta-set user:/tests/yamlcpp/movies array ''
kdb export user:/tests/yamlcpp/movies yamlcpp
#> - A Silent Voice
# Undo modifications to the key database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.4.1 Nested Arrays

The plugin also supports nested arrays.

```
# Mount yamlcpp plugin to `user:/tests/yamlcpp`
sudo kdb mount config.yaml user:/tests/yamlcpp yamlcpp
# Add some key value pairs
kdb set user:/tests/yamlcpp/key value
kdb set user:/tests/yamlcpp/array ""
kdb set user:/tests/yamlcpp/array/#0 scalar
kdb set user:/tests/yamlcpp/array/#1/key value
kdb set user:/tests/yamlcpp/array/#1/key see-no-evil monkey
kdb meta-set user:/tests/yamlcpp/array array '#1'
kdb ls user:/tests/yamlcpp
#> user:/tests/yamlcpp/array
#> user:/tests/yamlcpp/array/#0
#> user:/tests/yamlcpp/array/#1/key
#> user:/tests/yamlcpp/array/#1/key
#> user:/tests/yamlcpp/key
```

```
# Retrieve part of an array value
kdb get user:/tests/yamlcpp/array/#1/key
#> value
# Since an array saves a list of values, an array parent
# - which represent the array - does not store a value!
echo "user:/tests/yamlcpp/array: "`kdb get user:/tests/yamlcpp/array`""
#> user:/tests/yamlcpp/array: ""
# Remove part of an array value
kdb rm user:/tests/yamlcpp/array/#1/key
kdb ls user:/tests/yamlcpp
#> user:/tests/yamlcpp/array
#> user:/tests/yamlcpp/array/#0
#> user:/tests/yamlcpp/array/#1/key
#> user:/tests/yamlcpp/key
# The plugin stores array keys using YAML sequences.
# Since yaml-cpp stores keys in arbitrary order -
# either `key` or `array` could be the "first" key -
# we remove `key` before we retrieve the data. This way
# we make sure that the output below will always look
# the same.
kdb rm user:/tests/yamlcpp/key
kdb file user:/tests/yamlcpp | xargs cat
#> array:
#>   - "___dirdata: "
#>   - scalar
#>   - key: see-no-evil monkey
# Undo modifications to the key database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.4.2 Sparse Arrays

Since Elektra allows "holes" in a key set, YAML CPP has to support small key sets that describe relatively complex data.

```
# Mount yamlcpp plugin
sudo kdb mount config.yaml user:/tests/yamlcpp yamlcpp
kdb set      user:/tests/yamlcpp/#0/map/#1/#0 value
kdb set      user:/tests/yamlcpp ""
kdb meta-set user:/tests/yamlcpp          array '#0'
kdb set      user:/tests/yamlcpp/#0/map ""
kdb meta-set user:/tests/yamlcpp/#0/map    array '#1'
kdb set      user:/tests/yamlcpp/#0/map/#1 ""
kdb meta-set user:/tests/yamlcpp/#0/map/#1 array '#0'
kdb file user:/tests/yamlcpp | xargs cat
#> - "___dirdata: "
#> - map:
#>     - "___dirdata: "
#>     - ~
#>     -
#>       - "___dirdata: "
#>       - value
# The plugin adds the missing array parents to the key set
kdb ls user:/tests/yamlcpp
#> user:/tests/yamlcpp
#> user:/tests/yamlcpp/#0/map
#> user:/tests/yamlcpp/#0/map/#0
#> user:/tests/yamlcpp/#0/map/#1
#> user:/tests/yamlcpp/#0/map/#1/#0
# Undo modifications to the key database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.5 Metadata

The plugin supports metadata. The example below shows how a basic `Key` including some metadata, looks inside the YAML configuration file:

```
key without metadata: value
key with metadata: !elektra/meta
  - value2
  - metakey: metavalue
    empty metakey:
    another metakey: another metavalue
```

. As we can see above the value containing metadata is marked by the tag handle `!elektra/meta`. The data type contains a list with two elements. The first element of this list specifies the value of the key, while the second element contains a map saving the metadata for the key. The data above represents the following key set in Elektra if we mount the file directly to the namespace `user`:

Name	Value	Metaname	Metavalue
user:/key without metadata	value1	—	—

Name	Value	Metaname	Metavalue
user:/key with metadata	value2	metakey	metavalue
		empty metakey	—
		another metakey	another metavalue

. The example below shows how we can read and write metadata using the `yamlcpp` plugin via `kdb`.

```
# Mount yamlcpp plugin to 'user:/tests/yamlcpp'
sudo kdb mount config.yaml user:/tests/yamlcpp yamlcpp
# Manually add a key including metadata to the database
echo "key: !elektra/meta [unicorn, {comment: Unicorn}]" >  'kdb file user:/tests/yamlcpp'
kdb meta-ls user:/tests/yamlcpp/key
#> comment
kdb meta-get user:/tests/yamlcpp/key comment
#> Unicorn
# Add a new key and add some metadata to the new key
kdb set user:/tests/yamlcpp/brand new
kdb meta-set user:/tests/yamlcpp/brand comment "The Devil And God Are Raging Inside Me"
kdb meta-set user:/tests/yamlcpp/brand rationale "Because I Love It"
# Retrieve metadata
kdb meta-ls user:/tests/yamlcpp/brand
#> comment
#> rationale
kdb meta-get user:/tests/yamlcpp/brand rationale
#> Because I Love It
# Undo modifications to the key database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

We can also invoke additional plugins that use metadata like `type`.

```
sudo kdb mount config.yaml user:/tests/yamlcpp yamlcpp type
kdb set user:/tests/yamlcpp/typetest/number 21
kdb meta-set user:/tests/yamlcpp/typetest/number check/type short
kdb set user:/tests/yamlcpp/typetest/number "One"
# RET: 5
# STDERR: .*Validation Semantic.*
# ERROR: C03200
kdb get user:/tests/yamlcpp/typetest/number
#> 21
# Undo modifications to the key database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.6 Binary Data

YAML CPP also supports `base64` encoded data via the Base64 plugin.

```
# Mount YAML CPP plugin at 'user:/tests/binary'
sudo kdb mount test.yaml user:/tests/binary yamlcpp
# Manually add binary data
echo 'bin: !!binary aGk=' > 'kdb file user:/tests/binary'
# Base 64 decodes the data 'aGk=' to 'hi' and stores the value in binary form.
# The command 'kdb get' prints the data as hexadecimal byte values.
kdb get user:/tests/binary/bin
#> \x68\x69
# Add a string value to the database
kdb set user:/tests/binary/text mate
# Base 64 does not modify textual values
kdb get user:/tests/binary/text
#> mate
# The Base 64 plugin re-encodes binary data before YAML CPP stores the key set. Hence the
# configuration file contains the value 'aGk=' even after YAML CPP wrote a new configuration.
grep -q 'bin: !.* aGk=' 'kdb file user:/tests/binary'
# RET: 0
# Undo modifications to the database
kdb rm -r user:/tests/binary
sudo kdb umount user:/tests/binary
```

110.1.7 Empty

Sometimes you only want to save a key with an empty value. The commands below show that YAML CPP supports this scenario properly.

```
# Mount YAML CPP plugin at 'user:/tests/yamlcpp'
sudo kdb mount test.yaml user:/tests/yamlcpp yamlcpp
# Check if the plugin saves empty keys correctly
kdb set user:/tests/yamlcpp/empty ""
kdb set user:/tests/yamlcpp/empty/level1/level2 ""
kdb ls user:/tests/yamlcpp/empty
#> user:/tests/yamlcpp/empty
#> user:/tests/yamlcpp/empty/level1/level2
kdb get -v user:/tests/yamlcpp/empty | grep -vq 'The key is null.'
# Undo modifications to the database
```

```
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.8 Binary Values

Elektra saves binary data as either `0` or `1`. The YAML CPP plugin supports this design decision by converting between YAML's and Elektra's boolean type.

```
# Mount YAML CPP plugin at 'user:/tests/yamlcpp'
sudo kdb mount config.yaml user:/tests/yamlcpp yamlcpp
# Manually add boolean key
echo 'truth: true' > 'kdb file user:/tests/yamlcpp'
kdb get user:/tests/yamlcpp/truth
#> 1
# A boolean in Elektra has the type 'boolean'
kdb meta-get user:/tests/yamlcpp/truth type
#> boolean
# Add another boolean value
kdb set user:/tests/yamlcpp/success 0
kdb meta-set user:/tests/yamlcpp/success type boolean
kdb get user:/tests/yamlcpp/success
#> 0
kdb export user:/tests/yamlcpp/success yamlcpp
#> false
# Undo modifications to the database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.9 Dependencies

This plugin requires `yaml-cpp`. On a Debian based OS the package for the library is called `libyaml-cpp-dev`. On macOS you can install the package `yaml-cpp` via `HomeBrew`.

110.1.10 Limitations

110.1.10.1 Leaf Values

One of the limitations of this plugin is, that it only supports values inside `leaf nodes`. Let us look at an example to show what that means. The YAML file below:

```
root:
  subtree: fallen leaves
  below root: leaf
level 1:
  level 2:
    level 3: maple leaf
```

stores all of the values (`fallen leaves`, `leaf` and `maple leaf`) in the leaves of the mapping. The drawing below makes this situation a little bit clearer.
The key set that this plugin creates using the data above looks like this (assuming we mount the plugin to `user:/tests/yamlcpp`):

Name	Value
user:/tests/yamlcpp/level	
user:/tests/yamlcpp/level 1/level 2	
user:/tests/yamlcpp/level 1/level 2/level 3	maple leaf
user:/tests/yamlcpp/root	
user:/tests/yamlcpp/root/below root	leaf
user:/tests/yamlcpp/root/subtree	fallen leaves

. Now why is this plugin unable to store values outside leaf nodes? For example, why can we not store a value inside `user:/tests/yamlcpp/level 1/level 2`? To answer this question we need to look at the YAML representation:

```
level 1:
  level 2:
    level 3: maple leaf
```

. In a naive approach we might just try to add a value e.g. `see-no-evil monkey` right next to level 2:

```
level 1:
  level 2: see-no-evil monkey
    level 3:  maple leaf
```

. This however would be not correct, since then the YAML node `level 2` would contain both a scalar value

(`see-no-evil monkey`) and a mapping (`{ level 3: maple leaf }`). We could solve this dilemma using a list:

```
level 1:
  level 2:
    - see-no-evil monkey
    - level 3: maple leaf
```

. However, if we use this approach we are not able to support Elektra's array type properly.

110.1.10.1.1 Directory Values

To overcome the limitation described above, the YAML CPP plugin requires the Directory Value plugin. This plugin converts the value of a non-leaf node to a leaf node with the name `___dirdata`. For example, let us assume we have the following key set:

```
directory      = Directory Data
directory/file = Leaf Data
```

. The Directory Value plugin will convert the key set in the set (write) direction to

```
directory            =
directory/___dirdata = Directory Data
directory/file       = Leaf Data
```

. Consequently the YAML plugin will store the key set as

```
directory:
  ___dirdata: Directory Data
  file: Leaf Data
```

. A user of the YAML plugin will not notice this feature unless he edits the configuration file by hand, as the following example shows:

```
# Mount YAML CPP plugin at 'user:/tests/yamlcpp'
sudo kdb mount test.yaml user:/tests/yamlcpp yamlcpp
kdb set user:/tests/yamlcpp/directory 'Directory Data'
kdb meta-set user:/tests/yamlcpp/directory comment 'Directory Metadata'
kdb set user:/tests/yamlcpp/directory/file 'Leaf Data'
kdb ls user:/tests/yamlcpp/directory
#> user:/tests/yamlcpp/directory
#> user:/tests/yamlcpp/directory/file
kdb get user:/tests/yamlcpp/directory
#> Directory Data
kdb meta-get user:/tests/yamlcpp/directory comment
#> Directory Metadata
kdb get user:/tests/yamlcpp/directory/file
#> Leaf Data
# Undo modifications to the database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.10.2 Special Values

Due to the way the plugin writes data

- first converting the key set into yaml-cpp's `Node` data structure, and then
- writing this data structure into a file,

and the way the yaml-cpp library handles writing `Nodes`, the plugin does currently not handle data with special meaning according to the `YAML spec` correctly. For example, if you use the `kdb` tool to save the value `true` in a key, then the plugin will not quote this value and you will end up with a boolean value.

```
# Mount plugin
sudo kdb mount test.yaml user:/tests/yamlcpp yamlcpp
kdb set user:/tests/yamlcpp/boolean true
# The following command should print a quoted YAML scalar
# (e.g. `"true"` or `'true'`).
kdb export user:/tests/yamlcpp/boolean yamlcpp
#> true
# Since the value is not quoted the YAML CPP plugin will
# correctly convert the YAML data into one of Elektra's
# boolean values (`0` or `1`).
kdb get user:/tests/yamlcpp/boolean
#> 1
# Undo modifications to the database
kdb rm -r user:/tests/yamlcpp
sudo kdb umount user:/tests/yamlcpp
```

110.1.10.3 Other Limitations

- Adding and removing keys does remove **comments** inside the configuration file
- The plugin currently lacks proper **type support** for scalars.

- If Elektra uses YAML CPP as **default storage** plugin, multiple tests of the test suite fail. However, if you mount YAML CPP at `/`:

```
kdb mount default.yaml / yamlcpp
```

all tests should work correctly. The problem here is that Elektra does not load additional required plugins (`infos/needs`) for a default storage plugin.

Chapter 111

Plugin: zeromqrecv

- infos = Information about the zeromqrecv plugin is in keys below
- infos/author = Thomas Wahringer `waht@libelektra.org`
- infos/licence = BSD
- infos/provides = notification
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage
- infos/status = maintained unittest libc global experimental
- infos/description = Receives notifications using a ZeroMq subscriber socket

111.1 Introduction

This plugin is a notification plugin, which receives notifications using ZeroMq subscribe (`ZMQ_SUB`) sockets from the compatible zeromqsend plugin.

111.2 Installation

See installation. The package is called `libelektra5-zeromq`.

111.3 Dependencies

- `libzmq3-dev` (ZeroMQ C bindings > 3.2)

111.4 Usage

The recommended way is to globally mount the plugin together with the zeromqsend plugin:

```
kdb global-mount zeromqsend zeromqrecv
```

This plugin is designed to be used as a transport plugin for Elektra's notification feature. If notification is not enabled (i.e. in the tool `kdb` or in any other application that does not use `elektraNotifiationContract()`) this plugin does performs no operations.
Since ZeroMQ sockets only provide a 1:n mapping (i.e. one publisher with many subscribers or one subscriber and many publishers) the `zeromqsend` and `zeromqrecv` plugins require a XPUB/XSUB endpoint. The kdb tool `"hub-zeromq"` provides these endpoints.

111.5 Transport Plugin

Mount this plugin globally with default settings to use it as *receiving* transport plugin for Elektra's notification feature:

```
kdb global-mount zeromqrecv
```

111.6 Configuration

This plugin supports the following configuration options when mounting:

- **endpoint**: ZeroMQ XPUB or PUB socket to connect to. The `ipc` and `tcp` ZeroMQ transports are recommended. The default value is "tcp://localhost:6001".

111.7 Notification Format

For the notification format please see `the zeromqsend plugin documentation`.

Chapter 112

Plugin: zeromqsend

- infos = Information about the zeromqsend plugin is in keys below
- infos/author = Thomas Wahringer `waht@libelektra.org`
- infos/licence = BSD
- infos/provides = notification
- infos/needs =
- infos/recommends =
- infos/placements = postgetstorage postcommit
- infos/status = maintained unittest libc global experimental
- infos/description = Sends notifications over ZeroMq publish sockets when a key is changed

112.1 Introduction

This plugin is a notification plugin, which sends notifications using ZeroMq publish (`ZMQ_PUB`) sockets when the key database (KDB) has been modified. It is compatible with the sending zeromqrecv plugin.

112.2 Installation

See installation. The package is called `libelektra5-zeromq`.

112.3 Dependencies

- `libzmq3-dev` (ZeroMQ C bindings > 3.2)

112.4 Usage

The recommended way is to globally mount the plugin together with the zeromqrecv plugin:

```
kdb global-mount zeromqsend zeromqrecv
```

This plugin is designed to be used as a transport plugin for Elektra's notification feature. Since ZeroMq creates threads for asynchronous I/O this plugin always operates asynchronously.
Since ZeroMQ sockets only provide a 1:n mapping (i.e. one publisher with many subscribers or one subscriber and many publishers) the `zeromqsend` and `zeromqrecv` plugins require a XPUB/XSUB endpoint. The kdb tool `"hub-zeromq"` provides these endpoints.

112.5 Transport Plugin

Mount this plugin globally with default settings to use it as *sending* transport plugin for Elektra's notification feature:

```
kdb global-mount zeromqsend
```

112.6 Configuration

This plugin supports the following configuration options when mounting:

- **endpoint**: ZeroMQ XSUB or SUB socket to connect to. The `ipc` and `tcp` ZeroMQ transports are recommended. The default value is "tcp://localhost:6000".
- **connectTimeout**: Timeout for establishing connections in milliseconds. The default value is "1000".
- **subscribeTimeout**: Timeout for waiting for subscribers in milliseconds. The default value is "200".

112.7 Notification Format

The ZeroMQ transport plugins use the publish/subscribe sockets (`ZMQ_PUB` and `ZMQ_SUB`) for notification transport.
Each notification is a multipart message. The first part contains the type of change, the second part contains the name of the changed key.
Possible only current change is `Commit`.

Chapter 113

Authors

This is a list of people that contributed to the Elektra Initiative.

113.1 Markus Raab

maintainer, main development, documentation, quality

- email: `markus@libelektra.org`
- devel/test on: Debian stable

113.2 Mihael Pranjić

maintainer, development of the cache and mmapstorage plugins

- email: `mpranj@limun.org`
- GitHub user: `mpranj`
- devel/test on: Fedora, Debian, macOS

113.3 Klemens Böswirth

a bit of everything, including:

- current implementation for `libelektra-kdb`, including new concept of backend plugins
- created `libelektra-highlevel`
- extended `spec` plugin for use with `libelektra-highlevel`
- created `quickdump` and `specload` plugins
- created `kdb gen` code generator
- many, many bugfixes
- email: `k.boeswirth+git@gmail.com`
- GitHub user: `kodebach`
- devel/test on: Manjaro

113.4 Robert Sowula

packaging, release automation

- email: robert@sowula.at
- GitHub user: robaerd
- devel/test on: Debian, Ubuntu

113.5 Michael Tucek

Java bindings

- email: libelektra@hellwach.at
- GitHub user: tucek
- devel/test on: Debian (WSL2), Ubuntu (WSL2)

113.6 Manuel Mausz

SWIG bindings

- email: manuel-elektra@mausz.at
- devel/test on: Fedora

113.7 Dardan Haxhimustafa

development of KConfig plugin

- email: mail@dardan.im
- GitHub user: darddan
- devel/test on: Arch Linux

113.8 Felix Berlakovich

journald plugin, augeas plugin, keytometa

- email: elektra@berlakovich.net
- devel/test on: Debian

113.9 Kai-Uwe Behrmann

test & packaging on CentOS, Fedora, OpenSUSE, RHEL and SLE

- email: kai-uwe@behrmann.name

113.10 Charles Lindsay

ni library

113.11 Avi Alkalay

initial concept & implementation

- email: avi@Unix.sh

113.12 Patrick Sabin

trie development, previous python bindings

- email: patricksabin@gmx.at

113.13 Daniel Bugl

documentation, web ui

- email: me@omnidan.net

113.14 Kurt Micheli

optimizing ksLookup with order preserving minimal perfect hash map, Markdown link converter for doxygen

- email: e1026558@student.tuwien.ac.at

113.15 Alexander Rössler

improvements to the dbus plugin

- email: mail@roessler.systems

113.16 Peter Nirschl

development of the crypto plugin, minor bug fixes, BSD/macOS support

- email: peter.nirschl@gmail.com
- GitHub user: petermax2
- devel on: Fedora
- test on: Fedora, Debian, macOS

113.17 René Schwaiger

documentation, bug fixes

- email: sanssecours@me.com
- GitHub user: sanssecours
- devel/test on: macOS

113.18 Marvin Mall

development of the REST service & frontend for sharing of conf. snippets, cachefilter plugin, website

- email: marvin-mall@msn.com
- GitHub user: Namoshek
- devel/test on: Debian Jessie

113.19 Thomas Waser

main development, many plugins, ansible

- email: thomas.waser@libelektra.org
- GitHub user: tom-wa
- devel/test on: Debian Testing/Sid

113.20 Raffael Pancheri

design and implementation of the qt-gui

- email: raffael@libelektra.org
- GitHub user: 0003088

113.21 Bernhard Denner

ruby bindings, ruby plugin, build server, Puppet module

- email: bernhard.denner@gmail.com
- GitHub user: BernhardDenner
- devel/test on: Ubuntu and Debian

113.22 Thomas Wahringer

notification system, I/O bindings

- email: waht@libelektra.org
- GitHub user: waht
- devel/test on: Debian

113.23 Maximilian Irlinger

documentation, elektraMerge including its Python bindings, copy-on-write functionality of Key and KeySet, generic changetracking implementation

- email: max@maxirlinger.at
- GitHub user: atmaxinger
- devel/test on: Fedora, macOS

113.24 Stefan Hanreich

elektra-core, rust implementation

- email: e01227486@student.tuwien.ac.at
- GitHub user: lawli3t
- devel/test on: Arch Linux / Debian Docker

113.25 Hannes Laimer

KDB CLI rewrite C++ -> C, CLI cleanup

- email: hannes.laimer@tuwien.ac.at
- GitHub user: hannes99
- devel/test on: Arch

113.26 Tomislav Makar

spec plugin, go-elektra, elektrad, web-ui

- email: tmakar23@gmail.com
- GitHub user: tmakar
- devel/test on: macOS

113.27 Florian Lindner

mounting library, unixODBC backend

- email: florian.lindner@student.tuwien.ac.at
- GitHub user: flo91
- devel/test on: Gentoo Linux

Chapter 114

Big Picture

Elektra solves a non-trivial issue: how to abstract configuration in a way that software can be integrated and reconfiguration can be automated. Elektra solves this problem in a holistic way. Read why Elektra for an explanation of why such a solution is necessary.

114.1 Virtual File System Analogy

If you know virtual file systems, you already know a very similar solution to a very similar problem (otherwise first read about what a virtual file system is here).
Before file systems (or for devices without operating system) software simply wrote at discs (think of `dd of=/dev/sda`) to persistently store data. This obviously does not work with multiple applications. To allow multiple applications to access data in `/dev/sda`, a file system structures them in a way that every application knows where its bits are. In analogy configuration files are application-specific initialization that cannot be shared with other applications. So as first steps we need to have a "file system" that describes the content of a configuration file in a uniform way. In Elektra plugins represent file systems: they know how the data in configuration files should be interpreted.
For file systems, the API is `open`, `read`, `write`, and then `close`. For configuration key-value access is more suitable because values are so small that a single read/write always suffices. Thus Elektra has a key-value API with `kdbOpen`, `kdbGet`, `kdbSet` and `kdbClose`. Not every application is written in C, thus many `bindings` were written to access file systems. For example, in C++ you have `fstream`, and in Java `File↩ Reader`. Also Elektra provides different bindings: In C++ you have a class `KDB` and can use `kdb.get` or `kdb.↩ set`. Furthermore, every language has native support for the language's iterators which make Elektra easier to use.
Furthermore, command-line tools like `cat` and `ls` provide an additional interface to the content of file systems for users and administrators. In Elektra also command-line tools for the analog purpose exist: With `kdb cp`, `kdb mv`, and `kdb ls` some command-line tools operating on file systems are mimicked.
There is not one file system satisfying every need, e.g., `proc`, `tmpfs`, `nfs` and `ext4` have quite different use cases and are needed at the same time in parallel. In analogy, different configuration file format (parsers) have different advantages and disadvantages. For example, `/etc/passwd` or `/etc/hosts` are structured, while many other configuration files are semi-structured.
To have multiple file systems present at the same time a virtual file system is able to `mount (2)` concrete file systems and thus give applications a way to uniformly access them. Similarly, Elektra also implements a mount functionality in its core. In Elektra a `contract` specifies the obligations between plugins.
Sometimes, it is even useful to have multiple file systems at the same mount point, so-called `stacked` file systems. They allow you to combine features of different file systems. For example, `eCryptfs` allows you to encrypt directories or files. In Elektra, stacking plugins is a core feature and heavily used to avoid feature-bloated plugins. For example, the crypto plugin allows you to encrypt individual keys or the iconv plugin to change the character encoding.
In file systems metadata describes information about files, e.g. when they were last accessed and who they are owned by (`ls -l`). In the same way Elektra has metadata that describe individual key-value pairs. In Elektra metadata is `defined globally` but implemented in many plugins.
Implementations of file systems is not an easy task. The idea of FUSE (Filesystem in Userspace) is to make file system development easier by having the conveniences of userspace together with a helper library `libfuse`. In particular this allowed developers to use any programming language and easier abstractions. Elektra also tries hard

to make plugin development simple. For example, special interpreter plugins enable developers to also write plugins in different languages. Furthermore, other libraries also assist in creating plugins.
Of course not every feature of virtual file systems or Elektra has an analogy in the other system. If they would solve the same problem, one of them would be useless. The main differences are:

- API (get/set vs. read/write)
- commit semantics: one `kdbSet` can change many configuration files atomically. This is important if you want, e.g., a new host in `/etc/hosts` and use this host in some other configuration files.
- namespaces there are many places where the same configuration is stored. All of these configuration files have the same semantics and they override each other (think of command-line arguments, `/etc`, `$HOME/.config`, ...)
- Elektra interacts closely with the program execution environment such as command-line parsing. The namespace `proc` is specifically reserved for this purpose.
- in Elektra it is possible to create holes (files without directories above them) which are needed because of these override semantics: we want to be able to override a single value without duplicating the whole skeleton.
- validation: in Elektra you can describe how valid configuration should look like and reject invalid configuration.
- and much more...

114.2 Further Readings

- Compile and Install Elektra
- Then continue reading the tutorials
- Read about bindings
- Read about tools
- Look into the glossary.
- Start reading about command-line tools

Chapter 115

Buildserver

The Elektra buildserver handles a variety of tasks that reaches from testing Pull Requests (PRs) to deploying new versions of the Elektra website.
We reworked our build system to use a more modern Jenkinsfile approach with benefits such as

- tracking modifications to our build process in SCM
- tracking changes to the build environment
- speeding up builds

115.1 Setup

This section aims to give an introduction into this setup.

115.1.1 Multibranch Pipeline Jobs (libelektra)

We use the Jenkins Job type called Multibranch Pipeline provided by the Pipeline Multibranch Plugin for our CI tests called libelektra.
Simplified a multibranch pipeline job acts as an umbrella job that spawns child jobs for different branches (hence multibranch). The main purpose of the libelektra job is to scan the repository for changes to existing branches or to find new ones (for example branches that are used in PR's). It also contains the information on how to build the jobs in the form of a path that points to a Jenkinsfile containing more detailed instructions. The job also takes care of handling build artifacts that are archived and cleaning them out once the PRs are closed and a grace period has expired.
Summarized libelektra's job purpose is to combine the where (our Git repository), with a when (tracking changes via polling or webhooks) with a how (pointing to the Jenkinsfile + configuration).

115.1.2 Jenkins Shared Library

We use a shared library to share common functionalities across multiple pipelines. This shared library contains all docker related functionalities, code to publish artifacts and various other directives and helper functions.
After integrating this shared library with:

```
@Library('libelektra-shared') _
pipelineConfig {
  now = new Date()
}
```

we can use any function that is declared in the shared library.
It is available in the following repository. For more detailed usage instruction or instructions on how to integrate this shared library into Jenkins, see its README.md.

115.1.3 Jenkinsfiles

Jenkinsfiles describe what actions the build system should execute on which build slave. Currently Elektra uses four different files.

115.1.3.1 Jenkinsfile.daily

- Jenkinsfile.daily contains daily maintenance tasks, like cleaning up build servers.
- `Buildjob: libelektra-daily`
- `Jenkinsfile.daily`

115.1.3.2 Jenkinsfile.monthly

- Jenkinsfile.monthly contains monthly triggering of libelektra job on master.
- `Buildjob: libelektra-monthly`
- `Jenkinsfile.monthly`

115.1.3.3 Jenkinsfile.release

- Triggered manually with optional arguments. It is used for automated releasing a new version of Elektra.
- Jenkinsfile.release contains description how to release a new version of Elektra.
- `Buildjob: libelektra-release`
- `Jenkinsfile.release`

115.1.3.4 Jenkinsfile

- Triggered on code changes and is for testing changes to the codebase.
- Jenkinsfile contains descriptions how to build, test and deploy Elektra.
- `Buildjob: libelektra`
- `Jenkinsfile`

115.1.3.5 DSL

The language used is a groovy based DSL described in the `Jenkinsfile book`. Most groovy syntax is allowed to describe pipelines in a Jenkinsfile, but it is executed in a sandbox which might block certain calls.
Since plugins might extend the pool of available commands or variables a full list of currently available syntax can be seen in `pipeline syntax` after a login to the build server. Some functionality is not covered by this page when the responsible plugin is not implementing it. Usually an approach similar to what is described on `stackoverflow` can be used to track down the responsible code inside the providing plugins source code.
We also provide a number of helper functions in our Jenkinsfiles that are documented at the function head. Most common use cases, for example adding a new build with certain CMake flags, are easy to add because of them. For example, the configuration that is responsible for the `debian-stable-full` stage is generated completely by a single helper function called `buildAndTest`:

```
tasks << buildAndTest(
  "debian-stable-full",
  DOCKER_IMAGES.stretch,
  CMAKE_FLAGS_BUILD_ALL +
    CMAKE_FLAGS_BUILD_FULL +
    CMAKE_FLAGS_BUILD_STATIC +
    CMAKE_FLAGS_COVERAGE
  ,
  [TEST.ALL, TEST.MEM, TEST.NOKDB, TEST.INSTALL]
)
```

When adding new stages to the build chain it is generally a good idea to look up an existing stage which does something similar and adapt it to the new use case.

115.1.4 Security

Since a malicious PR could easily destroy the build server and slaves or expose credentials some restrictions are introduced. Only PR authors that have the right to push to libelektra can modify the Jenkinsfile and have those changes be respected for the respective branch.

115.1.5 Test Environments

We use Docker containers to provide the various test environments.
They are described in the repository and the Jenkinsfile describes how to build them. If a rebuild of the images is needed is determined by the hash of the Dockerfile used to describe it. If it has not changed the build step will be skipped. In the case that an image needed a build it will afterwards be uploaded into a private Docker image registry (on a7) and thus is shared between all Docker capable build slaves.

115.2 Jenkinsfile (Main CI Pipeline)

115.2.1 Tests

We will use the Docker images build as described earlier to run compilations and tests for Elektra. This allows us to run tests independent of which nodes are available (as the environment is portable).
The Jenkinsfile describes the steps used to run tests. Helper functions for easily adding new tests are available (`buildAndTest`, `BuildAndTestAsan`, ...).
The `withDockerEnv` helper, that is available in the shared library, makes sure to print the following information at the start of a test branch:

- branch name
- build machine
- docker image id

Coverage reports are generated automatically when using the `buildAndTest` helper and the appropriate CMake flags for coverage generation have been set. They are uploaded to https://doc.libelektra.org/coverage/.
Artifacts from `ctest` are also preserved automatically when using `buildAndTest`. The function also takes care of providing a stage name based path so multiple tests can not overwrite files that share the same name.
Tests are executed in order dictated by the Jenkinsfile. In general new tests should be added to the 'full build stage' that will only run after a standard full test run succeeded. This saves execution time on the build server for the most common errors.
Since there is no strict way to enforce the way tests are added we encourage you to read the existing configuration and modify existing tests so they suite your needs.

115.2.2 Deployment

For runs of the build job that are run in the master branch we also execute deployment steps after all tests pass. We use these steps to build Debian and Fedora packages and move them into the repositories hosted on the community node.
Additionally we recompile the homepage and deploy it on the community node.

115.3 Jenkinsfile.release (Release Pipeline)

115.3.1 Release Builds

The `buildRelease` function runs various test suites and collects and archives debug information. It also generates source-packages and packages for Debian based distributions and Fedora.
The generated packages are additionally installed and tested on a clean Docker image.

115.3.2 Manual approval

In order to verify the correct behavior and results of the previous steps and their generated artifacts, we use the `Input Step` to pause the pipeline. After manually verifying the correctness, the pipeline run can be resumed.

115.3.3 Publishing

The previously generated packages are published to the repositories, as described above in *Jenkinsfile (Main CI Pipeline)*. The API documentation and source packages get published to the relevant Git repositories. Finally the Alpine release image is published to Docker Hub.

115.4 Jenkins Setup

This section describes how to replicate the current Jenkins configuration.

115.4.1 Jenkins libelektra Configuration

The `libelektra` build job is a multibranch pipeline job. It is easiest to add via the BlueOcean interface.
Most of the default settings should be ok, however some settings need to be verified or added to build Elektra correctly:

- `Advanced clone behaviors` should be added and the path to the Git mirror needs to be specified: `/home/jenkins/git_mirrors/libelektra`. This reference repository is created and maintained by our `daily buildjob`.
- Under Property strategy you can add `Trigger build on pull request comment. jenkins build (libelektra|all) please` is a good starting point. This functionality is provided by the `GitHub PR Comment Build Plugin`.
- For Build Configuration you want to specify `by Jenkinsfile` and add the script path: `scripts/jenkins/Jenkinsfile`.

115.4.2 Jenkins libelektra-release Configuration

The `libelektra-release` job is a normal pipeline job.
Most of the default settings can be used.

- Under General you need to enable `Permission to Copy Artifact` and set the field to `*`.
- For Build Configuration you want to specify `by Jenkinsfile` and add the script path: `scripts/jenkins/Jenkinsfile.release`.

115.4.3 General Set-Up of a New Pipeline

We will cover how to set-up a pipeline that uses a different Git repository than this `libelektra` repository.
Most of the default settings can be used.

- If a private repository were to be used with a GitHub type job, a credentials pair of `elektrabot` must be selected to access this repository.
- For Build Configuration you want to specify `by Jenkinsfile` and add the script path to the Jenkinsfile.

We first start by including the shared library as described above in *Jenkins Shared Library*. Then we declare in a Jenkinsfile an empty map that will contain all Docker images we will use in this pipeline:

```
DOCKER_IMAGES = [:]
```

This map will later be populated in a function. For a Docker image with Ubuntu Focal and the Dockerfile located at `./ci/Dockerfile` this function would need to look as follows:

```
def dockerInit() {
  node('master') {
    checkout scm
    DOCKER_IMAGES.focal = dockerUtils.createDockerImageDesc(
      'ubuntu-focal', dockerUtils.&idTesting,
      './ci/',
      './ci/Dockerfile'
    )
    // more docker images ...
  }
}
```

We can now begin to create the stages that initialize, pull and if necessary also build these Docker images:

```
stage('Init docker images') {
```

```
  dockerInit()
}
stage('Pull docker images') {
  parallel dockerUtils.generateDockerPullStages(DOCKER_IMAGES)
}
maybeStage('Build docker images', DOCKER_IMAGES.any { img -> !img.value.exists }) {
  lock('docker-images') {
        parallel dockerUtils.generateDockerBuildStages(DOCKER_IMAGES)
  }
}
```

Now the actual stages that contain our CI/CD code can be created.
To allow parallel execution of stages we create generator functions that return a map of stages that can be executed in parallel. For example the following generator function would create a map of two tasks that could be run in parallel.

```
def generateTestStages() {
  def tasks = [:]
  tasks << test(
    'ubuntu-focal-test',
    DOCKER_IMAGES.focal
  )
  tasks << test(
    'ubuntu-bionic-test',
    DOCKER_IMAGES.bionic
  )
  return tasks
}
```

Suppose we have a bash script `test.sh` in the root of our Git repository. This script can be executed inside one of our Docker images by using the `withDockerEnv` function:

```
def test(stageName, image) {
  return [(stageName): {
    stage(stageName) {
      withDockerEnv(image) {
        sh "./test.sh"
      }
    }
  }]
}
```

We can now call the generator function `generateTestStages` in a new stage:

```
stage('Testing') {
  parallel generateTestStages()
}
```

115.4.4 Adding a Jenkins Node

A node needs to have a JRE (Java Runtime Environment) installed. Further it should run an SSH (Secure SHell) server. Docker need to be installed as well.
A `jenkins` user with 47110:47110 ids should be created as this is what is expected in Docker images. `useradd -u 47110 jenkins` Additionally a public key authentication should be set up so the jenkins master can establish an SSH connection with the node. If the node should be able to interact with Docker the jenkins user should be added to the `docker` group.
Nodes should be set to only build jobs matching their labels and to be online as much as possible. As for labels `gitmirror` should be if you want to cache repositories on this node. If Docker is available the `docker` label should be set.
All files and folders in the Node under `/home/jenkins` should be owned by user `jenkins`.

115.5 Understanding Jenkins Output

Our Jenkins build uses parallel steps inside a single build job to do most of the work. To reliably determine which stages failed it is best to look over the build results in the Jenkins Blue Ocean view. It is the default View opened when accessing the build results from GitHub. For libelektra the URLs are https://build.libelektra.org/job/libelektra/ and https://build.libelektra.org/blue/organizations/jenkins/libelektra/branches/ Failed stages are marked in red. On selecting a stage you can further expand all steps in the stage. Of interest should be one of the first steps which echos out the Docker image used to run the stage. You also want to look for whatever failed (which should be in a step also marked red to indicate failure).

115.6 Reproducing Build Server Errors Locally

First you have to determine which image is used. This is described above in *Understanding Jenkins output*.
Afterwards you can download it from our registry via `docker pull`. Pay attention that you have to use **hub-public.libelektra.org** as this subdomain does not require authentication for GET operations used by the Docker client. As an example:

```
docker pull
      hub-public.libelektra.org/build-elektra-alpine:201809-791f9f388cbdff0db544e02277c882ad6e8220fe280cda67e6ea6358767a065e
```

You can also rebuild the images locally, which is useful if you want to test changes you made to the Dockerfiles themselves. Locate which Dockerfile you need by looking up the reference the stage that used it in the Jenkinsfile. For *alpine* this would be `DOCKER_IMAGES.alpine`. You can search for this entry in the Jenkinsfile to find that this image is build from the context `./scripts/docker/alpine/*` and uses `./scripts/docker/alpine/*/Dockerfile` as a Dockerfile. Now you can build the image as described in `scripts/docker/README.md`.
You can find more information on how to use our images in `scripts/docker/README.md`.
Alternatively, you can follow the tutorial.

115.7 Modify Test Environments

You can also modify the test environments (update a dependency, install a new dependency, ...) by editing the Dockerfiles checked into SCM. Determine which ones you need to modify by tracing which images are used for what stages via the `DOCKER_IMAGES` map used in our Jenkinsfile. The `dockerInit` function should be a good start point.
Following docker best practises you should try to minimize layer count when possible. Our docker images are quite large and hence take a long time to build so make sure to test any modifications locally first.

115.8 Triggers

All Triggers are described in the configuration of the respective build jobs.
The `libelektra` build is triggered for all branches of the libelektra repository except for `debian`. Additionally, all open branches in forks targeting libelektra's repository via PRs are going to be build. Pushes to any of those branches will trigger a new build automatically.
The `daily build` and `monthly build` are executed according to a cron schedule.
The `release pipeline` is triggered manually with optional arguments.
The following phrases can be used as comments to manually trigger a specific build:

- jenkins build `libelektra` please
- jenkins build `daily` please
- jenkins build `monthly` please

Additionally `jenkins build all please` can be used to trigger all build jobs relevant for PR's. This is not necessary anymore as all relevant tests have been moved into the libelektra job.

115.9 Authorization

If you are not yet authorized, the following question will be asked (by user @markus2330):

```
Can one of the admins verify if this patch should be build?
```

Then one of the admins (sorted by activity):

- @sanssecours
- @markus2330
- @beku
- @BernhardDenner

- @fberlakovich
- @manuelm

need to confirm by saying:

```
.*add\W+to\W+whitelist.*
```

or if just the pull request should be checked:

```
.*build\W+allow.*
```

115.10 Issues with the Build Environment

If you have issues that are related to the build system you can open a normal issue and tag it with `build` and `question`. If you feel like your inquiry does not warrant an issue on its own, please use `our buildserver issue`.

Chapter 116

CODING

This document provides an introduction in how the source code of libelektra is organized and how and where to add functionality.

116.1 Folder structure

After you downloaded and unpacked Elektra you should see some folders. The most important are:

- **src:** This directory contains the source of the libraries, tools and plugins.
- **doc:** General documentation for the project and the core library.
- **examples:** Examples on how to use the core library.
- **tests:** Contains the testing framework for the source (**src**).

You can find more information about the structure in the `README.md`.

116.2 General Information

- Make sure that the codestyle of contributions complies with the already established codestyle of this project, except if this document says otherwise: then please open an issue for offending code.
- We use Unix line endings
- Large files should be in https://blobs.libelektra.org/ or https://rawdata.libelektra.org
- 'find . -not -regex '[/.a-zA-Z0-9_-]*' should not find your file names

116.3 Source Code

The same guidelines apply to all code in the repository including the plugins.

116.3.1 General Guidelines

You are only allowed to break a guideline if there is a good reason to do so. When you do, document the fact as comment next to the code.
Of course, all rules of good software engineering apply: Use meaningful names and keep the software both testable and reusable.
The purpose of the guidelines is to have a consistent style, not to teach programming.
If you see code that breaks guidelines do not hesitate to fix them or at least open issues.
If you see inconsistency within the rules do not hesitate to open an issue about it.

116.3.2 Code Comments

Code is not only for the computer, but it should be readable for humans, too. Up-to-date code comments are essential to make code understandable for others. Thus please use following techniques (in order of preference):

1. Comment functions with `/**/` and Doxygen, see below.
2. You should also add assertions to state what should be true at a specific position in the code. Their syntax is checked and they are automatically verified at run-time. So they are not only useful for people reading the code but also for tools. Assertions in Elektra are used by:

   ```
   #include <kdbassert.h>
   ELEKTRA_ASSERT (condition, "formatted text to be printed when assert fails", ...)
   ```

   Note: Do not use assert for user-APIs, always handle arguments of user-APIs like untrusted input.
3. If the "comment" might be useful to be printed during execution, use logging:

   ```
   #include <kdblogger.h>
   ELEKTRA_LOG ("formatted text to be printed according to log filters", ...)
   ```

   Read HERE for how to enable the logger.
4. Otherwise comment within source with `//` or with `/**/` for multi-line comments. Use `TODO` to indicate that something is not yet done. Before merging, relevant `TODO`s should be fixed or `issues` created for left-overs.

116.3.3 Coding Style

- Limits
  - Functions should not exceed 100 lines.
  - Files should not exceed 1000 lines.
  - A line should not be longer than 140 characters.

  Split up when those limits are reached. Rationale: Readability with split windows.
- Indentation
  - Use tabs for indentation.
  - One tab equals 8 spaces.
- Blocks
  - Use blocks even for single line statements.
  - Curly braces go on a line on their own on the previous indentation level.
  - Avoid multiple variable declarations at one place.
  - Declare Variables as late as possible, preferable within blocks.
- Naming
  - Use camelCase for functions and variables.
  - Start types with upper-case, everything else with lower-case.
  - Prefix names with `elektra` for internal usage. External API either starts with `ks`, `key` or `kdb`.
- Whitespaces
  - Use space before and after equal when assigning a value.
  - Use space before round parenthesis (`(`).
  - Use space before and after `*` from Pointers.

- Use space after `,` of every function argument.

The reformat script can ensure most code style rules, but it is obviously not capable of ensuring everything (e.g. naming conventions). So do not give this responsibility out of hands entirely. You can use docker to ensure that you have the correct version of all our reformatting tools at hand. You can also enable that formatting runs automatically before commiting using `scripts/dev/enable-pre-commit-hook` and disable it again using `scripts/dev/disable-pre-commit-hook`

116.3.4 C Guidelines

- The compiler shall not emit any warning (or error).
- Use goto only for error situations.
- Use `const` as much as possible.
- Use `static` methods if they should not be externally visible.
- C-Files have extension `.c`, Header files `.h`.
- Use internal functions: prefer to use `elektraMalloc`, `elektraFree`.

Example: `src/libs/elektra/kdb.c`

116.3.4.1 Clang Format

To guarantee consistent formatting we use `clang-format` (version `12`) to format all C and C++ code in the repository. Since our build servers also check the style for every pull request you might want to make sure you reformat your C/C++ code changes with this tool.
To find out which version of `clang-format` a certain build server uses please check:

- the Debian sid Docker image,
- the GitHub Actions configuration files, and
- the Cirrus configuration file

and search for the relevant packages (`clang-format`, `llvm`). Currently we use clang-format `12`

- in the GitHub Actions configuration files,
- in the Debian sid Docker container on the Jenkins build server, and
- in the Cirrus macOS build jobs.

116.3.4.1.1 Installation

macOS On macOS you can install `clang-format` using `Homebrew` either directly:
```
brew install clang-format
```
or by installing the whole `LLVM` infrastructure:
```
brew install llvm
```
Please note, that both of these commands will install current versions of `clang-format` that might format code a little bit differently than Clang-Format `12` in certain edge cases. If you want you can also install a specific version of LLVM and Clang-Format: e.g. Clang-Format `11` using LLVM `11`:
```
brew install llvm@11
```

Debian In Debian (Sid) the package for Clang-Format `12` is called `clang-format-12`:
```
apt-get install clang-format-12
```

116.3.4.1.2 Usage For the basic use cases you can use `clang-format` directly. To do that, just call the tool using the option `-i` and specify the name of the files you want to reformat. For example, if you want to reformat the file `src/bindings/cpp/include/kdb.hpp` you can use the following command:

```
# On some systems such as Debian the `clang-format` executable also contains
# the version number. For those systems, please replace `clang-format`,
# with `clang-format-12` in the command below.
clang-format -i src/bindings/cpp/include/kdb.hpp
```

While this works fine, if you want to format only a small number of file, formatting multiple files can be quite tedious. For that purpose you can use the script `reformat-c` that reformats all C and C++ code in Elektra's code base

```
scripts/dev/reformat-c # This script will probably take some seconds to execute
```

116.3.4.1.3 Tool Integration If you work on Elektra's code base regularly you might want to integrate the formatting step directly in your development setup. `ClangFormat's homepage` includes a list of integrations for various tools that should help you to do that. Even if this webpage does not list any integrations for your favorite editor or IDE, you can usually add support for external tools such as `clang-format` to advanced editors or IDEs pretty easily.

TextMate While `TextMate` supports `clang-format` for the current file directly via the keyboard shortcut `ctrl+⇑+H`, the editor does not automatically reformat the file on save. To do that

1. open the bundle editor ("Bundles" → "Edit Bundles"),
2. navigate to the "Reformat Code" item of the C bundle ("C" → "Menu Actions" → "Reformat Code"),
3. insert `callback.document.will-save` into the field "Semantic Class", and
4. change the menu option for the field "Save" to "Nothing"

After that change TextMate will reformat C and C++ code with `clang-format` every time you save a file.

116.3.5 C++ Guidelines

- Everything as in C if not noted otherwise.
- Do not use goto at all, use RAII instead.
- Do not use raw pointers, use smart pointers instead.
- C++-Files have extension `.cpp`, Header files `.hpp`.
- Do not use `static`, but anonymous namespaces.
- Write everything within namespaces and do not prefix names.
- Oriented towards `more safe and modern usage`.

Example: `src/bindings/cpp/include/kdb.hpp`

116.3.6 CMake Guidelines

We use a similar style for CMake as we do for other code:

- The length of a functions should not exceed 100 lines.
- The length of a file should not exceed 1000 lines.
- A line should not be longer than 140 characters.
- Use tabs for indentation.
- One tab equals 8 spaces.
- Declare variables as late as possible, preferable within blocks.
- Add a space character before round parenthesis (`(`).
- Use lower case for command names (e.g. `set` instead of `SET`)

116.3.6.1 CMake format

We use cmake-format to reformat code according to the guidelines given above. Since cmake-format currently does not support tabs, we use the standard command unexpand to fix this issue. For example, to reformat the file CMakeLists.txt in the root folder of the repository we use the following command:

```
# This command uses `sponge`, which is part of the [moreutils](https://joeyh.name/code/moreutils/) package.
cmake-format CMakeLists.txt | unexpand | sponge CMakeLists.txt
```

116.3.6.1.1 Installation Since cmake-format is written in Python you usually install it via Python's package manager pip:

```
# Install cmakelang `0.6.13` with support for YAML config files
pip install cmakelang[yaml]==0.6.13
```

Please make sure, that you install the correct version (0.6.13) of cmake-format:

```
cmake-format --version
#> 0.6.13
```

since otherwise the formatted code might look quite different.

We also use the moreutils in our CMake formatting script, which you can install on macOS using Homebrew:

```
brew install moreutils
```

and on Debian using apt-get:

```
apt-get install moreutils
```

116.3.6.1.2 Usage If you want to reformat the whole codebase you can use the script `reformat-cmake`:

```
scripts/dev/reformat-cmake # Running this script for the whole code base takes some time.
```

To reformat specific files add a list of file paths after the command:

```
# The command below reformats the file `cmake/CMakeLists.txt`.
scripts/dev/reformat-cmake cmake/CMakeLists.txt
```

116.3.6.1.3 Tool Integration If you work on CMake code quite often you probably want to integrate cmake-format into your development workflow. The homepage of cmake-format list some integration options.

TextMate While TextMate does not support cmake-format directly, you can quickly create a command that applies cmake-format every time you save a CMake file yourself. The steps below show one option to do that.

1. Open the "Bundle Editor": Press ^ + alt + cmd + B
2. Create a new command:
   (a) Press cmd + N
   (b) Select "Command"
   (c) Press the button "Create"
3. Configure your new command
   (a) Use "Reformat Document" or a similar text as "Name"
   (b) Enter source.cmake in the field "Scope Selector"
   (c) Use ^ + ⇑ + H as "Key Equivalent"
   (d) Copy the text callback.document.will-save into the field "Semantic Class"
   (e) Select "Document" as "Input"
   (f) Select "Replace Document" in the dropdown menu for the option "Output"
   (g) Select "Line Interpolation" in the menu "Caret Placement"
   (h) Copy the following code into the text field:
   ``` #!/bin/bash
   set -o pipefail if ! "${TM_CMAKE_FORMAT:-cmake-format}" - | ${TM_CMAKE_FORMAT_FILTER:-tee}; then . "$TM_SUPPORT_PATH/lib/bash_init.sh" exit_show_tool_tip fi ```
   (i) Save your new command: cmd + S
   (j) Store the value unexpand in the variable TM_CMAKE_FORMAT_FILTER. To do that save the text

```
TM_CMAKE_FORMAT_FILTER = "unexpand"
```

in a file called .tm_properties in the root of Elektra's repository.
```

116.3.7 Java & Groovy Guidelines

Please follow Google Java Style Guide for Groovy (used by Jenkins) and Java files.
Most notably use:

- 2 spaces for indentation
- Variable and function names in lowerCamelCase
- K & R style brackets

116.3.7.1 Usage

If you want to reformat all Java files in the repository you can use the script `reformat-java`:
```
scripts/dev/reformat-java
```
To reformat specific files add a list of file paths after the command:
```
# The command below reformats the file 'src/bindings/jna/libelektra/src/main/java/org/libelektra/KDB.java'.
scripts/dev/reformat-java src/bindings/jna/libelektra/src/main/java/org/libelektra/KDB.java
```

116.3.8 JavaScript Guidelines

116.3.8.1 Prettier

We use prettier to format JavaScript files.

116.3.8.1.1 Installation

macOS On macOS you can install prettier using Homebrew:
```
brew install prettier
```

General To install prettier using Node's package manager npm you can use the command below
```
npm install --global prettier@2.8.4
```

116.3.8.1.2 Usage To format all JavaScript files in the repository you can use the script `reformat-javascript`:
```
scripts/dev/reformat-javascript
```
To format only some files, please specify a list of filenames after the command:
```
scripts/dev/reformat-markdown src/tools/qt-gui/qml/ErrorDialogCreator.js # Reformat this file
```

116.3.9 Markdown Guidelines

- File Ending is `.md` or integrated within Doxygen comments
- Only use # characters at the left side of headers/titles
- Use fences for code/examples
- Prefer fences which indicate the used language for better syntax highlighting
- Fences with sh are for the shell recorder syntax
- `README.md` and tutorials should be written exclusively with shell recorder syntax so that we know that the code in the tutorial produces output as expected
- Please use **`title-case`** for headings in the general documentation.
- For man pages please use **only capital letters for subheadings** and only **small letters for the main header**. We use this header style to match the look and feel of man pages for Unix tools such as `ls` or `mkdir`.

116.3.9.1 Prettier

We use `prettier` to format the documentation according to the guidelines given above.
Under certain **exceptional** circumstances you might want to prevent `prettier` from formatting certain parts of a Markdown file. To do that you can

- enclose the Markdown code in `<!-- prettier-ignore-start -->` and `<!-- prettier-ignore-end -->` tags, or
- use `<!-- prettier-ignore -->` to disable formatting till the end of a file

116.3.9.1.1 Installation For information on how to install the tool, please take a look at "JavaScript Guidelines" → "Prettier" → "Installation".

116.3.9.1.2 Usage You can format all Markdown files in the repository using the script `reformat-markdown`:

```
scripts/dev/reformat-markdown
```

To format only some files, please specify a list of filenames after the command:

```
scripts/dev/reformat-markdown doc/CODING.md # Reformat this file
```

116.3.9.1.3 Tool Integration The homepage of Prettier lists various options to integrate the tool into your workflow.

TextMate To reformat a Markdown document in TextMate every time you save it, please follow the steps listed below.

1. Open the "Bundle Editor": Press ^ + `alt` + `cmd` + `B`
2. Create a new command:
   (a) Press `cmd` + `N`
   (b) Select "Command"
   (c) Press the button "Create"
3. Configure your new command
   (a) Use "Reformat Document" or a similar text as "Name"
   (b) Enter `text.html.markdown` in the field "Scope Selector"
   (c) Use ^ + ⇑ + `H` as "Key Equivalent"
   (d) Copy the text `callback.document.will-save` into the field "Semantic Class"
   (e) Select "Document" as "Input"
   (f) Select "Replace Input" in the dropdown menu for the option "Output"
   (g) Select "Line Interpolation" in the menu "Caret Placement"
   (h) Copy the following code into the text field:
   ``` #!/bin/bash
   if ! "${TM_PRETTIER:-prettier}" –stdin –stdin-filepath "${TM_FILEPATH}" then . "$TM_SUPPORT_↩ PATH/lib/bash_init.sh" exit_show_tool_tip fi ```
   (i) Save your new command: `cmd` + `S`

### 116.3.10 Shell Guidelines

- Please only use POSIX functionality.
   ```

116.3.10.1 Shebang

Every shell script must start with either of two shebangs:

- `#!/bin/sh`
- `#!/usr/bin/env <shell>`, where `<shell>` is an appropriate shell, e.g. `#!/usr/bin/env bash`

Note that even if a shebang is added by a preprocessor macro or similar code generation tools, it must also be present in the templated file.

116.3.10.2 shfmt

We use `shfmt` to format Shell files in the repository.

116.3.10.2.1 Installation

macOS You can install `shfmt` on macOS using `Homebrew`:

```
brew install shfmt
```

General `shfmt's GitHub release page` offers binaries for various operating systems. For example, to install the binary for the current user on Linux you can use the following command:

```
mkdir -p "$HOME/bin" && cd "$HOME/bin" && \
  curl -L "https://github.com/mvdan/sh/releases/download/v3.3.1/shfmt_v3.3.1_linux_amd64" -o shfmt && \
  chmod u+x shfmt
```

Please note that you have to make sure, that your `PATH` includes `$HOME/bin`, if you use the command above:

```
export PATH=$PATH:"$HOME/bin"
```

116.3.10.2.2 Usage We provide the script `reformat-shell` that formats the whole codebase with `shfmt`:

```
scripts/dev/reformat-shell
```

You can also reformat specific files by listing filenames after the script:

```
scripts/dev/reformat-shell scripts/dev/reformat-shell # Reformat the source of `reformat-shell`
```

116.3.10.2.3 Tool Integration The GitHub project page of `shfmt` offers some options to integrate the tool into your development workflow `here`.

TextMate The steps below show you how to create a `TextMate` command that formats a documents with `shfmt` every time you save it.

1. Open the "Bundle Editor": Press ^ + `alt` + `cmd` + `B`
2. Create a new command:
   (a) Press `cmd` + `N`
   (b) Select "Command"
   (c) Press the button "Create"
3. Configure your new command
   (a) Use "Reformat Document" or a similar text as "Name"
   (b) Enter `source.shell` in the field "Scope Selector"
   (c) Use ^ + ⇑ + `H` as "Key Equivalent"
   (d) Copy the text `callback.document.will-save` into the field "Semantic Class"
   (e) Select "Document" as "Input"
   (f) Select "Replace Input" in the dropdown menu for the option "Output"
   (g) Select "Line Interpolation" in the menu "Caret Placement"
   (h) Copy the following code into the text field:
   ``` #!/bin/bash
   if ! "${TM_SHFMT_FORMAT:-shfmt}" -s -sr; then . "$TM_SUPPORT_PATH/lib/bash_init.sh" exit_↩ show_tool_tip fi ```
   (i) Save your new command: `cmd` + `S`
   ```

116.3.11 Doxygen Guidelines

`doxygen` is used to document the API and to build the html and pdf output. We also support the import of Markdown pages. Doxygen 1.8.8 or later is required for this feature (Anyway, you can find the `API Doc` online). Links between Markdown files will be converted with the Markdown Link Converter. **Markdown pages are used in the pdf, therefore watch which characters you use and provide a proper encoding!**

- use `@` to start Doxygen tags
- Do not duplicate information available in Git in Doxygen comments.
- Use `@copydoc`, `@copybrief` and `@copydetails` intensively (except for file headers).

Further Doxygen documentation can be found in the Doxygen README.

116.3.12 File Headers

Files should start with:
Note:

- `@ file` has *no* parameters.
- `@ brief` should contain a short statement about the content of the file and is needed so that your file gets listed at `https://doc.libelektra.org/api/master/html/files.html`

The duplication of the filename, author and date is not needed, because this information is tracked using Git and doc/AUTHORS.md already.

116.4 Further Readings

- Make sure to read DESIGN together with this document.

Chapter 117

Compile

117.1 Dependencies

For the base system you only need `cmake3`, `Git`, a C99 compiler and essential build tools (make and some standard Unix tools; alternatively ninja and clang are also supported but not described here). Those can be installed as follows:

- on APT-based systems (Ubuntu, Debian):

```
sudo apt-get install cmake git build-essential
```

- on RPM-based systems (CentOS, Red Hat Enterprise Linux, Oracle Linux):

```
sudo yum install -y cmake git gcc-c++ make
```

- on macOS, most of the build tools can be obtained by installing Xcode (from the App Store). Other required tools may be installed using `brew`. First install brew as described on their website. Then issue the following command to get cmake and git in order to complete the basic requirements:

```
brew install cmake git
```

117.2 Quick Guide

Run the following commands to compile Elektra with non-experimental parts where your system happens to fulfill the dependencies (continue reading the rest of the document for details about these steps):

```
git clone https://github.com/ElektraInitiative/libelektra.git
cd libelektra
mkdir build
cd build
cmake ..  # watch output to see if everything needed is included
ccmake .. # optional: overview of the available build settings (needs cmake-curses-gui)
cmake --build . -- -j5
cmake --build . --target run_nokdbtests # optional: run tests
```

The last line only runs tests without writing onto your system. See TESTING for how to run more tests. Afterwards you can use `sudo make install && sudo ldconfig` to install Elektra. See INSTALL for more information about installation of self-compiled Elektra (such as how to uninstall it).

117.3 Optional Dependencies

> Note: You do not need to install the dependencies listed here. If they are not available, some of the functionality gets disabled automatically. The core of Elektra never depends on other libraries.

117.3.1 Documentation dependencies

To build the documentation you need doxygen (we recommend 1.8.8+), graphviz and `ronn-ng`. These can be installed as follows:

- on APT-based systems (Ubuntu, Debian):

```
apt-get install doxygen graphviz ruby
gem install ronn-ng -v 0.10.1.pre1
```

- on RPM-based systems (CentOS, Red Hat Enterprise Linux, Oracle Linux):

```
sudo yum install -y doxygen docbook-style-xsl graphviz ruby
gem install ronn-ng -v 0.10.1.pre1
```

- on macOS using brew:

```
brew install doxygen graphviz
brew install ruby # in case ruby is not already installed
gem install ronn-ng -v 0.10.1.pre1
```

To build PDF documentation you need `pdflatex`. You can install it as follows:

- on APT-based systems (Ubuntu, Debian):

```
apt-get install \
  texlive-latex-base \
  texlive-latex-recommended \
  texlive-latex-extra \
  texlive-fonts-recommended \
  texlive-fonts-extra \
  texlive-science
```

117.3.2 Plugin dependencies

For dependencies of plugins, please refer to the `README.md` of the respective plugin.
A small subset of build dependencies to get you started:

- on RPM-based systems (CentOS, Red Hat Enterprise Linux, Oracle Linux):

```
sudo yum install -y libdb-devel GConf2-devel libxml2-devel yajl-devel   \
libcurl-devel augeas-devel libgit2-devel lua-devel swig python34-devel python-devel \
java-17-openjdk-devel jna ruby-devel byacc
```

- on APT-based systems (Ubuntu, Debian):

```
sudo apt install -y libxerces-c-dev libxml2-dev libyajl-dev \
libcurl4-gnutls-dev libaugeas-dev git git-buildpackage dh-lua liblua5.2-dev \
dh-python python3-all python3-dev openjdk-17-jdk libjna-java ruby-dev flex bison
```

117.4 Preparation

Elektra uses `CMake3`. Tested are CMake version 3.0.2 and 3.7.2 among others.
To configure Elektra graphically (with curses) run (`..` belongs to command):

```
mkdir build && cd build && ccmake ..
```

and press `c` to configure the cache (might be necessary multiple times, and once on the first time in case you don't see any settings). After applying the desired settings, press `g` to generate the make file.
All options described here, can also be used with `cmake` rather than `ccmake` (`..` does also here belong to the command):

```
mkdir build && cd build && cmake -D<OPTION1>=<VAR1> -D<OPTION2>=<VAR2> ..
```

For information what you can use as `OPTION1` and `OPTION2`, see above. Note: You have to enclose a value with quotes `" "` if it contains a semicolon (`;`). E.g.:

```
cmake -DPLUGINS="dump;resolver_fm_hpu_b;yajl;list;spec" ..
```

Some scripts in the folder of the same name may help you running CMake.

117.4.1 Compilers

You should be able to compile Elektra with any C99 compiler. For a list of compilers we test with have a look at:

- our Docker containers orchestrated by our Jenkinsfile being built on `our build server`
- Cirrus
- GitHub Actions

Here is an additional list of compilers used by developers (for build servers, see links above):

Compiler	Version	Target
gcc	gcc (Debian 8.3.0-6) 8.3.0	x86_64-linux-gnu
gcc	gcc (Debian 10.2.1-6) 10.2.1 20210110	x86_64-linux-gnu
gcc	gcc (GCC) 11.2.1 20220127 (Red Hat 11.2.1-9)	x86_64-redhat-linux
gcc	gcc (GCC) 12.2.1 20220819 (Red Hat 12.2.1-2)	x86_64-redhat-linux
gcc	gcc-12 (Homebrew GCC 12.2.0) 12.2.0	x86_64-apple-darwin21
gcc	Homebrew clang version 15.0.1	x86_64-apple-darwin21.↩ 6.0
clang	clang version 14.0.5 (Fedora 14.0.5-1.fc36)	x86_64-redhat-linux-gnu
clang	Apple clang version 14.0.0 (clang-1400.0.29.102)	x86_64-apple-darwin21.↩ 6.0

(1) OpenBSD ships an old version of GCC per default, which can not compile Elektra. A manual installation of egcc/eg++ is required. Note that not every OpenBSD mirror provides the eg++ package. Elektra builds are confirmed with egcc/eg++ 4.9.4 in OpenBSD 6.3. The packages are called gcc and g++. Compile with `CC=/usr/local/bin/egcc CXX=/usr/local/bin/eg++`.

To change the compiler, use CMake settings `CMAKE_C_COMPILER` and `CMAKE_CXX_COMPILER`.
To use gcc-4.3 for example

```
cmake -DCMAKE_C_COMPILER=gcc-4.3 -DCMAKE_CXX_COMPILER=g++-4.3 ..
```

To change the compiler with `ccmake`, you may need to toggle advanced options (key `t`).

117.4.2 Options

Some options, i.e. `PLUGINS`, `BINDINGS` and `TOOLS` are either:

- a list of elements separated with a semicolon (`;`) (note that shells typically need `;` to be escaped)
- a special uppercase element that gets replaced by a list of elements, that are:
  - `ALL` to include all elements (except elements with unfulfilled dependencies)
  - `NODEP` to include all elements without dependencies
- elements prefixed with a minus symbol (`-`) to exclude an element

Examples for this are especially in the subsection `PLUGINS` below, but they work in the same fashion for `BINDINGS` and `TOOLS`.

117.4.2.1 Plugins

Read about available plugins here.
Because the core of Elektra is minimal, plugins are needed to actually read and write to configuration files (*storage plugins*), commit the changes (*resolver plugins*, also takes care about how the configuration files are named) and also do many other tasks related to configuration.
The minimal set of plugins you should add:

- dump is the default storage. If you remove it, make sure you add another one and set `KDB_DEFAULT_↩ STORAGE` to it.
- resolver is the default resolver. If you remove it, make sure you add another one and set `KDB_DEFAULT_↩ RESOLVER` to it.
- spec copies metadata from spec namespace to other namespaces. Needed for tests. (Required with `ENABLE_TESTING`, except on mingw.)
- sync is very useful to not lose any data. If you do not want to include it, make sure to set `/sw/elektra/kdb/#0/current/plugins` to a value not containing sync (e.g. an empty value). See kdb-mount(1).

By default CMake adds nearly all plugins if the dependencies are present. Only experimental plugins will be omitted by default:
```
-DPLUGINS="ALL;-EXPERIMENTAL"
```
To add also experimental plugins, you can use:
```
-DPLUGINS=ALL
```

> Note that plugins are only built if their dependencies are satisfied. So make sure to install all dependencies you need before you run `cmake`. For example, to include the plugin `yajl`, make sure `libyajl-dev` is installed.

To add all plugins except some plugins you can use:
```
-DPLUGINS="ALL;-plugin1;-plugin2"
```
For example, if you want all plugins except the jni plugin you would use:
```
-DPLUGINS="ALL;-jni"
```
To add all plugins not having additional dependencies (they need only POSIX), you can use
```
-DPLUGINS=NODEP
```
Note, that every `infos/provides` and `infos/status` field written uppercase can be used to select plugins that way (see README of individual plugins). You also can combine any of these fields and add/remove other plugins to/from it, e.g. to include all plugins without deps, that provide storage (except `yajl`) and are maintained, but not include all plugins that are experimental, you would use:
```
-DPLUGINS="NODEP;STORAGE;-yajl;MAINTAINED;-EXPERIMENTAL"
```
The inclusion is determined by following preferences:

1. if the plugin is explicit excluded with `-plugin`
2. if the plugin is explicit included with `plugin`
3. if the plugin is excluded via a category `-CATEGORY`
4. if the plugin is included via a category `CATEGORY`
5. plugins are excluded if they are not mentioned at all (neither by category nor by name)

Note, that changing `PLUGINS` will not modify the defaults used after Elektra was installed. For this endeavour you need to change:
```
-DKDB_DEFAULT_RESOLVER=resolver
```
and
```
-DKDB_DEFAULT_STORAGE=dump
```
The default resolver and storage will write to `KDB_DB_FILE` and `KDB_DB_INIT` (for bootstrapping).
Obviously, you can pass the exact list of plugins you want, e.g.:
```
-DPLUGINS="resolver;sync;dump"
```
Some plugins are compile-time configurable. Then you can choose which features are compiled in or out. This is especially important in the bootstrapping phase, because then only the compiled in configuration applies. To compile-time-configure a plugin, you just pass a underscore (_) and flags after the name of the plugin.
The resolver for example distinguish between 3 different kind of flags:
```
-DPLUGINS="resolver_baseflags_userflags_systemflags"
```
The following base flags are available:

- `c` for debugging conflicts
- `f` for enabling file locking
- `m` for enabling mutex locking

The user flags are (the order matters!):

- `p` use passwd/ldap to lookup home directory using `getpwuid_r`
- `h` use the environment variable HOME
- `u` use the environment variable USER
- `b` use the built-in default CMake variable `KDB_DB_HOME`

The system flags are (the order matters!):

- x use the environment variable XDG_CONFIG_DIRS (: are interpreted as part of filename, no searching is done!) This option is not recommended (unless for testing), because it allows users to fake system configuration.
- b use the built-in default CMake variable KDB_DB_SYSTEM
- note: if a path that begins with a slash (/) is chosen, the system flags are irrelevant and the path is taken as-is.

For example, one may use:
```
-DPLUGINS="resolver_lm_uhpb_b"
```
To add resolver_l_h_b you need to specify
```
-DPLUGINS="resolver;resolver_l_h_b"
```
You can add resolver with any combination of the flags, even if they are not available in ALL.

117.4.2.2 Tools

Tools are used to add extra functionality to Elektra. The flag used to specify which tools are compiled is -DTOOLS, thus flag works similarly to the -DPLUGINS flag, but is more limited in its functionality (which does not matter, because there are not so many tools).
To add all non-experimental tools, you can use::
```
-DTOOLS=ALL
```

> Note that the behavior is different to PLUGINS which includes all PLUGINS if ALL is used.

To add all tools except of race, you can use:
```
-DTOOLS="ALL;-race"
```
To specify specific tools you can use, e.g.:
```
-DTOOLS=qt-gui;kdb
```

117.4.2.3 Bindings

Bindings are used in a like as PLUGINS. For example, to build all maintained bindings and exclude experimental bindings you can use:
```
-DBINDINGS=MAINTAINED;-EXPERIMENTAL
```
The SWIG executable may be specified with:
```
-DSWIG_EXECUTABLE=/usr/bin/swig3.0
```
If this option is not used, CMake will find the first occurrence of swig in your environment's path.
Some bindings provide different APIs (and not a different language), e.g:

- gsettings
- INTERCEPT with intercept_fs and intercept_env
- IO with io_uv

To not add such APIs, but only swig bindings and cpp, you can use:
```
-DBINDINGS="SWIG;cpp"
```
For a list of available bindings see binding's README.md.

117.4.2.4 <tt>CMAKE_BUILD_TYPE</tt>

Debug, Release or RelWithDebInfo See help bar at bottom of ccmake for that option or: https://www.cmake.org/Wiki/CMake_Useful_Variables

117.4.2.5 <tt>BUILD_SHARED</tt>, <tt>BUILD_FULL</tt> and <tt>BUILD_STATIC</tt>

BUILD_SHARED is the typical build you want to have on systems that support dlopen. It can be used for desktop builds, but also embedded systems as long as they support dlopen, for example, BUILD_SHARED is used on OpenWRT with musl. Using BUILD_SHARED every plugin is its own shared object.
BUILD_FULL links together all parts of Elektra as a single shared .so library. This is ideal if shared libraries are available, but you want to avoid dlopen. Some tests only work with BUILD_FULL, so you might turn it on to get full coverage.

BUILD_STATIC also links together all parts but as static .a library. It is only useful for systems without dlopen or if the overhead of dlopen needs to be avoided.
All three forms of builds can be intermixed freely.
For example, to enable shared and full build, but disable static build, one would use:

```
cmake -DBUILD_SHARED=ON -DBUILD_FULL=ON -DBUILD_STATIC=OFF ..
```

117.4.2.6 <tt>BUILD_DOCUMENTATION</tt>

Build documentation with doxygen (API) and ronn-ng (man pages).
If ronn-ng is not found, already compiled man pages will be used instead.

> Note: Turning off building the documentation, also turns off installing the documentation, see https://issues.libelektra.org/2522 Then no man pages are available.

117.4.2.7 <tt>BUILD_PDF</tt>

Build documentation with LaTeX.
See Documentation dependencies for the required dependencies.

117.4.2.8 Developer Options

As developer you should enable ENABLE_DEBUG and ENABLE_LOGGER:

- ENABLE_DEBUG:
  - enables assertions
  - adds RTLD_NODELETE so that debugger finds symbols even after dlclose
- ENABLE_LOGGER: enables logging By default no logging will take place, see CODING for how to get log messages.

Continue reading testing for more information about testing.

117.4.2.9 <tt>CMAKE_INSTALL_PREFIX</tt>

CMAKE_INSTALL_PREFIX defaults to /usr/local. So by default most files will be installed below /usr/local. Exceptions to this are files handled by INSTALL_SYSTEM_FILES.
Edit that cache entry to change that behavior. Also called system prefix within the documentation.
If you want to create a package afterwards it is ok to use paths that you can write to (e.g. -DCMAKE_INSTALL_PREFIX=/home/username/)

117.4.2.10 <tt>LIB_SUFFIX</tt>

Lets you install libraries into architecture specific folder. E.g. for 32/64 bit systems you might install libraries under lib64. Set LIB_SUFFIX to 64 to achieve exactly that. So the system library folder will be CMAKE_INSTALL_PREFIX/lib64 then.

117.4.2.11 <tt>TARGET_INCLUDE_FOLDER</tt>

By default include folders will be installed below CMAKE_INSTALL_PREFIX/include/elektra. This entry let you change the elektra. If the entry is empty, the include files will be installed directly to CMAKE_INSTALL_PREFIX/include.

117.4.2.12 <tt>TARGET_PLUGIN_FOLDER</tt>

Similar to above, but with the plugins. Default is: CMAKE_INSTALL_PREFIX/lib${LIB_SUFFIX}/elektra
It can be also left empty to install plugins next to other libraries.

117.4.2.13 <tt>GTEST_ROOT</tt>

This value specifies the root directory of a local copy of the Google Test framework.

- If it is empty (""), then the build system will download a copy of Google Test into the build directory.
- Otherwise the build system will search for the file CMakeLists.txt in the top level directory of GTEST_ROOT. If this file exists, then the build system will use the sources files at GTEST_ROOT to translate tests that use Google Test.

It can be provided as CMake or environment variable. If both options are provided the value passed via CMake takes precedence.
It is recommended that you browse through all the options using ccmake. Afterwards press c again (maybe multiple times until all variables are resolved) and then g to generate. Finally press e to exit.

117.4.2.14 <tt>INSTALL_BUILD_TOOLS</tt>

Specifies that the build tools, i.e. elektra-export-symbols and elektra-export-symbols are installed (by default off). Is needed for cross-compilation.

117.4.2.15 <tt>INSTALL_SYSTEM_FILES</tt>

Some of Elektra's targets require to be installed into specific folders in the file system hierarchy to work properly. This variable is disabled by default, since it requires the install target to have the rights to write into the corresponding folders. Set -DINSTALL_SYSTEM_FILES=ON, if you also want to install the files listed below.
If you do not have root rights you can also copy the files manually to your user folder.
Currently the installed system files are as following:

Module	Description	Install Path
bash-completion	bash tab auto completion file	completionsdir from pkg-config (1)
zsh-completion	zsh tab auto completion file	/usr/share/zsh/vendor-completions
GIR	introspection file for bindings	INTROSPECTION_GIRDIR from pkg-config
GSettings	GSettings backend module	GIO_MODULE_DIR from pkg-config

(1) Or /usr/share/bash-completion/completions as fallback.

117.4.2.16 <tt>ENABLE_OPTIMIZATIONS</tt>

In order to keep the binaries as small as possible this flag allows trading memory for speed.

117.5 Building

117.5.1 Without IDE

To build the source use:
```
make
```
You can pass:

- -j for parallel builds
- VERBOSE=1 to see the invocations of the compiler

Continue by reading INSTALL.

117.5.2 With CodeBlocks

You can build Elektra using Code::Blocks under Gentoo:
Precondition: Make sure you have a compiler, xml2 (for kdb tool) and xsl (see later) installed. cmake configure will help you with that, it will make sure you don't forget something essential.

For Most Linux system all you have to do is open up a console and
```
mkdir build
cd build
cmake .. -G 'CodeBlocks - Unix Makefiles'
make package
```
Note 1: You can use other editor if you like just type cmake at the console to get a list of option you can pass to cmake as long as well as a list of what code editor project cmake can create.
Note 2: For Unix if you have nCurses install you can run `ccmake` to set important option after running cmake like to enable debug symbol.
Note 3: For Gentoo it's recommended to emerge sys-apps/lsb-release to name the package right even thou not required.

117.6 Maintainer's Guide

117.6.1 Multiarch

On multiarch (multiple architectures installed in one system), you need to set `LIB_SUFFIX`. For example, if you want to have the binaries in `lib64` and `lib32`, you would use for the libraries to be installed in `${CMAKE_INSTALL_PREFIX}/lib64`:
```
-DLIB_SUFFIX="64"
```
If there is a directory for different architectures, simply prepend an `/`. For example, for Debian:
```
-DLIB_SUFFIX="/$(DEB_HOST_MULTIARCH)"
```

117.6.2 <tt>RPATH</tt>

By default Elektra uses `RPATH` to hide its plugins. This makes it obvious that external applications should *not* link against plugins. Instead every application should use the `elektraModulesLoad()` API to load Elektra's modules.
The folder where the plugins are located is a subdirectory of where the libraries are installed. The name of the subdirectory can be specified using `TARGET_PLUGIN_FOLDER` and is `elektra` by default. You might want to encode Elektra's `SOVERSION` into the folders name, if you want different major versions of Elektra be co-installable.
Elektra's use case for `RPATH` is considered acceptable, so we recommend using it because:

- plugins do not clutter the library folder nor the `ld.so.cache`
- it works well with multiarch (`LIB_SUFFIX` is also honored for plugins)
- which plugins are used should be decided at mount-time and be globally available in the same way for every application. `RPATH` supports exactly that because it even overrides `LD_LIBRARY_PATH`.

Unfortunately, there are also drawbacks:

- it makes Elektra non-relocatable (`RPATH` is decided at compile-time, so you cannot simply move Elektra's installations within the file system (e.g. from `/usr/local` to `/usr`)
- it requires modern `ld.so` implementations that honor `RPATH` from libraries. This is the case for most `libc` implementations including Linux and macOS, but not for, e.g., `musl`.

If you want Elektra to *not* use `RPATH`, you can add:
```
-DTARGET_PLUGIN_FOLDER="" -DCMAKE_SKIP_INSTALL_RPATH=ON
```
Then all plugins are directly installed to the library directory and loaded like other libraries (in any of `ld.so` paths). Alternatively, which gives you the advantage not to clutter the main library path, is to add the plugin folder in `/etc/ld.so.conf.d/elektra`. Note that it still allows applications to link against plugins.

117.7 Troubleshooting

117.7.1 Dependencies not Available for Cent OS

Please enable EPEL https://fedoraproject.org/wiki/EPEL
```
# Install EPEL for RHEL 7
curl -o epel-release-7-8.noarch.rpm \
  https://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-8.noarch.rpm
sudo rpm -ivh epel-release-7-8.noarch.rpm
```

```
sudo yum update
```

For Bindings swig3 is recommended. swig2 only works on some distributions. E.g., for Debian Jessie the bindings will crash.

At time of writing, no swig 3 was available, not even in EPEL. Thus you need to install swig3 manually:

```
curl https://codeload.github.com/swig/swig/tar.gz/rel-3.0.10 | tar xz
cd swig-rel-3.0.10 && ./autogen.sh && ./configure && make
sudo make install
cd ..
```

Also, no ronn-ng was available, thus you need to do:

```
gem install ronn-ng -v 0.10.1.pre1
```

117.7.2 Cross Compiling

In Elektra cross compiling needs two steps. If you get errors like `elektra-export-errors_EXE_LOC` not found, go on reading.

In the first step, you need to compile Elektra for the host architecture and install the build tools:

```
cmake -DINSTALL_BUILD_TOOLS=ON \
      -DCMAKE_PREFIX_PATH=$(STAGING_DIR_HOST) \
      ..
cmake --build . -- -j5
cmake --build . --target install
```

Where must be a directory to be found in the later build process. In particular, `/bin` must be in a directory found by a later `find_program`.

Then you need to compile Elektra again, but for the target architecture. Now, the build tools such as `elektra-export-errors` should be found in the `/bin` where they were installed before.

For reference, you can look into the `OpenWRT Elektra Makefile` and the `CMake in OpenWRT`.

117.8 See Also

- INSTALL
- TESTING
- BUILDSERVER

Chapter 118

review.md

On potential changes (including extensions) of the API/ABI please check following list:

- [ ] use case for API exists
- [ ] API design review template is created by author of API change
- [ ] **Before** starting to implement the change: decisions for API is in step `decided`
- [ ] full Doxygen documentation (all parameters, full design-by-contract, brief, long, examples, etc.)
- [ ] still ABI/API forward-compatible (It is okay to break backward-compatible to add new symbols)
- [ ] symbol versioning done correctly (./doc/dev/symbol-versioning.md)
- [ ] all Bindings written & tested (only if bindings exist for the module)
- [ ] full Testcoverage
- [ ] change is mentioned in the release notes, section `Compatibility`
- [ ] API design is reviewed by someone else, too
- [ ] decisions for API is in step `implemented`

Chapter 119

kdbClose

- start = 2021-03-09 01:55
- end = 2021-03-09 02:10
- moderator = Stefan Hanreich `stefanhani@gmail.com`

119.1 Signature

```
int kdbClose(KDB *handle, Key *errorKey)
```

119.2 Checklist

119.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] add possible error messages `errorKey` could contain
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] move below description
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] add `kdbOpen()`

119.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

119.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

119.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

119.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

119.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

119.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

119.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] line 587
  - [ ] return value never gets checked
- [ ] All possible error states are covered by tests
  - [ ] add test for `handle == NULl`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

119.3 Summary

- [ ] What happens if `errorKey` is `NULL`?

119.4 Other Issues discovered (unrelated to function)

- [ ] `ksRenameKeys()` in KDB-Module?

Chapter 120

kdbGet

- start = 2021-03-20 19:10
- end = 2021-03-20 19:25
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

120.1 Signature

```
int kdbGet(KDB *handle, KeySet *returned, Key *parentKey)
```

120.2 Checklist

120.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Atomic operation to retrieve Keys from a KDB'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] explain in more detail which keys are included (`keyIsBelow()`)
  - [ ] move above description out of `@param`
- [ ] Description of functions reads nicely
  - [ ] remove indentation from first paragraph in Optimization
- [ ] `@pre`
  - [ ] move below description
  - [ ] KeySet is not named `returned` anymore

- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] `keyIsBelow()` for rules on parents / children

120.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

120.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

120.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [ ] Wherever possible, function parameters should be `const`
  - [ ] handle
  - [ ] parentKey
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

120.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

120.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

120.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

120.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Lines 1185:1188
  - [ ] Lines 1218:1220
  - [ ] Lines 1225:1227
  - [ ] Lines 1262:1288
  - [ ] Lines 1298:1310
  - [ ] Lines 1345:1347
  - [ ] Line 1205
  - [ ] Line 1234
  - [ ] Line 1368
  - [ ] Line 1403
  - [ ] add `kdbGet` specific tests
- [ ] All possible error states are covered by tests
  - [ ] parentKey namespace is `KEY_NS_META`
  - [ ] parentKey namespace is `KEY_NS_NONE`
  - [ ] `ks == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

120.3 Summary

120.4 Other Issues discovered (unrelated to function)

Chapter 121

kdbOpen

- start = 2021-03-20 18:55
- end = 2021-03-20 19:10
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

121.1 Signature

```
KDB * kdbOpen(Key *errorKey)
```

121.2 Checklist

121.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] how do errors end up in `errorKey`?
  - [ ] what kind of errors end up in `errorKey`?
- [ ] Description of functions reads nicely
  - [ ] monospace 'system:/elektra/mountpoints'
  - [ ] move 'You must always call...' one paragraph up
- [ ] `@pre`
  - [ ] remove first (duplicated)
  - [ ] move below description

- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter
  - [ ] explain format of contract
- [ ] `@return` / `@retval`
  - [ ] specify failure conditions
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split into two lines
  - [ ] remove `kdbGet`?

121.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

121.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

121.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

121.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

121.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

121.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

121.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Lines 433:443
  - [ ] Lines 451:470
  - [ ] Line 478
  - [ ] Lines 508:515
  - [ ] Lines 531:539
  - [ ] return value never checked in tests
- [ ] All possible error states are covered by tests
  - [ ] `errorKey == NULl`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

121.3 Summary

121.4 Other Issues discovered (unrelated to function)

`ksRenameKeys` and `elektraOpenBootstrap` in API Documentation

Chapter 122

kdbSet

- start = 2021-03-20 19:10
- end = 2021-03-20 19:30
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

122.1 Signature

```
int kdbSet(KDB *handle, KeySet *returned, Key *parentKey)
```

122.2 Checklist

122.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Atomic operation to set Keys for a KDB'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] move description out of `@param`
  - [ ] add info what happens if keys are read-only
- [ ] Description of functions reads nicely
  - [ ] 'give a hint' -> 'specify'
- [ ] `@pre`
  - [ ] move below description
  - [ ] KeySet is not named `returned` anymore

- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter
  - [ ] move parentKey description to description text
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

122.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

122.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

122.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

122.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

122.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

122.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

122.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Lines 1737:1740
  - [ ] Lines 1745:1748
  - [ ] Lines 1753:1756
  - [ ] Lines 1761:1764
  - [ ] Lines 1796:1798
  - [ ] Lines 1806:1809
  - [ ] Line 1906
- [ ] All possible error states are covered by tests
  - [ ] `handle == NULL`
  - [ ] `ks == NULL`
  - [ ] `parentKey == NULL`
  - [ ] read-only parentKey
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

122.3 Summary

122.4 Other Issues discovered (unrelated to function)

Chapter 123

keyAddBaseName

- start = 2021-03-04 21:35
- end = 2021-03-04 22:00
- reviewer = Stefan Hanreich stefanhani@gmail.com

123.1 Signature

```
ssize_t keyAddBaseName(Key *key,const char *baseName)
```

123.2 Checklist

123.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Adds -> Appends
- [ ] Simple example or snippet how to use the function
  - [ ] move below brief description
- [ ] Longer description of function containing common use cases
  - [ ] add notice for read-only keys
- [ ] Description of functions reads nicely
  - [ ] Add comma after "When baseName is 0"
  - [ ] 0 -> null pointer
- [ ] `@pre`
  - [ ] add
- [ ] `@post`

    - [ ] add

- [ ] `@invariant`

    - [ ] add

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

    - [ ] add read-only case

- [ ] `@since`

    - [ ] add

- [x] `@ingroup`
- [ ] `@see`

    - [ ] move to bottom

123.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

123.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

123.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

123.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swapped with `keySetBaseName`
- [x] No functions with similar purpose exist

123.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

123.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

123.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

123.3 Summary

123.4 Other Issues discovered (unrelated to function)

Chapter 124

keyAddName

- start = 2021-02-26 12:25
- end = 2021-02-26 18:10
- moderator = Stefan Hanreich stefanhani@gmail.com
- reviewer = Michael Tucek michael@tucek.eu

124.1 Signature

```
ssize_t keyAddName(Key *key, const char *addName)
```

124.2 Checklist

124.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Add an already escaped key name part to the key name
- [ ] Simple example or snippet how to use the function
  - [ ] add very simple example, with one simple key name part without dots
- [ ] Longer description of function containing common use cases
  - [ ] add common use cases (append to a key name,..)
- [ ] Description of functions reads nicely
  - [ ] move examples to bottom
  - [ ] passed name below brief description
  - [ ] replace list with reference to canonicalization docs
  - [ ] 'key name' rules no space before dot

- [ ] `@pre`
  - [ ] move above `@param`
  - [ ] writeable key
  - [ ] newName must be a valid name (reference to key name rules)
  - [ ] move "It is not allowed to" precondition
- [ ] `@post`
  - [ ] newName has been added to key
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter
  - [ ] move below `@pre`
  - [ ] key - pointer to Key object
  - [ ] newName - name part to append to `key`'s name
- [ ] `@return` / `@retval`
  - [ ] move below `@param`
  - [ ] move `@retval` into list
  - [ ] make error cases differentiable (key errors, name errors)?
- [ ] `@since`
  - [ ] move lower
- [x] `@ingroup`
- [ ] `@see`
  - [ ] `keyAddBaseName()`
  - [ ] `keySetName()`
  - [ ] `keySetBaseName()`

124.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous
  - [ ] Add -> Append
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] newName -> nameParts

124.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

124.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

124.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] order is different
- [ ] No functions with similar purpose exist
  - [ ] `keyAddName()` merge with `keySetName()`

124.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

124.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

124.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] check if we can remove uncovered code
  - [x] Line 542
- [ ] All possible error states are covered by tests
  - [x] add test case for read-only key
  - [x] test invalid name
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

124.3 Summary

124.4 Other Issues discovered (unrelated to function)

- What is a valid Key exactly? Or does it just mean a valid pointer?
- Different Return Vlaues for different errors
- Set Doxygen Options to sort after header file

Chapter 125

keyBaseName

- start = 2021-03-03 00:10
- end = 2021-03-03 00:25
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

125.1 Signature

```
const char *keyBaseName(const Key *key)
```

125.2 Checklist

125.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add a simple example below the description
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [x] `@return` / `@retval`

- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split first line

125.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

125.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

125.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

125.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

125.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

125.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

125.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

125.3 Summary

125.4 Other Issues discovered (unrelated to function)

Chapter 126

keyClear

- start = 2021-02-21 22:40
- end = 2021-02-21 23:05
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

126.1 Signature

```
int keyClear(Key *key)
```

126.2 Checklist

126.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] remove first line, second line could be improved a bit
    (what is meant with internal data, exactly? is the name included?)
- [x] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
  - [ ] be more explicit which data is actually included
- [ ] Description of functions reads nicely
  - [ ] very short sentences
- [ ] `@pre`
  - [x]
    Precondition

    key is a valid key
- [ ] `@post`
  - [x]

Postcondition

key contains not internal data

- [ ] `@invariant`
  - [x]
    Invariant

    key stays a valid key
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
  - [ ] mark default case with `@return`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] maybe `keyDel()`

126.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

126.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

126.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

126.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swap with `keyDel()`
- [ ] No functions with similar purpose exist
  - [ ] new keyCopy can also clear a key, maybe make `keyClear` an alias?

126.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

126.2.6 Extensibility

- [ ] Function is easily extensible, e.g., with flags
  - [ ] could add flags for clearing parts of the key like `keyCopy()`
- [x] Documentation does not impose limits, that would hinder further extensions

126.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

126.3 Summary

126.4 Other Issues discovered (unrelated to function)

Chapter 127

keyCmp

- start = 2021-03-07 17:40
- end = 2021-03-07 17:55
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

127.1 Signature

```
int keyCmp(const Key *k1, const Key *k2)
```

127.2 Checklist

127.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move last example below brief explanation
- [ ] Longer description of function containing common use cases
  - [ ] add description about behavior of namespace comparison
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] remove `@return` in favor of `@retval`'s?
  - [ ] add `@retval` for k1 > k2
  - [ ] add `@retval` for k1 == k2
  - [ ] add `@retval` for k1 < k2
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

127.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] Define Cmp or write it out?
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [ ] Parameter names should neither be too long, nor too short
  - [ ] k1 -> key1 / this - [ ] k2 -> key2 / that
- [x] Parameter names should be clear and unambiguous

127.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

127.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

127.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

127.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

127.2.6 Extensibility

- [ ] Function is easily extensible, e.g., with flags
  - [ ] add flags for comparing specific parts of the key
- [ ] Documentation does not impose limits, that would hinder further extensions

127.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] L517
  - [x] L558
- [ ] All possible error states are covered by tests
  - [x] test key with same name, but different owners
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

127.3 Summary

move implementation out of keyset.c ?

127.4 Other Issues discovered (unrelated to function)

Chapter 128

keyCopy

invariants?

- start = 2021-02-13 12:00
- end = 2021-02-13 13:10
- moderator = Stefan Hanreich `stefanhani@gmail.com`
- reviewer = Markus Raab `markus@libelektra.org`

128.1 Signature

```
int keyCopy(Key *dest, const Key *source);
```

128.2 Checklist

128.2.1 Doxygen

- [ ] first line explains briefly what the function does
  - [x] Clear -> clear
  - [x] do not instantly reference `keyDup`
- [ ] `@see`
  - [x] add `@see` for `keyDup`
- [ ] `@since` (optional: for after 1.0.0)
  - [x] add `@since` 0.9.5
- [x] `@ingroup`
- [ ] `@retval`
  - [x] use retval instead of return (see snippet below)
- [ ] good example or snippet how to use the function
  - [ ] move example to `keyDup`

- [ ] simple examples go first
  - [x] move simple example to the top
- [ ] Precondition
  - [x] add precondition (valid key values)
  - [x]
    Precondition

    dest must be a valid Key (created with keyNew)
  - [x]
    Precondition

    source must be a valid Key or NULL
- [ ] Postcondition
  - [x] add postcondition
  - [x]
    Postcondition

    Value is written to key dest
- [ ] Invariant
  - [x] add invariant
  - [x] Key name stays valid (document at struct)
  - [x] Key name stays valid until delete
- [ ] `@param` for every parameter
  - [x] better describe flags (|)

invariants?

```
@retval dest
@retval NULL in case of error
```

128.2.2 Naming

- [x] Abbreviations used in function names must be defined in the Glossary src/libs/elektra/symbols.map
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [x] add dest to glossary - maybe rename dest to destination?
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

128.2.3 Compatibility

- [x] ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

128.2.4 Parameter & Return Types

- Functions should return the most specific type possible
- Functions should require the most general type possible
- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

128.2.5 Structural Clarity

- [ ] Functions should do exactly one thing
  - [x] remove keySetName keySetRaw ksClear functionality from keyCopy?
- [x] Function name has the appropriate prefix
- [x] Signature in kdb.h.in has same order as Doxygen docu
- [ ] No functions with similar purpose exist
  - [x] keyCopyAllMeta

128.2.6 Memory Management

- [x] Memory Management should be handled by the function wherever possible

128.2.7 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [x] KEY_CP_ALL equivalent to KEY_CP_NAME | KEY_CP_VALUE | KEY_CP_META

128.2.8 Tests

- [ ] Added functions are fully covered by tests https://doc.libelektra.org/coverage/master/debian-bust
c.gcov.html
- [ ] All possible error states are covered by tests and documented
- [ ] All possible enum values are covered by tests
  - [x] use flags other than KEY_CP_ALL
- [ ] No inconsistencies between tests and documentation
- [ ] Functions should have no side effects (idempotency)

128.3 Summary

128.4 Other Issues discovered (unrelated to function)

Chapter 129

keyCopyAllMeta

- start = 2021-02-27 17:45
- end = 2021-02-27 18:00
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

129.1 Signature

```
int keyCopyAllMeta(Key *dest, const Key *source)
```

129.2 Checklist

129.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move simple example below `@brief`
- [ ] Longer description of function containing common use cases
  - [ ] add info that if `dest` is read-only nothing will happen
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add 'dest is a valid Key'
  - [x] add 'source is a valid Key'
- [x] `@post`
- [ ] `@invariant`
  - [x] add 'dest stays a valid Key'
  - [x] add 'source stays a valid Key'

- [ ] `@param` for every parameter
  - [ ] move in front of return values
- [ ] `@return` / `@retval`
  - [ ] if **meta-information** was successfully copied
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] add `keyCopyMeta()`

129.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

129.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

129.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

129.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

129.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

129.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

129.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for dest NULL
  - [x] add test for source NULL
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

129.3 Summary

129.4 Other Issues discovered (unrelated to function)

Chapter 130

keyCopyMeta

- start = 2021-02-27 17:30
- end = 2021-02-27 17:45
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

130.1 Signature

```
int keyCopyMeta(Key *dest, const Key *source, const char *metaName)
```

130.2 Checklist

130.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] of metadata -> of the metaData with name `metaName`
- [ ] Simple example or snippet how to use the function
  - [ ] move below `@brief`
- [ ] Longer description of function containing common use cases
  - [ ] describe the functionality more explicitly where meta-information gets deleted when `source`'s `metaName` is not set in `dest`
  - [ ] add information that it does nothing on read-only keys
- [ ] Description of functions reads nicely
  - [ ] it will take -> will take
- [ ] `@pre`
  - [x] 'dest is a valid Key'
  - [x] 'source is a valid Key'

- [x] `@post`
- [ ] `@invariant`
  - [x] 'dest stays a valid Key'
  - [x] 'source stays a valid Key'
- [ ] `@param` for every parameter
  - [ ] too -> to
  - [ ] name of the metadata -> name of the key in the metadata
- [ ] `@return` / `@retval`
  - [ ] move `1` to `@return`
  - [ ] add `-1` if meta-information is read-only
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] add `keyCopyAllMeta()`

130.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

130.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

130.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

130.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

130.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

130.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

130.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Lines 294:298
- [ ] All possible error states are covered by tests
  - [ ] add test case for clearing meta-information key
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

130.3 Summary

130.4 Other Issues discovered (unrelated to function)

Chapter 131

keyCurrentMeta

- start = 2021-02-27 17:20
- end = 2021-02-27 17:30
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

131.1 Signature

```
const Key *keyCurrentMeta(const Key *key)
```

131.2 Checklist

131.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] returns the Key of the meta-information the internal iterator points at
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] The pointer -> The returned pointer
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`

  - [x] add

- [ ] `@param` for every parameter

  - [ ] Key to get the current meta-information from

- [ ] `@return` / `@retval`

  - [ ] a pointer to the current meta-information's Key

- [ ] `@since`

  - [ ] add

- [x] `@ingroup`

- [ ] `@see`

  - [ ] split first line into two lines

131.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

131.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

131.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

131.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

131.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

131.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] pointer is NULL after `ksRewind()`

131.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add check for null pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

131.3 Summary

131.4 Other Issues discovered (unrelated to function)

Chapter 132

keyDecRef

- start = 2021-02-21 21:10
- end = 2021-02-21 21:25
- reviewer = Stefan Hanreich stefanhani@gmail.com

132.1 Signature

```
ssize_t keyDecRef(Key *key)
```

132.2 Checklist

132.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] change viability to reference counter
- [ ] Simple example or snippet how to use the function
  - [ ] add example
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] description seems a bit unclear when reading
- [ ] `@pre`
  - [x]
    Precondition

    key is a valid key
- [ ] `@post`
  - [x]

Postcondition

reference counter of the key is decremented by one

- [ ] `@invariant`
  - [x]
    Invariant

    key stays a valid key
- [x] `@param` for every parameter
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - split to multiple lines

132.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - add dec to glossary
  - add ref to glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

132.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

132.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

132.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

132.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

132.2.6 Extensibility

- Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

132.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] test decrementing key with reference counter = 0
- [ ] All possible error states are covered by tests
  - [x] test decrementing key with reference counter = 0
  - [x] test null pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

132.3 Summary

132.4 Other Issues discovered (unrelated to function)

Chapter 133

keyDel

- start = 2021-02-14 03:55
- end = 2021-02-14 04:25
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

133.1 Signature

```
int keyDel(Key *key)
```

133.2 Checklist

133.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add example
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x]
    Precondition

    Key must not be null
- [ ] `@post`
  - [x]
    Postcondition

    Key memory has been freed
- [ ] `

Invariant

  - [x] add (tbd)

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
  - [ ] `0` when the key was freed and no references are held in keysets
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

133.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

133.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

133.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [ ] Wherever possible, return types should be `const`
  - [ ] could maybe change this to const
- [x] Functions should have the least amount of parameters feasible

133.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swap keyClear and keyDel
- [x] No functions with similar purpose exist

133.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

133.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] TBD cannot change behavior of referenced keys

133.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

133.3 Summary

133.4 Other Issues discovered (unrelated to function)

Chapter 134

keyDup

- start = 2021-02-20 18:10
- end = 2021-02-20 18:35
- moderator = Stefan Hanreich `stefanhani@gmail.com`

134.1 Signature

```
Key *keyDup(const Key *source)
```

134.2 Checklist

134.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] could be a bit more specific
- [ ] Simple example or snippet how to use the function
  - [ ] remove the wrapping function as it only distracts
- [ ] Longer description of function containing common use cases
  - [ ] be a bit more explicit
- [ ] Description of functions reads nicely
  - [ ] "Like for a new key" could be formulated better
- [ ] `@pre`
  - [ ]

Precondition

source must be a valid key

- [ ] `@post`
  - [ ]
    Postcondition

    returns a full copy of the key including all info (metadata, ..)
- [ ] `@invariant`
  - [ ]
    Invariant

    source stays a valid key
- [x] `@param` for every parameter
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] `keyCopy()`

134.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] add dup to glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

134.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

134.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

134.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

134.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

134.2.6 Extensibility

- [ ] Function is easily extensible, e.g., with flags
  - [ ] could add flags similar to `keyCopy()`
- [x] Documentation does not impose limits, that would hinder further extensions

134.2.7 Tests

- [x] Function code is fully covered by tests
  - Error state is hard to check, cause it only appears on null pointers which gets checked in the function itself already
- [x] All possible error states are covered by tests
  - see above
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

134.3 Summary

134.4 Other Issues discovered (unrelated to function)

Chapter 135

keyGetBaseName

- start = 2021-03-04 21:05
- end = 2021-03-04 21:35
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

135.1 Signature

```
ssize_t keyGetBaseName(const Key *key, char *returned, size_t maxSize)
```

135.2 Checklist

135.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] add description of what happens when maxSize < baseNameSize
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split first line
  - [ ] maybe remove second line?

135.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] change to returnedBaseName in order to be more in line with other functions?

135.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

135.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

135.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

135.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

135.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

135.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Lines 1147:1148
- [ ] All possible error states are covered by tests
  - [x] Lines 1147:1148
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

135.3 Summary

add more tests for special cases? (spaces, escaped slashes,...)

135.4 Other Issues discovered (unrelated to function)

Chapter 136

keyGetBaseNameSize

- start = 2021-03-03 00:30
- end = 2021-03-03 00:45
- moderator = Stefan Hanreich `stefanhani@gmail.com`

136.1 Signature

```
ssize_t keyGetBaseNameSize(const Key *key)
```

136.2 Checklist

136.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - add including `\0`
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] does -> do
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`

    - [ ] add

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
    - [ ] add special case `""`
    - [ ] add special case `key == NULL`
- [ ] `@since`
    - [ ] add
- [x] `@ingroup`
- [ ] `@see`
    - [ ] split into multiple lines
    - [ ] remove `keyGetName()`
    - [ ] remove `keySetName()`

136.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

136.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

136.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

136.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

136.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

136.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

136.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add NULL pointer test case
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

136.3 Summary

136.4 Other Issues discovered (unrelated to function)

Chapter 137

keyGetBinary

- start = 2021-03-07 17:15
- end = 2021-03-07 17:30
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

137.1 Signature

```
ssize_t keyGetBinary(const Key *key, void *returnedBinary, size_t maxSize)
```

137.2 Checklist

137.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] as binary data
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [ ] Longer description of function containing common use cases
  - [ ] merge reference to string functions with `@see`
  - [ ] how exactly is empty defined for binary?
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add

- [ ] `@invariant`
  - [x] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
  - [ ] merge lines for maxSize errors
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split first line

137.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

137.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

137.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

137.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

137.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

137.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

137.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] L417
  - [x] L422
  - [x] L427
- [ ] All possible error states are covered by tests
  - [x] add test for key with string value
  - [x] L422 seems to be in the tests - check
  - [x] L427 seems to be in the tests - check
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

137.3 Summary

"Clear" binary array on error, to mirror behavior from string? What to fill it with, though? Zeroes?

137.4 Other Issues discovered (unrelated to function)

Chapter 138

keyGetMeta

- start = 2021-02-26 14:20
- end = 2021-02-26 14:45
- moderator = Stefan Hanreich `stefanhani@gmail.com`

138.1 Signature

```
const Key *keyGetMeta(const Key *key, const char* metaName)
```

138.2 Checklist

138.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] returns the Key of a meta...
  - [ ] name -> metaName
- [ ] Simple example or snippet how to use the function
  - [ ] move up
  - [ ] assign to variable
  - [ ] add comment about value in variable
  - [ ] remove function, if
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] Remove 'You are not allowed....'
- [ ] `@pre`

    - [x] add
- [ ] `@post`
    - [x] add
- [ ] `@invariant`
    - [x] add
- [ ] `@param` for every parameter
    - [ ] split to two lines
    - [ ] add dot after first sentence
- [ ] `@return` / `@retval`
    - [ ] move `@return` up
    - [ ] `@return`: value -> Key
    - [ ] `@retval` 0 metaName invalid
- [ ] `@since`
    - [ ] add
- [x] `@ingroup`
- [ ] `@see`
    - [ ] add `keyMeta()`

138.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

138.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

138.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

138.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist
  Similar thing could be achieved via
  `ksLookupByName (keyMeta (key), metaName, 0)`

138.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

138.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

138.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] see below
- [ ] All possible error states are covered by tests
  - [x] key is 0
  - [x] metaName is 0
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

138.3 Summary

138.4 Other Issues discovered (unrelated to function)

Chapter 139

keyGetName

- start = 2021-02-22 13:40
- end = 2021-02-22 14:20
- moderator = Stefan Hanreich stefanhani@gmail.com
- reviewer = Philipp Gackstatter

139.1 Signature

```
ssize_t keyGetName(const Key *key, char *returnedName, size_t maxSize)
```

139.2 Checklist

139.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] owner still recent?
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [ ] Longer description of function containing common use cases
  - [ ] maxSize special case has very long description (>50% of body)
  - [ ] maxSize special case move description to pre / post / return values
  - [ ] mention explicitly, that user has to manage their buffer
  - [ ] move special case 'not enough space' to return values / pre / post
- [ ] Description of functions reads nicely
  - [ ] remove / define abbreviated (make it more explicit)

  - [ ] "Writes keyName of `key` into the provided buffer `returnedName` if it is shorter than `maxSize`"

- [ ] `@pre`
  - [ ] `key` is a valid Key
  - [ ] `returnedName` is a pre-allocated buffer
  - [ ] maxSize is the size of `returnedName`
- [ ] `@post`
  - [ ] `returnedName` contains the name of the Key
  - [ ] `key` is not modified
- [ ] `@invariant`
  - [ ] `key` is not modified
- [ ] `@param` for every parameter
  - [ ] `returnedName`: pre-allocated buffer to write the key name into
- [ ] `@return` / `@retval`
  - [ ] `1`: when only a null-terminator was written
  - [ ] `-1`: then -> than
  - [ ] `-1`: standardize / change 'keyname' / 'key name' or 'Key name'
  - [ ] `-1` check if null pointer case is even still applicable
  - [ ] `-1` split into two lines: maxSize / NULL pointer
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] keyName
  - [ ] keyGetUnescapedName
  - [ ] keyGetBaseName
  - [ ] keyGetNamespace

139.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous
  - [ ] `keyName` vs `keyGetName()`
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] max - maybe just rename to `bufferSize`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `returnedName` -> `name` / `nameBuffer`
- [ ] Parameter names should be clear and unambiguous
  - [ ] `maxSize` -> `bufferSize`

139.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

139.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

139.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - wrong order
- [x] No functions with similar purpose exist

139.2.5 Memory Management

- [ ] Memory Management should be handled by the function wherever possible
  - [ ] pass variable and return size instead
  - [ ] remove `maxSize` param

139.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] nothing is written on maxSize < nameSize (would be fixed with memory management suggestion from above)

139.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Lines 365 / 366
- [ ] All possible error states are covered by tests
  - [x] test empty string
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

139.3 Summary

Different error codes for different errors

139.4 Other Issues discovered (unrelated to function)

Chapter 140

keyGetNameSize

- start = 2021-02-22 13:10
- end = 2021-02-22 13:40
- moderator = Stefan Hanreich `stefanhani@gmail.com`
- reviewer = Philipp Gackstatter

140.1 Signature

```
ssize_t keyGetNameSize(const Key *key)
```

140.2 Checklist

140.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] only describes return value
  - [ ] null-termination
  - [ ] mit return mergen
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] better describe common use case
  - [ ] example for error case
  - [ ] example for empty string
- [x] Description of functions reads nicely
- [ ] `@pre`

    - [ ] add 'Key should be valid'
- [ ] `@post`
    - [ ] add 'key doesn't get modified'
- [ ] `@invariant`
    - [ ] add 'key doesn't get modified'
- [ ] `@param` for every parameter
    - [ ] 'key object to get the size of the name from'
    - [ ] make consistent
- [ ] `@return` / `@retval`
    - [ ] without owner?
- [ ] `@see`
    - [ ] add parentheses to keyGetUnescapedNameSize

140.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

140.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

140.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

140.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

140.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

140.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

140.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test case for NULL pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

140.3 Summary

try splitting ifs to two lines, for better coverage reports

140.4 Other Issues discovered (unrelated to function)

Chapter 141

keyGetNamespace

- start = 2021-03-04 22:40
- end = 2021-03-04 22:50
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

141.1 Signature

```
elektraNamespace keyGetNamespace(Key const* key)
```

141.2 Checklist

141.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] add 'returns the `elektraNamespace` for a given key'
- [x] Simple example or snippet how to use the function
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

  - [ ] add `@retval` for key == `NULL` ?

- [ ] `@since`

  - [ ] add

- [x] `@ingroup`
- [ ] `@see`

  - [ ] add `keySetNamespace()`

141.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

141.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

141.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

141.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

141.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

141.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

141.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- [ ] All possible enum values are covered by tests
  - [x] `KEY_NS_META`
  - [x] `KEY_NS_PROC`
  - [x] `KEY_NS_DIR`
  - [x] `KEY_NS_DEFAULT`
- [x] No inconsistencies between tests and documentation

141.3 Summary

141.4 Other Issues discovered (unrelated to function)

Chapter 142

keyGetRef

- start = 2021-02-21 21:25
- end = 2021-02-21 21:35
- moderator = Stefan Hanreich `stefanhani@gmail.com`

142.1 Signature

```
ssize_t keyGetRef(const Key *key)
```

142.2 Checklist

142.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add simple example for `keyGetRef()` base case without ks
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ]
    Precondition

    key is a valid key
- [ ] `@post`
  - [ ]
    Postcondition

    key stays unchanged
- [ ] `@invariant`
  - [ ]

Invariant

key stays a valid key

- [x] `@param` for every parameter
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - maybe add `keyDel()`

142.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] add Ref to the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

142.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

142.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

142.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

142.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

142.2.6 Extensibility

- Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

142.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] check for null pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

142.3 Summary

142.4 Other Issues discovered (unrelated to function)

Chapter 143

keyGetString

- start = 2021-03-06 16:25
- end = 2021-03-06 16:40
- moderator = Stefan Hanreich `stefanhani@gmail.com`

143.1 Signature

```
ssize_t keyGetString(const Key *key, char *returnedString, size_t maxSize)
```

143.2 Checklist

143.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Copies the string value of a Key into a provided buffer
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [ ] Longer description of function containing common use cases
  - [ ] move binary functions to `@see`
- [ ] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`

    - [x] add

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

    - [ ] merge maxSize errors?

- [ ] `@since`

    - [ ] add

- [x] `@ingroup`
- [ ] `@see`

    - [ ] split and add description

143.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

143.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

143.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

143.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

143.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

143.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

143.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 306
  - [x] Line 317
    (impossible to check - overflow)
- [ ] All possible error states are covered by tests
  - [x] check return value for binary strings
    (seems to already happen? tests/abi/testabi_key.c:1007 ff.)
  - [x] check return value for maxSize too small
    (seems to already happen? tests/abi/testabi_key.c:913)
  - [x] check return value for maxSize > SSIZE_MAX
    (impossible to check - overflow)
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

143.3 Summary

143.4 Other Issues discovered (unrelated to function)

- Find a different term for functions that get a pointer and functions that
- return in to a buffer?

Chapter 144

keyGetUnescapedNameSize

- start = 2021-03-02 22:00
- end = 2021-03-02 22:15
- moderator = Stefan Hanreich `stefanhani@gmail.com`

144.1 Signature

```
ssize_t keyGetUnescapedNameSize(const Key *key)
```

144.2 Checklist

144.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] return -> Returns the size of the unescaped name, including ...
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] add `@return` for base case
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] add `keyGetNameSize()`
  - [ ] add `keyGetUnescapedName()`

144.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

144.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

144.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

144.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] Order is mixed up, needs to be lower in Doxygen
- [x] No functions with similar purpose exist

144.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

144.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

144.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] no explicit calls in tests suite
- [ ] All possible error states are covered by tests
  - [x] no explicit calls in tests suite
- All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation
  - [x] no explicit calls in tests suite

144.3 Summary

144.4 Other Issues discovered (unrelated to function)

Chapter 145

keyGetValueSize

- start = 2021-02-26 16:40
- end = 2021-02-26 17:15
- moderator = Stefan Hanreich stefanhani@gmail.com
- reviewer = Robert Sowula

145.1 Signature

```
ssize_t keyGetValueSize(const Key *key)
```

145.2 Checklist

145.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move below brief description
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] If the value is binary, there...
  - [ ][..] for binary data, so 0 will be returned.
  - [ ] String types have it, so to there length will be added 1 to have enough space to store it. -> Strings are null-terminated, and the null terminator will be considered for the length.
- [ ] `@pre`
  - [x] add?
- [ ] `@post`
  - [x] add?

- [ ] `@invariant`
  - [x] add?
- [ ] `@param` for every parameter
  - [ ] key: key -> Key
- [ ] `@return` / `@retval`
  - [ ] 1: not binary -> string
  - [ ] 0: binary -> not string (?)
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] `keyGetString()`
  - [ ] `keyGetBinary()`

145.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

145.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

145.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

145.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swapped `with keyGetValueSize()`
- [x] No functions with similar purpose exist

145.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

145.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

145.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 249
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

145.3 Summary

145.4 Other Issues discovered (unrelated to function)

Chapter 146

keyIncRef

- start = 2021-02-21 20:50
- end = 2021-02-21 21:05
- reviewer = Stefan Hanreich stefanhani@gmail.com

146.1 Signature

```
ssize_t keyIncRef(Key *key)
```

146.2 Checklist

146.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] change viability to 'reference counter'
- [ ] Simple example or snippet how to use the function
  - [ ] add example
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x]
    Precondition

    key is a valid key
- [ ] `@post`
  - [x]

Postcondition

reference counter of the key is incremented by one

- [ ] `@invariant`
  - [x]
    Invariant

    key stays a valid key
- [x] `@param` for every parameter
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] add `keyGetRef()`
  - [ ] add `keyDecRef()`

146.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] add inc to glossary
  - [ ] add ref to glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

146.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

146.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

146.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

146.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

146.2.6 Extensibility

- Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions

146.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for null pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

146.3 Summary

146.4 Other Issues discovered (unrelated to function)

Chapter 147

keyIsBelow

- start = 2021-03-07 18:55
- end = 2021-03-07 19:10
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

147.1 Signature

```
int keyIsBelow(const Key *key, const Key *check)
```

147.2 Checklist

147.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] capitalize Key to make it distinct from parameter
- [ ] Simple example or snippet how to use the function
  - [ ] change example into real code
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add
- [ ] `@param` for every parameter

  - [ ] rework descriptions to be more clear

- [x] `@return` / `@retval`
- [ ] `@since`

  - [ ] add

- [x] `@ingroup`
- [ ] `@see`

  - [ ] split

147.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous

  - [ ] the name of `check` might be improved

147.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

147.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

147.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

147.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

147.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

147.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 214
  - [ ] Line 227
  - [x] Line 256
- [ ] All possible error states are covered by tests
  - [x] 214 seems to be tested
  - [x] 256 seems to be tested
  - [x] add test for cascading key vs root key
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

147.3 Summary

147.4 Other Issues discovered (unrelated to function)

Chapter 148

keyIsBelowOrSame

- start = 2021-01-23 18:10
- end = 2021-01-23 18:10
- moderator = Stefan Hanreich `stefanhani@gmail.com`

148.1 Signature

```
int keyIsBelowOrSame(const Key *key, const Key *check)
```

148.2 Checklist

148.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
- [ ] `@pre`
- [ ] `@post`
- [ ] `@invariant`
- [ ] `@param` for every parameter
- [ ] `@return` / `@retval`
- [ ] `@since`
- [ ] `@ingroup`
- [ ] `@see`

148.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
- [ ] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous

- [ ] Abbreviations used in parameter names must be defined in the Glossary
- [ ] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous

148.2.2 Compatibility

(only in PRs)

- [ ] Symbol versioning is correct for breaking changes
- [ ] ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

148.2.3 Parameter & Return Types

- [ ] Function parameters should use enum types instead of boolean types wherever sensible
- [ ] Wherever possible, function parameters should be `const`
- [ ] Wherever possible, return types should be `const`
- [ ] Functions should have the least amount of parameters feasible

148.2.4 Structural Clarity

- [ ] Functions should do exactly one thing
- [ ] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist

148.2.5 Memory Management

- [ ] Memory Management should be handled by the function wherever possible

148.2.6 Extensibility

- [ ] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions

148.2.7 Tests

- [ ] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
- [ ] All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation

148.3 Summary

148.4 Other Issues discovered (unrelated to function)

Chapter 149

keyIsBinary

- start = 2021-03-09 01:40
- end = 2021-03-09 01:55
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

149.1 Signature

```
int keyIsBinary(const Key *key)
```

149.2 Checklist

149.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'a key' -> 'the value of a key'
  - [ ] 'of binary type'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add

- [ ] `@param` for every parameter
  - [ ] move up
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split into two lines

149.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

149.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

149.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

149.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

149.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

149.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

149.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for `key == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

149.3 Summary

149.4 Other Issues discovered (unrelated to function)

Chapter 150

keyIsDirectlyBelow

- start = 2021-03-07 19:10
- end = 2021-03-07 19:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

150.1 Signature

```
int keyIsDirectlyBelow(const Key *key, const Key *check)
```

150.2 Checklist

150.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] split examples
  - [ ] turn examples into proper code
- [ ] Longer description of function containing common use cases
  - [ ] move description from examples to documentation body
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
  - [ ] 'check is below key' -> 'check is directly below key'
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split and add descriptions

150.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] the name of `check` might be improved

150.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

150.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

150.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swapped with `keyIsBelowOrSame()`
- [x] No functions with similar purpose exist

150.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

150.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

150.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 293
  - [x] Line 326
- [ ] All possible error states are covered by tests
  - [x] Line 293 seems to be checked
  - [x] Line 326 seems to be checked
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

150.3 Summary

150.4 Other Issues discovered (unrelated to function)

- [ ] `keyIsBelowOrSame` is not contained in Doxygen
- [ ] maybe merge `keyBelowFamily` into one function with flags

Chapter 151

keyIsLocked

- start = 2021-02-21 22:15
- end = 2021-02-21 22:40
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

151.1 Signature

```
int keyIsLocked (const Key * key, elektraLockFlags what)
```

151.2 Checklist

151.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] add better description or reference to `keyLocked()`
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] `key` is a valid key
  - [x] `what` contains valid elektraLockFlags
- [ ] `@post`
  - [x] `key` stays unchanged
- [ ] `@invariant`
  - [x] `key` stays a valid key

- [ ] `@param` for every parameter
  - [ ] add
- [ ] `@return` / `@retval`
  - [ ] change default case to `@return`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] move to bottom

151.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] rename what to flags?

151.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

151.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

151.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] move below `keyGetRef()`-family of functions
- [x] No functions with similar purpose exist

151.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

151.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

151.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] test for null pointer return value
- [x] All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

151.3 Summary

151.4 Other Issues discovered (unrelated to function)

Chapter 152

keyIsString

- start = 2021-03-07 17:55
- end = 2021-03-07 18:05
- reviewer = Stefan Hanreich stefanhani@gmail.com

152.1 Signature

```
int keyIsString(const Key *key)
```

152.2 Checklist

152.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Check if the value of a key is of string type'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] rework last sentence, especially last part (split into two sentences?)
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`

  - [x] add

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

  - [ ] replace 'it' with 'value of key' / 'key's value'

- [ ] `@since`

  - [ ] add

- [x] `@ingroup`
- [ ] `@see`

  - [ ] split into two lines

152.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

152.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

152.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

152.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

152.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

152.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

152.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for null pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

152.3 Summary

152.4 Other Issues discovered (unrelated to function)

Chapter 153

keyLock

- start = 2021-02-21 21:45
- end = 2021-02-21 22:05
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

153.1 Signature

```
int keyLock (Key * key, elektraLockFlags what)
```

153.2 Checklist

153.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] key is a valid key
  - [x] what contains valid elektraLockFlags
- [ ] `@post`
  - [x] parts of the keys, as stated in the what-parameter, are locked
- [ ] `@invariant`
  - [x] key stays a valid key
- [ ] `@param` for every parameter

    - [ ] key
    - [ ] what
- [ ] `@return` / `@retval`
    - [ ] move above see also
    - [ ] move `>0` to default case (`@return`)
- [ ] `@since`
    - [ ] add
- [ ] `@ingroup`
    - [ ] add
- [ ] `@see`
    - [ ] add `keyIsLocked()`

153.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
    - [ ] rename what to flags?

153.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

153.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

153.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] move below `keyGetRef()`-family of functions
- [x] No functions with similar purpose exist

153.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

153.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] keys cannot be unlocked

153.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] test null pointer
- [x] All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

153.3 Summary

Also check return values of function in tests

153.4 Other Issues discovered (unrelated to function)

Chapter 154

keyMeta

- start = 2021-01-23 18:05
- end = 2021-01-23 18:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

154.1 Signature

```
KeySet * keyMeta (Key * key)
```

154.2 Checklist

154.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] keyset -> KeySet
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] what happens if metadata is read-only?
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`

    - [x] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
    - [ ] move `@return` above `@retval`
    - [ ] add `@retval` 0 if key has no metadata
- [ ] `@since`
    - [ ] add
- [x] `@ingroup`
- [ ] `@see`
    - [ ] split into two lines

154.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

154.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

154.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [ ] Wherever possible, function parameters should be `const`
    - [ ] could key be const here? it doesn't get modified per se
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

154.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

154.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

154.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

154.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add check if key is NULL
  - [x] add check if key has no metadata
  - [x] add test if metadata is read-only
- All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation

154.3 Summary

Should this function work if metadata of Key is set to read-only?

154.4 Other Issues discovered (unrelated to function)

Chapter 155

keyName

- start = 2021-02-14 02:12
- end = 2021-02-14 02:44
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

155.1 Signature

```
const char *keyName(const Key *key)
```

155.2 Checklist

155.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [x] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
- [ ] `@pre`
  - [ ]
    Precondition

    key must be a valid key
- [ ] `@post`
  - [ ]
    Postcondition

    Pointer to key that can change over time
- [ ] `@invariant`
  - [ ] key name stays valid
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] add default return value to `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

155.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous
  - [ ] Function name might lead to confusions with `keyGetName()`, make their intentions more clear in their names
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

155.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

155.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

155.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist
  - [ ] `keyUnescapedName` might be considered too similar

155.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

155.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

155.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] empty keyName is not covered in tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

155.3 Summary

155.4 Other Issues discovered (unrelated to function)

Chapter 156

keyNeedSync

- start = 2021-03-07 18:05
- end = 2021-03-07 18:15
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

156.1 Signature

```
int keyNeedSync(const Key *key)
```

156.2 Checklist

156.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

    - [ ] split first line into two lines

- [ ] `@since`

    - [ ] add

- [x] `@ingroup`
- [ ] `@see`

    - [ ] split into two lines

    - [ ] move to bottom

156.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

156.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

156.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

156.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

156.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

156.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

156.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for `key == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

156.3 Summary

156.4 Other Issues discovered (unrelated to function)

Chapter 157

keyNew

- start = 2021-02-14 02:55
- end = 2021-02-14 03:22
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

157.1 Signature

```
Key *keyNew(const char *keyname, ...)
```

157.2 Checklist

157.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [x] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
  - [ ] is 0 a valid name? returns NULL if 0 is param
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x]
    Precondition

    name must be a valid key name
- [ ] `@post`
  - [x]

Postcondition

returns a valid key object

- [ ] `@invariant`
  - [x] add
- [ ] `@param` for every parameter
  - [ ] add `@param` for `...`
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

157.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

157.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

157.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

157.2.4 Structural Clarity

- [x] Functions should do exactly one thing
  - it doesn't technically - but its a convenience function
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

157.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

157.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [x] Docs say `Key *k = keyNew(0);` has same effect as `Key *k =keyNew("", KEY_END);`

157.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- [ ] All possible enum values are covered by tests
  - [x] KEY_META
  - [x] KEY_FLAGS
- [ ] No inconsistencies between tests and documentation
  - [x] Documentation says i can work with `Key *k =keyNew("", KEY_END);`
    Tests say
    ```
    k = keyNew ("", KEY_END); succeed_if (k == NULL, "should be invalid");
    keyDel (k);
    ```
  - [x] same as above with `keyNew(0)`

157.3 Summary

157.4 Other Issues discovered (unrelated to function)

Chapter 158

keyNextMeta

- start = 2021-02-27 17:10
- end = 2021-02-27 17:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

158.1 Signature

```
const Key *keyNextMeta(Key *key)
```

158.2 Checklist

158.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Sets internal iterator to the next meta-information entry'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add 'key is a valid key'
- [ ] `@post`
  - [x] add 'internal iterator set to next meta-information entry'
- [ ] `@invariant`
  - [x] add 'key stays valid'
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] `keyRewindMeta()`
  - [ ] `keyCurrentMeta()`

158.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

158.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

158.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

158.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

158.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

158.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

158.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add check for NULL pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

158.3 Summary

158.4 Other Issues discovered (unrelated to function)

Chapter 159

keyRewindMeta

- start = 2021-02-27 17:00
- end = 2021-02-27 17:10
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

159.1 Signature

```
int keyRewindMeta(Key *key)
```

159.2 Checklist

159.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] first entry in meta-information
- [ ] Simple example or snippet how to use the function
  - [ ] move up a bit
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] Use it -> Use this function
  - [ ] -then
  - [ ] So you want -> Afterwards, when you want to iterate the meta-information, you have
  - [ ] to **use** `keyNextMeta()`
- [ ] `@pre`
  - [x] add 'key must be a valid key'

- [ ] `@post`
  - [x] add 'key stays valid'
  - [x] add 'internal iterator set to first entry in metadata'
- [ ] `@invariant`
  - [x] add
- [ ] `@param` for every parameter
  - [ ] key: Key whose internal iterator should be rewinded
- [ ] `@return` / `@retval`
  - [ ] move success to normal `@return`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split first line into two

159.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

159.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

159.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

159.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

159.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

159.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

159.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for null pointer
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

159.3 Summary

159.4 Other Issues discovered (unrelated to function)

Chapter 160

keySetBaseName

- start = 2021-03-04 22:00
- end = 2021-03-04 22:35
- reviewer = Stefan Hanreich stefanhani@gmail.com

160.1 Signature

```
ssize_t keySetBaseName(Key *key,const char *baseName)
```

160.2 Checklist

160.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [x] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
  - [ ] add notice for read-only
  - [ ] add notice that root-name can't be removed
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] add `@retval` for read-only
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] remove first `@see` and merge description into the parts below

160.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

160.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

160.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

160.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swapped with `keyAddBaseName()`
- [x] No functions with similar purpose exist

160.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

160.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

160.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 1475
- [ ] All possible error states are covered by tests
  - [x] `baseNamePtr == NULL`
  - [x] `key == NULL`
  - [x] key is read-only
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

160.3 Summary

160.4 Other Issues discovered (unrelated to function)

Chapter 161

keySetBinary

- start = 2021-03-07 17:30
- end = 2021-03-07 17:40
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

161.1 Signature

```
ssize_t keySetBinary(Key *key, const void *newBinary, size_t dataSize)
```

161.2 Checklist

161.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'as a binary' -> to a binary value `newBinary`
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] add notice about read-only keys
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

161.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

161.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

161.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

161.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

161.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

161.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

161.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] text maxSize > SSIZE_MAX
    (impossible to check - overflow)
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

161.3 Summary

161.4 Other Issues discovered (unrelated to function)

Chapter 162

keySetMeta

- start = 2021-02-26 13:30
- end = 2021-02-26 14:15
- moderator = Stefan Hanreich `stefanhani@gmail.com`
- reviewer = Klemens Boeswirth

162.1 Signature

```
ssize_t keySetMeta(Key *key, const char* metaName,const char *newMetaString)
```

162.2 Checklist

162.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Set the value of a meta-information key
- [ ] Simple example or snippet how to use the function
  - [ ] add simple example
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] improve description
- [ ] `@pre`
  - [x] key should not have read-only metadata
  - [x] metaName must be a valid key name
- [ ] `@post`

  - [x] value set
  - [x] Key has a meta-information Keyset (?)
- [ ] `@invariant`
  - [x] key stays valid
- [ ] `@param` for every parameter
  - [ ] metaName - add dot, split into two lines
- [ ] `@return` / `@retval`
  - [ ] split `-1` into multiple lines
- [ ] `@since`
  - [ ] add `1.0.0`
- [x] `@ingroup`
- [ ] `@see`
  - [ ] add `keyMeta()`

162.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [ ] Parameter names should neither be too long, nor too short
  - [ ] newMetaString -> metaValue
- [ ] Parameter names should be clear and unambiguous
  - [ ] newMetaString -> metaValue

162.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

162.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

162.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist
  - [ ] `keyDelMeta()` ?
  - Similar thing could be achieved with `ksAppendKey ( keyMeta (key), keyNew (metaName, KEY_VALUE, newMeta↩ String, KEY_END) )`

162.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

162.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

162.2.7 Tests

- [x] Function code is fully covered by tests
  - [x] memory errors hard to cover
- [ ] All possible error states are covered by tests
  - [x] memory errors hard to cover
- All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation
  - [x] test case for key 0
  - [x] test case for name 0
  - [ ] test case for invalid name

162.3 Summary

- For 1.0.0 only metaNames with namespace `meta:/` should be accepted

162.4 Other Issues discovered (unrelated to function)

Chapter 163

keySetName

- start = 2021-02-14 03:30
- end = 2021-02-14 03:50
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

163.1 Signature

```
ssize_t keySetName(Key *key, const char *newname)
```

163.2 Checklist

163.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
- [ ] `@pre`
  - [ ]
    Precondition

    newName must be a valid name
  - [ ]
    Precondition

    must not be a read-only key
  - [ ]

Precondition

must not have been inserted before

- [ ] `@post`
  - [ ]
    Postcondition

    Key has (possibly modified) newName as name
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
  - [ ] add `@retval` -1 if key is read-only
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] add `keySetNameSpace()`

163.2.1 Naming

- [x] Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [x] Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

163.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

163.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [ ] Wherever possible, return types should be `const`
  - [ ] might be possible to make it `const`
- [x] Functions should have the least amount of parameters feasible

163.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swapped with functions for unescaped
- [x] No functions with similar purpose exist

163.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

163.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] behavior on invalid names

163.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] test_bit "CANNOT" fail, so might not be necessary to cover this
    for now should be made more resistant to future changes
- [ ] All possible error states are covered by tests
  - [x] test read-only keys
- All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation
  - [x] https://github.com/ElektraInitiative/libelektra/blob/master/tests/abi/testabi_key.c#L357
    checks for -1 if null pointer is provided
    documentation says 0 will be returned
  - [x] Documentation says name will be "" after an invalid name
    Tests show that name stays unchanged
    https://github.com/ElektraInitiative/libelektra/blob/master/tests/abi/testabi_key.c#L601
  - [x] Documentations should include stripping trailing /

163.3 Summary

163.4 Other Issues discovered (unrelated to function)

Chapter 164

keySetNamespace

- start = 2021-03-04 22:50
- end = 2021-03-04 23:05
- moderator = Stefan Hanreich `stefanhani@gmail.com`

164.1 Signature

```
ssize_t keySetNamespace(Key * key, elektraNamespace ns)
```

164.2 Checklist

164.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] add info for read-only keys
- [ ] Description of functions reads nicely
  - [ ] References to Key not working
- [x] `@pre`
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] move return values below returns
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] maybe add `keyGetNamespace()`

164.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

164.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

164.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

164.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

164.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

164.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

164.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] several cases in switches for enums not covered
  - [x] ns == `key->ukey[0]`
- [ ] All possible error states are covered by tests
  - [x] key == `NULL`
  - [x] ns == `KEY_NS_NONE`
- [ ] All possible enum values are covered by tests
  - [x] several cases in switches for enums not covered
- [x] No inconsistencies between tests and documentation

164.3 Summary

comment in Line 1619: growing -> shrinking

164.4 Other Issues discovered (unrelated to function)

Chapter 165

keySetString

- start = 2021-02-21 12:20
- end = 2021-02-21 18:10
- moderator = Stefan Hanreich stefanhani@gmail.com
- reviewer = Dominic Jäger

165.1 Signature

```
ssize_t keySetString(Key *key, const char *newString)
```

165.2 Checklist

165.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- Longer description of function containing common use cases
  - All uses cases covered by brief description
- [ ] Description of functions reads nicely
  - [ ] 'as new string value' seems strange
  - [ ] 'private copy' is unclear
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add

  - [x] string saved as UTF-8 in backend

- [ ] `@invariant`

  - [x] add

- [ ] `@param` for every parameter

  - [ ] 'text string' maybe redundant?

- [x] `@return` / `@retval`
- [ ] `@since`

  - [ ] add

- [x] `@ingroup`
- [ ] `@see`

  - [ ] split to multiple lines
  - [ ] shortly explain differences between `keyString` `keyGetString`
  - [ ] add `keySetBinary` ?
  - [ ] remove `keyValue`, as it is not related to string

165.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [ ] Parameter names should neither be too long, nor too short

  - [ ] `newStringValue` - remove Value?

- [x] Parameter names should be clear and unambiguous

165.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

165.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

165.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

165.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

165.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] iconv-Plugin
  - [ ] UTF-8
  - [ ] Return Value on NULL

165.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

165.3 Summary

Behavior when `newStringValue` is a NULL pointer seems strange

165.4 Other Issues discovered (unrelated to function)

Exact difference keyString keyGetString ?

Chapter 166

keyString

- start = 2021-02-26 16:05
- end = 2021-02-26 16:40
- moderator = Stefan Hanreich `stefanhani@gmail.com`
- reviewer = Robert Sowula

166.1 Signature

```
const char *keyString(const Key *key)
```

166.2 Checklist

166.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Get the pointer to the string representing the Key's value.
- [ ] Simple example or snippet how to use the function
  - [ ] add example
- [ ] Longer description of function containing common use cases
  - [ ] add explanation about pointers
  - [ ] add explanation about modifications from the user
- [ ] Description of functions reads nicely
  - [ ] `(null)` -> "`(null)`"
  - [ ] `(binary)` -> "`(binary)`"
- [ ] `@pre`

  - [x] add?

- [ ] `@post`

  - [x] add?

- [ ] `@invariant`

  - [x] add?

- [ ] `@param` for every parameter

  - [ ] move before return

  - [ ] key: key -> Key

- [ ] `@return` / `@retval`

  - [ ] pointer to the string representing the Key's value

  - [ ] `(null)` -> "`(null)`"

  - [ ] `(binary)` -> "`(binary)`"

- [ ] `@since`

  - [ ] add `1.0.0`

- [x] `@ingroup`
- [ ] `@see`

  - [ ] `keyGetString()`

  - [ ] `keyGetBinary()`

  - [ ] `keyValue()`

166.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous

  - [ ] `keyString()` vs `keyGetString()`

- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

166.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

166.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

166.2.4 Structural Clarity

- [ ] Functions should do exactly one thing
  - [ ] Return Values (null) and (binary)
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] swapped `with keyGetValueSize()`
- [x] No functions with similar purpose exist

166.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

166.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - "(null)" & "(binary)"

166.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 198
  - [x] Line 203
- All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

166.3 Summary

Think about changing the return values for 1.0.0. It seems like a hard step, but also inevitable to do at some point. Now would probably be better than in the future.

166.4 Other Issues discovered (unrelated to function)

Chapter 167

keyUnescapedName

- start = 2021-02-26 11:15
- end = 2021-02-26 12:15
- moderator = Stefan Hanreich `stefanhani@gmail.com`
- reviewer = Michael Tucek `michael@tucek.eu`

167.1 Signature

```
const void *keyUnescapedName(const Key *key)
```

167.2 Checklist

167.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Merge first and second sentence
    'Returns a key's name where the key name parts are separated by null bytes and does not use backslash for escaping'
- [ ] Simple example or snippet how to use the function
  - [ ] add example
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
  - [ ] key stays unchanged

  - [ ] returns pointer to unescaped Name

- [ ] `@invariant`

  - [ ] add

  - [ ] key stays unchanged

- [ ] `@param` for every parameter

  - [ ] 'optional pointer to Key object'

- [ ] `@return` / `@retval`

  - [ ] swap order

  - [ ] 'if Key has no name'

  - [ ] 'the Key's name'

- [ ] `@since`

  - [ ] add

- [ ] `@ingroup`

  - [ ] add

- [ ] `@see`

  - [ ] keyGetUnescapedName() for getting a copy

167.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous

  - [ ] `keyGetUnescapedName()` make difference to this function more clear?

- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

167.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

167.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

167.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] mismatch, move up in docs to match place in header file
- [x] No functions with similar purpose exist

167.2.5 Memory Management

- Memory Management should be handled by the function wherever possible

167.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

167.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] no tests for this function
- [ ] All possible error states are covered by tests
  - [x] no tests for this function
- [ ] All possible enum values are covered by tests
  - [x] no tests for this function
- [ ] No inconsistencies between tests and documentation
  - [x] no tests for this function

167.3 Summary

- Why is return type void* exactly?
- include optional in key `@param` - define schema for key param
- Add explicit tests for this function

167.4 Other Issues discovered (unrelated to function)

- Convention for keyX / keyGetX (pointer vs copy) intended?
- Remove elektra-internal functions
- Functions are sorted alphabetically - can we change this easily?

Chapter 168

keyValue

- start = 2021-03-20 18:45
- end = 2021-03-20 18:55
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

168.1 Signature

```
const void *keyValue(const Key *key)
```

168.2 Checklist

168.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [x] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
  - [ ] use correct testing functions in examples
- [ ] Description of functions reads nicely
  - 'and you should use it if' -> 'and you should use it, if'
- [ ] `@pre`
  - [x] add
- [ ] `@post`
  - [x] add
- [ ] `@invariant`
  - [x] add
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split first line and add explanations
  - [ ] add `keyIsBinary()`
  - [ ] add `keyIsString()`

168.2.1 Naming

- Abbreviations used in function names must be defined in the Glossary
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- Abbreviations used in parameter names must be defined in the Glossary
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

168.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

168.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

168.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [x] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

168.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

168.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions
  - [ ] return type on empty string/ binary?

168.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

168.3 Summary

168.4 Other Issues discovered (unrelated to function)

Chapter 169

keyVNew

- start = 2021-01-23 18:10
- end = 2021-01-23 18:10
- moderator = Stefan Hanreich `stefanhani@gmail.com`

169.1 Signature

```
Key *keyVNew(const char *keyname, va_list ap)
```

169.2 Checklist

169.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
- [ ] `@pre`
- [ ] `@post`
- [ ] `@invariant`
- [ ] `@param` for every parameter
- [ ] `@return` / `@retval`
- [ ] `@since`
- [ ] `@ingroup`
- [ ] `@see`

169.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
- [ ] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous

- [ ] Abbreviations used in parameter names must be defined in the Glossary
- [ ] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous

169.2.2 Compatibility

(only in PRs)

- [ ] Symbol versioning is correct for breaking changes
- [ ] ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

169.2.3 Parameter & Return Types

- [ ] Function parameters should use enum types instead of boolean types wherever sensible
- [ ] Wherever possible, function parameters should be `const`
- [ ] Wherever possible, return types should be `const`
- [ ] Functions should have the least amount of parameters feasible

169.2.4 Structural Clarity

- [ ] Functions should do exactly one thing
- [ ] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist

169.2.5 Memory Management

- [ ] Memory Management should be handled by the function wherever possible

169.2.6 Extensibility

- [ ] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions

169.2.7 Tests

- [ ] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
- [ ] All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation

169.3 Summary

169.4 Other Issues discovered (unrelated to function)

Chapter 170

ksAppend

- start = 2021-03-09 02:40
- end = 2021-03-09 02:50
- reviewer = Stefan Hanreich stefanhani@gmail.com

170.1 Signature

```
ssize_t ksAppend(KeySet *ks, const KeySet *toAppend)
```

170.2 Checklist

170.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Append all keys from `toAppend` to the end of KeySet `ks`'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [x] `@post`
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter
  - [ ] move above return

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [x] `@see`

170.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] `ks` -> `keySet`

170.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

170.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

170.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] fix
- [x] No functions with similar purpose exist

170.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

170.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

170.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 923
- [ ] All possible error states are covered by tests
  - [x] add test case for `ks == NULL`
  - [x] add test case for `toAppend == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

170.3 Summary

170.4 Other Issues discovered (unrelated to function)

Chapter 171

ksAppendKey

- start = 2021-03-20 18:25
- end = 2021-03-20 18:45
- reviewer = Stefan Hanreich stefanhani@gmail.com

171.1 Signature

```
ssize_t ksAppendKey(KeySet *ks, Key *toAppend)
```

171.2 Checklist

171.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move first example up
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter
  - [ ] move above `@return`

- [x] `@return` / `@retval`
- [ ] `@since`
  -[ ] add
- [ ] `@ingroup`
  -[ ] add
- [ ] `@see`
  - [ ] split first line
  - [ ] add explanations

171.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

171.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

171.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

171.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

171.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

171.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

171.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Line 812:813
  - [ ] Line 853:855
- [ ] All possible error states are covered by tests
  - [x] `ks == NULL`
  - [x] `toAppend == NULL`
  - [x] `toAppend->key == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

171.3 Summary

171.4 Other Issues discovered (unrelated to function)

Chapter 172

ksAtCursor

- start = 2021-03-14 10:35
- end = 2021-03-14 11:00
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

172.1 Signature

```
Key *ksAtCursor(KeySet *ks, elektraCursor cursor)
```

172.2 Checklist

172.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Return key at the position given by cursor `pos`'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] Doesn't seem to be accurate, change to cursor
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`

    - [ ] add
- [ ] `@param` for every parameter
    - [ ] `pos`: 'the cursor of the key that should be retrieved'
- [ ] `@return` / `@retval`
    - [ ] split `@retval` into multiple lines
- [ ] `@since`
    - [ ] add
- [ ] `@ingroup`
    - [ ] add
- [ ] `@see`
    - [ ] add `ksGetCursor()`
    - [ ] add `ksSetCursor()`

172.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
    - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
    - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
    - [ ] `ks` -> `keyset`
    - [ ] `pos` -> `position`
- [x] Parameter names should be clear and unambiguous

172.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

172.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [ ] Wherever possible, function parameters should be `const`
  - [ ] `ks` maybe could be `const`?
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

172.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

172.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

172.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

172.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test case for `pos >= ks->size`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

172.3 Summary

172.4 Other Issues discovered (unrelated to function)

Chapter 173

ksClear

- start = 2021-03-20 17:50
- end = 2021-03-20 17:50
- reviewer = Stefan Hanreich stefanhani@gmail.com

173.1 Signature

```
int ksClear(KeySet *ks)
```

173.2 Checklist

173.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
- [ ] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
- [ ] `@pre`
- [ ] `@post`
- [ ] `@invariant`
- [ ] `@param` for every parameter
- [ ] `@return` / `@retval`
- [ ] `@since`
- [ ] `@ingroup`
- [ ] `@see`

173.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
- [ ] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous

- [ ] Abbreviations used in parameter names must be defined in the Glossary
- [ ] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous

173.2.2 Compatibility

(only in PRs)

- [ ] Symbol versioning is correct for breaking changes
- [ ] ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

173.2.3 Parameter & Return Types

- [ ] Function parameters should use enum types instead of boolean types wherever sensible
- [ ] Wherever possible, function parameters should be `const`
- [ ] Wherever possible, return types should be `const`
- [ ] Functions should have the least amount of parameters feasible

173.2.4 Structural Clarity

- [ ] Functions should do exactly one thing
- [ ] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist

173.2.5 Memory Management

- [ ] Memory Management should be handled by the function wherever possible

173.2.6 Extensibility

- [ ] Function is easily extensible, e.g., with flags
- [ ] Documentation does not impose limits, that would hinder further extensions

173.2.7 Tests

- [ ] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
- [ ] All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation

173.3 Summary

173.4 Other Issues discovered (unrelated to function)

Chapter 174

ksCopy

- start = 2021-03-14 11:25
- end = 2021-03-14 11:40
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

174.1 Signature

```
int ksCopy(KeySet *dest, const KeySet *source)
```

174.2 Checklist

174.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move below description
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] Move text out of implementation section into normal description
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] `-1` when dest is a NULL pointer
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] split first line

174.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous

174.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

174.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

174.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist
  - [ ] `ksDup()`

174.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

174.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

174.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] `dest == NULL`
  - [x] `source == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

174.3 Summary

174.4 Other Issues discovered (unrelated to function)

Chapter 175

ksCurrent

- start = 2021-03-14 11:40
- end = 2021-03-14 12:00
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

175.1 Signature

```
Key *ksCurrent(const KeySet *ks)
```

175.2 Checklist

175.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Return the key the internal cursor currently points at'
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] split into multiple lines

175.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

175.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

175.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

175.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

175.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

175.2.6 Extensibility

- Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

175.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] `ks == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

175.3 Summary

175.4 Other Issues discovered (unrelated to function)

Chapter 176

ksCut

- start = 2021-03-20 17:30
- end = 2021-03-20 17:50
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

176.1 Signature

```
KeySet *ksCut(KeySet *ks, const Key *cutpoint)
```

176.2 Checklist

176.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] 'Cuts out all Keys from a KeySet that are below the specified key (see `keyIsBelow()`)'
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter

    - [ ] move above return
- [x] `@return` / `@retval`
- [ ] `@since`
    - [ ] add
- [ ] `@ingroup`
    - [ ] add
- [ ] `@see`
    - [ ] move to bottom
    - [ ] add `keyIsBelow()`

176.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
    - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
    - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
    - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

176.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

176.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

176.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

176.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

176.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

176.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] fully covered yes, but no return value is ever checked
  - [ ] add test case for the example at least
- [ ] All possible error states are covered by tests
  - [x] `ks == NULL`
  - [x] `cutpoint == NULL`
  - [x] `!ks->array`
  - [x] `cutpoint->key == NULL`
  - [x] 'cutpoint->key == ''`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

176.3 Summary

176.4 Other Issues discovered (unrelated to function)

Chapter 177

ksDel

- start = 2021-03-09 02:10
- end = 2021-03-09 02:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

177.1 Signature

```
int ksDel(KeySet *ks)
```

177.2 Checklist

177.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] fix `elektraFree()` ref
  - [ ] 'the release the'
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add

- [x] `@param` for every parameter
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [x] `@see`

177.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [x] Parameter names should be clear and unambiguous - [ ] ks -> keySet

177.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

177.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

177.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] fix
- [x] No functions with similar purpose exist

177.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

177.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

177.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for `keyset == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

177.3 Summary

177.4 Other Issues discovered (unrelated to function)

Chapter 178

ksDup

- start = 2021-03-09 02:20
- end = 2021-03-09 02:30
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

178.1 Signature

```
KeySet *ksDup(const KeySet * source)
```

178.2 Checklist

178.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [x] `@return` / `@retval`

- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] split
  - [ ] fix `keyDup()` refs

178.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] `ks` -> `keySet`

178.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

178.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

178.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] move to top
- [x] No functions with similar purpose exist

178.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

178.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

178.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] return value is never checked
- [ ] All possible error states are covered by tests
  - [x] add test case for `source == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

178.3 Summary

178.4 Other Issues discovered (unrelated to function)

Chapter 179

ksGetCursor

- start = 2021-03-14 11:00
- end = 2021-03-14 11:25
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

179.1 Signature

```
elektraCursor ksGetCursor(const KeySet *ks)
```

179.2 Checklist

179.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] Merge first two lines into brief description
  - [ ] 'Get `elektraCursor` for the key at the current internal cursor'
- [ ] Simple example or snippet how to use the function
  - [ ] move restoring state up
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add

- [x] `@param` for every parameter
- [ ] `@return` / `@retval`
  - [ ] move error cases to `@retval`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] split first line
  - [ ] add `ksAtCursor()`

179.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

179.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

179.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

179.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

179.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

179.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

179.2.7 Tests

- [ ] Function code is fully covered by tests
  - Line 1665
- [ ] All possible error states are covered by tests
  - [x] add test case if current cursor is at `NULL`
  - [x] add test case if current ks is `NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

179.3 Summary

179.4 Other Issues discovered (unrelated to function)

Chapter 180

ksGetSize

- start = 2021-03-09 02:30
- end = 2021-03-09 02:40
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

180.1 Signature

```
ssize_t ksGetSize(const KeySet *ks)
```

180.2 Checklist

180.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [x] `@return` / `@retval`

- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
- [ ] `@see`
  - [ ] remove params from `ksNew`
  - [ ] split lines
  - [ ] maybe remove both altogether?

180.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] `ks` -> `keySet`

180.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

180.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

180.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] fix
- [x] No functions with similar purpose exist

180.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

180.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

180.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [ ] add test for `ks == NULL`
    // todo
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

180.3 Summary

180.4 Other Issues discovered (unrelated to function)

Chapter 181

ksHead

- start = 2021-03-09 03:00
- end = 2021-03-09 03:10
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

181.1 Signature

```
Key *ksHead(const KeySet *ks)
```

181.2 Checklist

181.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

  - [ ] split `@retval` into two lines

- [ ] `@since`

  - [ ] add

- [ ] `@ingroup`

  - [ ] add

- [ ] `@see`

  - [ ] remove `ksRewind()` / `ksNext()`

181.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary

  - [ ] define `ks`

- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary

  - [ ] define `ks`

- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous

  - [ ] `ks` -> `keySet`

181.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

181.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

181.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] fix
- [x] No functions with similar purpose exist

181.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

181.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

181.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Line 1550
- [ ] All possible error states are covered by tests
  - [x] add test case for `ks == NULL`
  - [x] add test case for `ks->size == 0`
  - [x] check if `ksCurrent == NULL` if `ksCurrent == ksHead`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

181.3 Summary

181.4 Other Issues discovered (unrelated to function)

Chapter 182

ksLookup

- start = 2021-03-14 13:00
- end = 2021-03-14 13:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

182.1 Signature

```
Key *ksLookup(KeySet *ks, Key *k, elektraLookupFlags options)
```

182.2 Checklist

182.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] only example uses `ksLookupByName`
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [ ] `@return` / `@retval`
  - [ ] remove '0 otherwise' from `@return`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [x] `@see`

182.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Function names should neither be too long, nor too short
  - [ ] change name to `ksLookupByKey` to fit `ksLookupByName()`?
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

182.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

182.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

182.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

182.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

182.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

182.2.7 Tests

- [x] Function code is fully covered by tests
- [x] All possible error states are covered by tests
- [ ] All possible enum values are covered by tests
  - [x] `KDB_O_DEL`
- [x] No inconsistencies between tests and documentation

182.3 Summary

- [ ] Maybe convert some documentation from here to snippets and reference it in `ksLookupByName()`

182.4 Other Issues discovered (unrelated to function)

Chapter 183

ksLookupByName

- start = 2021-03-14 12:35
- end = 2021-03-14 13:00
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

183.1 Signature

```
Key *ksLookupByName(KeySet *ks, const char *name, elektraLookupFlags options)
```

183.2 Checklist

183.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [ ] Longer description of function containing common use cases
  - [ ] add part about options including reference to `elektraLookupFlags`
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter

  - [ ] shorten `options` by replacing explanation with reference to real explanation

- [ ] `@return` / `@retval`

  - [ ] remove '0 otherwise' from `@return`

- [ ] `@since`

  - [ ] add

- [ ] `@ingroup`

  - [ ] add

- [ ] `@see`

  - [ ] remove first 'See Also' section

183.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary

  - [ ] define `ks`

- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary

  - [ ] define `ks`

- [ ] Parameter names should neither be too long, nor too short

  - [ ] `ks` -> `keySet`

- [x] Parameter names should be clear and unambiguous

183.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

183.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

183.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

183.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

183.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

183.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] `name == NULL`
- [ ] All possible enum values are covered by tests
  - [ ] `KDB_O_DEL`
    // todo
- [ ] No inconsistencies between tests and documentation
  - [ ] several flags used in test, that can't be found in the documentation for `ksLookup()` or `elektra↩LookupFlags`
    // todo

183.3 Summary

- [ ] Maybe convert some documentation from `ksLookup()` to snippets and reference it in both

183.4 Other Issues discovered (unrelated to function)

Chapter 184

ksNeedSync

- start = 2021-03-20 16:55
- end = 2021-03-20 17:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

184.1 Signature

```
int ksNeedSync(const KeySet *ks)
```

184.2 Checklist

184.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [x] `@return` / `@retval`

- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] add `keyNeedSync()`

184.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

184.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

184.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

184.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

184.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

184.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

184.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [ ] add test case for `ks == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

184.3 Summary

184.4 Other Issues discovered (unrelated to function)

Chapter 185

ksNew

- start = 2021-03-14 22:15
- end = 2021-03-14 22:35
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

185.1 Signature

```
KeySet *ksNew(size_t alloc, ...)  ELEKTRA_SENTINEL
```

185.2 Checklist

185.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] 'you should read the next statements' seems a bit strange
  - [ ] remove 'va the list of arguments' heading
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add more
- [ ] `@invariant`
  - [ ] add

- [ ] `@param` for every parameter
  - [ ] improve description for alloc: clarify hint
- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] move to bottom
  - [ ] add `keyNew()`?
  - [ ] add `ksVNew()`?

185.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] `alloc` -> `size` / `initialSize`

185.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

185.2.3 Parameter & Return Types

- [x] Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

185.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist
  - [ ] `keyVNew()`

185.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

185.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

185.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] see review for `ksVNew()` as this is a wrapper for this
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

185.3 Summary

185.4 Other Issues discovered (unrelated to function)

Chapter 186

ksNext

- start = 2021-03-20 17:20
- end = 2021-03-20 17:30
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

186.1 Signature

```
Key *ksNext(KeySet *ks)
```

186.2 Checklist

186.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] refer to internal cursor?
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] split into two lines

186.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

186.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

186.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

186.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

186.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

186.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

186.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Line 1499
- [ ] All possible error states are covered by tests
  - [x] test case for iterating over empty keyset
  - [x] test case for `ks == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

186.3 Summary

186.4 Other Issues discovered (unrelated to function)

Chapter 187

ksPop

- start = 2021-03-14 22:35
- end = 2021-03-14 22:55
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

187.1 Signature

```
Key *ksPop(KeySet *ks)
```

187.2 Checklist

187.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add simple example
- [ ] Longer description of function containing common use cases
  - [ ] 'The reference counter' add 'of the Key'
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [ ] `@param` for every parameter

    - [ ] move up
- [x] `@return` / `@retval`
- [ ] `@since`
    - [ ] add
- [ ] `@ingroup`
    - [ ] add
- [ ] `@see`
    - [ ] remove `commandList()`
    - [ ] add `ksTail()`

187.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
    - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
    - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
    - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

187.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

187.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

187.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

187.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

187.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

187.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [ ] `ks == NULL`
    // todo
- All possible enum values are covered by tests
- [ ] No inconsistencies between tests and documentation

187.3 Summary

187.4 Other Issues discovered (unrelated to function)

Chapter 188

ksRewind

- start = 2021-03-09 02:50
- end = 2021-03-09 03:00
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

188.1 Signature

```
int ksRewind(KeySet *ks)
```

188.2 Checklist

188.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move below brief description
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [x] `@return` / `@retval`

- [ ] `@since`
  - [ ] add
- [x] `@ingroup`
- [ ] `@see`
  - [ ] split

188.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
  - [ ] `ks` -> `keySet`

188.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

188.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

188.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] fix
- [x] No functions with similar purpose exist

188.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

188.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

188.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] add test for `ks == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

188.3 Summary

188.4 Other Issues discovered (unrelated to function)

Chapter 189

ksSetCursor

- start = 2021-03-14 21:20
- end = 2021-03-14 21:40
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

189.1 Signature

```
int ksSetCursor(KeySet *ks, elektraCursor cursor)
```

189.2 Checklist

189.2.0.1 Doxygen

(bullet points are in order of appearance)

- [ ] First line explains briefly what the function does
  - [ ] append to 'to the given `cursor`'
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter

- [x] `@return` / `@retval`
- [ ] `@since`
  - [ ] add
- [ ] `@ingroup`
  - [ ] add
- [ ] `@see`
  - [ ] add `ksCurrent()` to `ksNext()`
  - [ ] move `ksGetCursor` to its own line

189.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
  - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary
  - [ ] define `ks`
- [ ] Parameter names should neither be too long, nor too short
  - [ ] `ks` -> `keySet`
- [x] Parameter names should be clear and unambiguous

189.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

189.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

189.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [x] No functions with similar purpose exist

189.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

189.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

189.2.7 Tests

- [x] Function code is fully covered by tests
- [ ] All possible error states are covered by tests
  - [x] `ks == NULL`
  - [x] `cursor == -1`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

189.3 Summary

return values are never checked in tests

189.4 Other Issues discovered (unrelated to function)

Chapter 190

ksTail

- start = 2021-03-09 03:10
- end = 2021-03-09 03:20
- reviewer = Stefan Hanreich `stefanhani@gmail.com`

190.1 Signature

```
Key *ksTail(const KeySet *ks)
```

190.2 Checklist

190.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] add
- [x] Longer description of function containing common use cases
- [x] Description of functions reads nicely
- [ ] `@pre`
  - [ ] add
- [ ] `@post`
  - [ ] add
- [ ] `@invariant`
  - [ ] add
- [x] `@param` for every parameter
- [ ] `@return` / `@retval`

  - [ ] split `@retval` into two lines

- [ ] `@since`

  - [ ] add

- [ ] `@ingroup`

  - [ ] add

- [ ] `@see`

  - [ ] remove `ksRewind()` / `ksNext()`

190.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary

  - [ ] define `ks`

- [x] Function names should neither be too long, nor too short
- [x] Function name should be clear and unambiguous
- [ ] Abbreviations used in parameter names must be defined in the Glossary

  - [ ] define `ks`

- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous

  - [ ] `ks` -> `keySet`

190.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

190.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

190.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
  - [ ] fix
- [x] No functions with similar purpose exist

190.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

190.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

190.2.7 Tests

- [ ] Function code is fully covered by tests
  - [ ] Line 1576
- [ ] All possible error states are covered by tests
  - [x] add test case for `ks == NULL`
  - [x] add test case for `ks->size == 0`
  - [x] check if `ksCurrent == ksTail` if `ksNext == NULL`
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

190.3 Summary

190.4 Other Issues discovered (unrelated to function)

Chapter 191

ksVNew

- start = 2021-03-14 21:40
- end = 2021-03-14 22:15
- reviewer = Stefan Hanreich <stefanhani@gmail.com>

191.1 Signature

```
KeySet *ksVNew(size_t alloc, va_list ap)
```

191.2 Checklist

191.2.0.1 Doxygen

(bullet points are in order of appearance)

- [x] First line explains briefly what the function does
- [ ] Simple example or snippet how to use the function
  - [ ] move up
- [x] Longer description of function containing common use cases
- [ ] Description of functions reads nicely
  - [ ] 'you should read the next statements' seems a bit strange
  - [ ] remove 'va the list of arguments' heading
- [ ] `@pre`
  - [ ] move up
  - [ ] add more
- [ ] `@post`
  - [ ] add more
- [ ] `@invariant`

    - [ ] add
- [ ] `@param` for every parameter
    - [ ] delete the first one
    - [ ] move below conditions / invariants
    - [ ] improve description for alloc: clarify hint
- [x] `@return` / `@retval`
- [ ] `@since`
    - [ ] add
- [ ] `@ingroup`
    - [ ] add
- [ ] `@see`
    - [ ] move to bottom
    - [ ] add `keyNew()`?
    - [ ] add `ksNew()`

191.2.1 Naming

- [ ] Abbreviations used in function names must be defined in the Glossary
    - [ ] define `ks`
- [x] Function names should neither be too long, nor too short
- [ ] Function name should be clear and unambiguous
    - [ ] What does the `V` mean?
- [ ] Abbreviations used in parameter names must be defined in the Glossary
    - [ ] define `ks`
- [x] Parameter names should neither be too long, nor too short
- [ ] Parameter names should be clear and unambiguous
    - [ ] `alloc` -> `size` / `initialSize`
    - [ ] `va` -> `params` / `parameters` / `va_list`

191.2.2 Compatibility

(only in PRs)

- Symbol versioning is correct for breaking changes
- ABI/API changes are forward-compatible (breaking backwards-compatibility to add additional symbols is fine)

191.2.3 Parameter & Return Types

- Function parameters should use enum types instead of boolean types wherever sensible
- [x] Wherever possible, function parameters should be `const`
- [x] Wherever possible, return types should be `const`
- [x] Functions should have the least amount of parameters feasible

191.2.4 Structural Clarity

- [x] Functions should do exactly one thing
- [x] Function name has the appropriate prefix
- [ ] Order of signatures in kdb.h.in is the same as Doxygen
- [ ] No functions with similar purpose exist
  - [ ] difference to `keyNew()`?

191.2.5 Memory Management

- [x] Memory Management should be handled by the function wherever possible

191.2.6 Extensibility

- [x] Function is easily extensible, e.g., with flags
- [x] Documentation does not impose limits, that would hinder further extensions

191.2.7 Tests

- [ ] Function code is fully covered by tests
  - [x] Line 254
  - [x] Line 270
- [ ] All possible error states are covered by tests
  - [x] above error cases seem hard to test
- All possible enum values are covered by tests
- [x] No inconsistencies between tests and documentation

191.3 Summary

Is this function still needed? Is it necessary to provide raw access, as keyNew is just a wrapper for this function that performs actions, which the user has to manually perform when using `keyVNew()`.

191.4 Other Issues discovered (unrelated to function)

Chapter 192

Elektra API Review

This folder contains all API design reviews conducted on `libelektra`. Below you can find a short description of the review process.

192.1 Review Process

This review is performed for each function separately. Each function should be evaluated according to the checklist. The reviewer judges for every bullet point, whether the function does / does not fulfill the bullet point. For every unfulfilled point a short explanation has to be given why the reviewer thinks the function does not fulfill the respective bullet point. This is done simply by providing a short description of the issue below the respective bullet point. After the review has been completed, the reviewer creates an issue for every unfulfilled bullet point so it can be fixed by the Elektra Initiative.

192.2 Checklist Legend

- [ ] not fulfilled
  This is the reason for why its is not fulfilled
- [x] fulfilled
- not applicable

192.3 Templates

The template and the script for generating review files based on that template can be found here

Chapter 193

API based on use cases for <tt>libelektra-core</tt>

193.1 Create <tt>Key</tt>

This use case is implemented as `keyNew()`.

193.2 <tt>Key</tt> Name

The name of a `Key` is stored in two forms: escaped and unescaped.
The escaped from can be accessed via `keyName()` and the unescaped form via `keyUnescapedName()`.
Manipulating the name is possible via the `keySetName()`, `keySetBaseName()`, `keyAddName()` and `keyAddBaseName()` functions, which allow various kinds of manipulation.

193.3 <tt>Key</tt> Namespace

This use case implemented via the `keyGetNamespace()` and `keySetNamespace()` functions.

193.4 <tt>Key</tt> Value

The value of a `Key` can be accessed via `keyValue()` and `keyString()`. It can be changed via `keySetString()` and `keySetBinary()`.

193.5 <tt>Key</tt> Metadata

This use case is implemented as the `keyMeta()` function.

193.5.1 <tt>Key</tt> Ordering

This use case is implemented as the `keyCmp()` function.

193.6 <tt>Key</tt> Hierarchy

This use case is currently implemented via the `keyIsBelow()`, `keyIsBelowOrSame()` and `keyIsDirectlyBelow()` functions, instead of being a single function.

193.7 Create <tt>KeySet</tt>

This use case is implemented as `ksNew()`. Beyond the described use case, the `ksNew()` function also supports creating a `KeySet` with predefined contents.

193.8 Insert <tt>Key</tt> into <tt>KeySet</tt>

This use case is implemented as `ksAppendKey()` and `ksAppend()`.

193.9 Remove <tt>Key</tt> from <tt>KeySet</tt>

This use case is implemented via `ksLookup()` with the `KDB_O_POP` option.

193.10 Direct lookup in <tt>KeySet</tt>

This use case is implemented via `ksLookup()`.

193.11 Cascading Lookup in <tt>KeySet</tt>

This use case is implemented via `ksLookup()`.

193.12 Index access to <tt>KeySet</tt>

This use case is implemented as `ksAtCursor()`.

193.13 Cut <tt>Key</tt> hierarchy from <tt>KeySet</tt>

This use case is implemented as `ksCut()`.

Chapter 194

API based on use cases for KDB

194.1 Get configuration

In `libelektra-kdb` this use case is directly implemented as the `kdbGet()` function. The API has the additional precondition, that `kdbOpen()` was used to create a `KDB` instance and that a `KeySet` exists into which the configuration will be loaded.
The use case is also implemented by `libelektra-highlevel`. However, it is not a direct mapping. Instead, the `elektraOpen()` function takes care of loading configuration, which can then be accessed key by key via the `elektraGet*()` function family.

194.2 Set configuration

In `libelektra-kdb` this use case is directly implemented as the `kdbSet()` function. The API has the additional precondition, that the current configuration has been loaded (at least once) via `kdbGet()` using the same `KDB` instance. Even though `kdbGet()` must be called, the entire configuration can be overwritten via `kdbSet()` as described in the use case.

194.3 Modify configuration

In `libelektra-kdb` this use case is directly implemented as the `kdbSet()` function.
The use case is also implemented by `libelektra-highlevel`. After loading the configuration via `elektraOpen()`, the `elektraSet*()` function family can be used to change individual configuration values (key by key).

194.4 Keeping configuration up-to-date

This use case is implemented by a combination of `libelektra-kdb`, `libelektra-notification` and at least one notification hook plugin. The notification hook plugin(s) must be enabled via the contract in `kdbOpen()`. Then `libelektra-kdb` will send update notifications via the notification hook plugin(s).
With the API of `libelektra-notification` applications can register callbacks to be called, when a notification is received.

194.5 Validating configuration with specification

This use case is implemented by a combination of `libelektra-kdb` and the spec hook plugin (normally the `spec` plugin). The `spec` plugin is enabled as a spec hook plugin by default, so this use case works out of the box. The main validation logic is implemented in the `spec` plugin.

Chapter 195

README.md

Files in this directory describe how the various use cases are implemented in the API.

Chapter 196

Contributor's glossary of Elektra

- **Change Tracking**: mechanism to detect changes that were made to keys in between `kdbGet` and `kdbSet` calls

Chapter 197

Copy-on-Write

The two basic Elektra datastructures `Key` and `KeySet` implement full copy-on-write (COW) functionality. If a key or a keyset gets copied, only a shallow copy with references to the original data (name, value, contained keys, etc.) is created. When this shared data is modified, new memory is allocated to keep the shared version in tact. Each piece of data (name, value, etc.) has a separate reference count to facilitate this. As a consequence, duplicated keys or keysets only require a fraction of the memory compared to their source counterparts.

197.1 Working principle

The main principle for the copy-on-write approach is simple: separate the data from the identity of a `Key` or `KeySet`. Developers using Elektra only get references to `Key` and `KeySet` objects. Those objects themselves do not hold any data. They only contain references to other entities holding the data. This allows transparently sharing the data-holding entities between multiple identifying entities.
The main entities for copy-on-write are depicted in the following diagram:

```
classDiagram
    Key *-- "0..1" KeyName : keyName
    Key *-- "0..1" KeyData : keyData
    KeySet *-- "0..1" KeySetData : data
    Key o-- "0..1" KeySet : meta
    KeySetData o-- "*" Key : array
    Key : - uint16_t refs
    Key : - keyflag_t flags
    KeyName : - char * name
    KeyName : - char * uname
    KeyName : - uint16_t refs
    KeyName : - uint16_t flags
    KeyData : - void * data
    KeyData : - uint16_t refs
    KeyData : - uint16_t flags
    KeySet : - uint16_t refs
    KeySet : - ksflag_t flags
    KeySetData : - uint16_t refs
    KeySetData : - uint16_t flags
```

- `Key`: logically represents a key. No data is actually stored in it.
  - `KeyName`: holds the name of a key. May be shared between multiple `Key` objects.
  - `KeyData`: holds the value of a key. May be shared between multiple `Key` objects.
- `KeySet`: logically represents a collection of keys. No data is actually stored in it.
  - `KeySetData`: holds an array of `Key` objects. May be shared between multiple `KeySet` objects.

When creating instances of `Key` and `KeySet` via `keyNew` or `ksNew`, we allocate the least amount possible. For `Key`, we only allocate a `KeyName` entity (while not technically necessary, it significantly simplifies some internal uses). For `KeySet`, we allocate no other entities. The missing entities will get allocated when needed.
If the name of a key gets modified, we first ensure that the `Key` holds the only reference to its `KeyName`. If it isn't, then a new `KeyName` object is allocated for it. Depending on the type of modification, the value of the name is also copied. The modifications to the name are then performed on the new `KeyName` entity.
The same applies for the value of a key, and for the data contained in keysets.
When copying the name or value of a key, or data of a keyset, via `keyCopy`, `ksCopy`, `keyDup` or `ksDup`, we point the according references in the `destination` object to the entities of the `source` object. The reference

counter of those entities is increased accordingly. The previous entities of the `destination` object will get their reference count decreased, and if they reach 0 they will be `free`d.

197.2 Reference counting

All entities are reference counted. The reference count is stored in each entity in an unsigned 16-bit integer variable `refs`. The special value `UINT16_MAX` is reserved as an error indicator, so the maximum number of references for each entity is `UINT16_MAX - 1` or 65534.
The reference counting mechanisms for `Key` and `KeySet` are part of the public API.
A reference count of 0 for `KeyName`, `KeyData` and `KeySetData` is illegal outside the public API function calls that cause the creation/deletion these structures.

197.3 Integration with <tt>mmapstorage</tt>

The `mmapstorage` plugin needs a flag for every entity to indicate whether the entity is stored via `mmap` or normally allocated via `malloc`. If an entity is flagged as being stored in `mmap`, it must **never** be freed using `free`.
The following flags are responsible for each entity:

- `Key`: `KEY_FLAG_MMAP_STRUCT` is set on the `flags` field
- `KeyName`: `KEY_FLAG_MMAP_KEY` is set on the `flags` field
- `KeyData`: `KS_FLAG_MMAP_Data` is set on the `flags` field
- `KeySet`: `KS_FLAG_MMAP_STRUCT` is set on the `flags` field
- `KeySetData`: `KS_FLAG_MMAP_ARRAY` is set on the `flags` field

Chapter 198

Documentation

This document gives guidelines for contributors concerning Elektra's documentation. This document takes preference to the actual situation. If you see documentation not according to this document, please `create an issue`. Alternatively, you can directly fix it with your next PR.

> **Note:** It is always allowed to improve the documentation, in every PR, even if the documentation fix is completely unrelated. However, separate PRs are preferred and can potentially get merged sooner.

198.1 Target Groups

We write documentation for three groups:

- **User:** A person who uses Elektra, e.g. by using an application, which utilizes Elektra, or one of Elektra's tools like `kdb`.
- **Developer:** A person who uses some library of Elektra within an application, tool or plugin.
- **Contributor:** A person who makes changes within Elektra's repository.

There are three separate folders for these three groups:

- for users
- for developers
- for contributors

> **Takeaway:** Every document must have a clear target group (user, developer or contributor). Sometimes it is clear from the directory, sometimes it must be explicitly stated, e.g. for tutorials.

198.2 Orientation

Each documentation should clearly be oriented to one of these three directions:

1. **learning-oriented:** First introduction is done via tutorials, they teach the *fundamentals*.
2. **information-oriented:** The `README.md` and API docs together are the *references*: They cover everything that someone needs to know about a module.
3. **understanding-oriented:** The doc/decisions explain the "Why?" something is done as it is done, i.e., the *rationale*.

Literature mentions also goal-oriented concepts, but we prefer *learning-oriented* approaches. E.g. of course you might have the goal to write a new plugin. But why not also learn about plugins while creating a new plugin?

> **Takeaway:** Don't try to combine different orientations in one document, instead split your documentation up in e.g. a `README.md` (*information*), tutorial (*learning*) and decisions (*understanding*).

198.3 Criteria

Elektra's documentation must fulfill:

- Always write what **is**, not what you would like to have. Explanations must always refer to the current situation (as changed with your PR).
- It is self-contained. It is not enough to link to some paper and an external page as explanation. All explanation must be within the repository, e.g., in case the external information goes away. This doesn't apply if the authoritative standard lives outside of Elektra. Then you would write, e.g., "The toml plugin implements [this standard](https://toml.io/en/v1.0.0), with following extensions:". The extensions, however, again must be fully listed within our repository. Make sure to link to the correct version of the standard.
- We use standard Markdown where possible, with only a few extensions:
  - styled fenced blocks
  - `- [ ]` option lists
  - `- <word>:<line break>` description lists
- The documentation should be as near to the code as possible.

  Takeaway: Include full API and Markdown documentation of the current situation directly in your PRs.

198.4 Style

- Sentences are short and written in one line. I.e. lines usually end with `.`, `:` or `;`. Avoid line breaks in the middle of the sentence.
- Use active and strong verbs early in your sentences. "We" refers to the community around the Elektra Initiative.
- Use headings and lists to keep a clear structure in the text.
- Use examples and images to emphasize important points, don't overuse emphasis in text (bold, etc.).
- Spelling is American English with spellings as defined here.
- It is consistent with our terminology.

  Note: Please extend terminology and spellings as needed.

198.5 Completeness

In general the documentation does not need to be complete. In particular, we do not want repetition of implementation details as documentation. Prefer to write self-documenting code. Nevertheless, there are a few must-haves:

- A `README.md` must be available for every module.
- A man page (`help/kdb-`) must be available for every command (including external commands).
- A tutorial must be present for every important concept.
- Every documentation page must be listed in the Website's structure.
- Everything copied must be properly licensed in reuse.

198.6 Links

Generously use links but be very careful to create a coherent documentation (German: "roter Faden"):

- Clearly separate between prerequisites and further readings.
- *Prerequisites:* Concepts people need to know before reading the documentation must be linked in the beginning.
- When adding links, check if users cannot easily get lost in circles.
- To link to Elektra's files use internal links. Use absolute or relative links as appropriate. E.g. for files within the same folder use relative links.
- For release notes use only external links to:
  - `www.libelektra.org` whenever possible, e.g. to tutorials, plugins, etc.
  - `master.libelektra.org` to link to source files
- For external links use **https** links, if available.
- Only use `*.libelektra.org/*` links, avoid `github.com/ElektraInitiative/*` links. Create an issue if redirects are missing. Rationale: Then we can more easily move to other Git hosting platforms. Redirects created for this purpose:
  - `issues.libelektra.org` for the issue tracker
  - `pulls.libelektra.org` for pull requests
  - `git.libelektra.org` as main page to the source repository

  Takeaway: Links are very helpful to readers. Make sure documentation can be read one after the other with these links (German: "roter Faden").

198.7 Templates

In general we use `arc42.org` but we use specialized templates for different modules:

- plugins
- libs
- bindings
- tools
- decisions

Chapter 199

mmapstorage

The mmapstorage plugin is a high performance storage plugin that supports full Elektra semantics.
The most important constraint for mmapstorage is that any structure (or bytes) that is an allocation unit (e.g. we malloc() the bytes needed for KeySet struct, so this is an unit) needs to have a flag to determine whether those bytes are actually malloc()ed or they are mmap()ed.
The mmapstorage plugin only calls munmap in some error cases, so basically the returned keyset is never invalidated and munmap is never called for its data.
During kdbSet the storage plugins always write to a temp file, due to how the resolver works. We also don't need to mmap the temp file here: when doing kdbSet we already have the KeySet at hand, mmap-ing it is not needed at all, because we have the data. We just want to update the cache file. The mmap/munmap in kdbSet are just so we can write the KeySet to a file in our format. (mmap() is just simpler, but we could also malloc() a region and then fwrite() the stuff)
Therefore the only case where we return a mmap()ed KeySet should be in kdbGet.
When the mmapstorage was designed/implemented, not all structures had refcounters, so there was no way to know when a munmap is safe. This was simply out of scope at that point in time.

Chapter 200

README.md

This folder contains information for contributors to Elektra.

- Documentation Guidelines
- Contributor's Glossary
- Copy on Write
- mmapstorage
- Session Recording
- API related to API design, maintenance etc. Please read if you plan any changes or extensions of the API.

Chapter 201

Session Recording Technical Documentation

A recording session is a period of time during which changes to the Key Database (KDB) are tracked and accumulated. It begins when recording starts and ends when recording stops. Throughout the recording session, all changes made to the KDB are recorded, including additions, modifications, and deletions of keys and their associated values. The recording session provides a complete audit trail of all changes made to the KDB during the specified period of time.
After every `kdbSet`, changes are calculated using Elektra's powerful changetracking API. The result of the calculation is an `ElektraDiff` instance we'll call *part diff* throughout this document. The session recording plugin merges those *part diffs* together and creates and persists an overall *session diff*.
Conceptually, this is depicted in the following image:
Importantly, the *session diff* is shared across processes. The *session diff* persistently records all changes made to the whole KDB from any process between the start and end of the session. Because of the cross-process nature of a *session diff*, it is also susceptible to "simultaneous write" conflicts.
We prevent inconsistent data by using locks. As long as recording is enabled, there is a global lock on write operations in Elektra. If a conflict occurs, it looks to the applications the same as if there was a conflict writing configuration data. The high-level Elektra bindings already resolve such conflicts transparently.
All persistable namespaces are monitored for changes.
An important concept for modified keys is the distinction between *old* and *new* keys. Old keys refer to the keys how they were (values, metadata) *before* the modifications. New keys, on the other hand, refer to how the keys are *after* the modifications. This concept does not apply to *added* or *removed* keys. You can think of added keys of only having new keys, and removed keys only having old keys.
We currently have no way of recording changes done outside of Elektra, i.e. when the configuration files got edited manually.

201.1 Storage of the Session Diff

The session diff is persisted in the respective namespace under `<namespace>:/elektra/record/session`. I.e. all keys in the diff of the `system` namespace are under `system:/elektra/record/session`.
The recording plugin needs its own KDB instance to store the session diff within Elektra. We provide hard coded default mountpoints for the

- `dir:/elektra/record/session`,
- `system:/elektra/record/session`,
- `spec:/elektra/record/session` and
- `user:/elektra/record/session` keys.

These mountpoints store the keys in the same storage format as the default mountpoints like `system:/` and `user:/`. The session storage file called `record-session.cfg` is located in the respective standard directories for their namespace.
The following list describes some important keys:

- `/elektra/record/config/active`

  - If the key is present, session recording is active.
  - The keys value is the parent key of the session.
  - We will only record changes made to keys same or below of this parent key.

- `/elektra/record/session`
  - Contains all the recorded data.
  - Should be mounted into separate files in each namespace.
- `/elektra/record/session/diff/added`
  - Contains all added keys.
- `/elektra/record/session/diff/modified/old`
  - Contains the *old* values and metadata for the keys that have been modified.
- `/elektra/record/session/diff/modified/new`
  - Contains the *new* values and metadata for the keys that have been modified.
- `/elektra/record/session/diff/removed`
  - Contains all removed keys.

201.2 Calculating the Session Diff

Keys in a diff are divided into different categories:

- Added: the key is new and did not exist before
- Modified: the key existed before and still exists but its value or metadata has been modified
- Removed: the key has been removed

Keys that stayed the same and therefore are not represented in a diff are called *unchanged* keys in the following paragraphs. The diagram below visualizes the state transitions when merging diffs.
The green ovals depict the state of a key in the session diff. The arrows depict the actions/state of a key in the new part diff.
For example, if a key is in `Added` state in the session diff, and it is in `Removed` state in the new part diff, then the key will be unchanged in the new version of the session diff. The transitions from `unchanged` to `Added`, `Modified` or `Removed` are not depicted, as they are quite trivial.
This functionality is quite generic and thus implemented as part of the normal diff API as `elektraDiffAppend`.

201.3 Architecture

The core recording feature has two main components:

1. Recording C API (`libelektra-record`): Implements everything the tooling needs.
2. Recording Plugin (`libelektra-recorder`): Gets loaded as a hook plugin and calls the appropriate functions in `libelektra-record`.

Chapter 202

DEBUGGING

Elektra can be debugged using gdb (either standalone or as part of an IDE like CLion).
To see the source files during debugging follow these steps:

1. Follow the Get started instructions to clone and install the repository.
   Note: To disable compiler optimizations and add debug symbols, add these flags when calling `cmake`:
   `cmake -DCMAKE_BUILD_TYPE=DEBUG -DENABLE_DEBUG=on -DCMAKE_C_FLAGS=-O0 -g`
2. Before starting gdb, set the environment variable `LD_LIBRARY_PATH` to the path where libelektra was installed during `make install` during step 1 (e.g. on Ubuntu 20.04.3 use `export LD_LIBRARY_PATH=/usr/local/lib/elektra`).
   If you use CLion, set the environment variable in your Debug Configuration.
3. After any code changes, you need to execute `make install` again to apply your changes.

Chapter 203

Builder Functions for <tt>Key</tt> and <tt>KeySet</tt>

203.1 Problem

The structs for `Key` and `KeySet` are opaque, i.e., only the `typedef`s are part of the public headers, the actual `struct` definitions are in a private header. Because of that, `libelektra-core` must provide a way to construct a new `Key` or `KeySet`.
Both `Key` and `KeySet` are rather complex structures consisting of multiple parts (name, value, metadata, and collection of `Key`s respectively), and `libelektra-core` is supposed to be minimal. It therefore makes sense to provide additional functions outside `libelektra-core` that make building `Key`s and `KeySet`s easier.
We call these functions "builder functions" compared to the Constructor Functions for `Key` and `KeySet`constructor functions"" in `libelektra-core`.

203.2 Constraints

- If the builder functions are in a non-language specific library, they must be directly callable from all languages for which Elektra provides bindings.
- In accordance with the Namespace and Name of KeysNamespace and Name of Keys" decision", the builder functions must not use the escaped name. Only the unescaped name may be used, but the namespace may be passed as a separate parameter, if this has benefits. Additionally, it might make sense to take the parts of a keyname as separate arguments.

203.3 Assumptions

203.4 Considered Alternatives

203.4.1 Language Agnostic

Note: This is basically, the "Full Arguments" solution from the constructor functions decision.

```
Key * keyBuild (const KeyName * name, const KeyValue * value, const KeyMeta metadata[]);
// allocates space for at least `alloc` keys
// inserts all keys from `keys` until it finds a NULL pointer, `keys` must contain at least one NULL
KeySet * ksBuild (size_t alloc, const Key * keys[]);
```

This could be called as:

```
int a = 7;
ksBuild(
  3,
  (Key*[]) {
    keyBuild (
      &(KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo\0bar", .size = 8 },
      &(KeyValue){.value = "1234", .size = 5},
      (KeyMeta[]){
        {.name = "type", .nameSize = 5, .value = {.value = "long", .size = 5}},
        {.name = "length\0min", .nameSize = 11, .value = {.value = "4", .size = 2}},
      }
```

```
    ),
    keyBuild (
      &(KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo", .size = 4 },
      &(KeyValue){.value = &a, .size = sizeof(a)},
      NULL
    ),
    keyBuild (&(KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "baz", .size = 4 }, NULL, NULL)
    NULL
  }
);
```

One neat benefit of this solution is that these simple functions can be called from any language.

203.4.2 Variadic Arguments

```
Key * keyNew (ElektraNamespace ns, ...);
```

This could be called as:

```
// system:/foo/bar with value 1234 and metadata: meta:/type=long, meta:/length/min=4
keyNew (ELEKTRA_NS_SYSTEM, "foo", "bar", NULL, KEY_VALUE, 5, "1234", KEY_META, "type", NULL, "long",
       KEY_META, "length", "min", NULL, 4, NULL);
// system:/foo/bar with value 1234 and no metadata
keyNew (ELEKTRA_NS_SYSTEM, "foo", "bar", NULL, KEY_VALUE, 5, "1234", NULL);
// system:/foo/bar with no value and no metadata
keyNew (ELEKTRA_NS_SYSTEM, "foo", "bar", NULL, NULL);
```

Compared to the language agnostic version, this loose type safety and doesn't really provide any advantages. Separating the keyname parts into separate arguments could also be done in the language agnostic version, by using an array `const char * parts[]` inside the `KeyName` struct.

203.4.3 Macros and Bindings

Some macros could make the language agnostic version above more ergonomic in C. Bindings in other languages, could also provide wrappers around the `keyBuild` and `ksBuild` functions. This could for example, take language-native array types as arguments to avoid passing sizes as separate arguments.
In C this could look like this:

```
#define KS_BUILD(...) (ksBuild (0, (Key *[]){__VA_ARGS__, NULL}))
#define KEY_NAME(ns_, name_) (KeyName){.ns = (ns_), .name = (name_), .size = sizeof (name_)}
#define KEY_VALUE_STRING(s) (KeyValue){.value = (s), .size = strlen((s))}
#define KEY_VALUE_PTR(v) (KeyValue){.value = &(v), .size = sizeof((v))
#define KEY_META_STRING(name_, value_) (KeyMeta){.name = (name_), .nameSize = sizeof (name_), .value = {
       .value = (value_), .size = strlen(value_) }}
```

This can be used like so:

```
int a = 7;
KS_BUILD(
  keyBuild (
    &KEY_NAME (ELEKTRA_NS_SYSTEM, "foo\0bar"),
    &KEY_VALUE_STRING ("1234"),
    (KeyMeta[]){
      KEY_META_STRING("type", "long"),
      KEY_META_STRING("length\0min", "4"),
    }
  ),
  keyBuild (
    &KEY_NAME (ELEKTRA_NS_SYSTEM, "foo"),
     &KEY_VALUE_PTR(a),
    NULL
  ),
  keyBuild (&KEY_NAME (ELEKTRA_NS_SYSTEM, "baz"), NULL, NULL)
);
```

203.5 Decision

203.6 Rationale

203.7 Implications

-
-
-

203.8 Related Decisions

- [](doc_decisions_0_drafts__README_md)
- [](doc_decisions_0_drafts__README_md)
- [](doc_decisions_0_drafts__README_md)

203.9 Notes

Chapter 204

Commit Function

204.1 Problem

When `kdbSet()` is called, plugins implementing the commit role, need to internally track their state to distinguish between carrying out that role and carrying out potential other roles (commit and setresolver for the resolver plugin, for example). This limits the possibilities of plugin reuse and the ways plugins can be combined.
More generally, this problem applies to all plugins with one function that is called multiple times for different reasons. Between `libelektra-kdb` and a backend plugins there is a contract that uses "phases" for this.
With the introduction of the new backend, a new `kdbCommit` function was introduced to `struct _Plugin` (matching `kdbError`). We also added a `kdbInit`. Therefore, some phases of `kdbGet`/`kdbSet` now use separate functions, but not all of them do. This partial use of separate functions can be weird and unintuitive.

204.2 Constraints

- Plugins should not have internal state that tracks how often a function has been called to determine what the function should do next.

204.3 Assumptions

204.4 Considered Alternatives

1. Keep the current "partial separation" approach.

   So far, there has been no technical reason why some phases use separate functions and others do not. At least there has been no explanation, why separate `kdbCommit`/`kdbError` would be better than having some other way of knowing the phase.

2. Fully separate the functions with one function for each of the phases `libelektra-kdb` knows.

   This would introduce a lot of symbols, which would bloat `struct _Plugin`. The bloat could be avoided by choosing the approach in the Plugin Contract FunctionPlugin Contract Function"" decision that removes the function pointers from `struct _Plugin`.

3. Only use `kdbOpen`, `kdbGet`, `kdbSet` and `kdbClose` functions matching the ones in `libelektra-kdb`. Communicate the current phase via the `elektraPluginGetPhase` function.

   Matching the functions from `libelektra-kdb` makes it clear, which plugin function gets used for which `kdb*` call. The API does not make it obvious that phases exist. Plugins that handle multiple phases must call `elektraPluginGetPhase` to know what phase is being executed. Also using a single `elektraPluginGetPhase` function means all phases must share a single type, even if not all phases are shared between `kdbGet` and `kdbSet`.

4. Like 3, but instead of using a `elektraPluginGetPhase` function, directly pass a `ElektraGetPhase phase`/`ElektraSetPhase phase` parameter.

This API makes it very clear that phases exist and that the plugin might need to handle them. It also allows using separate types for the phases of `kdbGet` and `kdbSet`. However, a big downside of this approach is that it adds an extra parameter to the API that will be unused by many plugins.

This API also allows plugins to call other plugins to execute a different phase. This could be a good thing or a bad thing.

204.5 Decision

204.6 Rationale

204.7 Implications

204.8 Related Decisions

- Plugin Contract Function
- Plugin Struct

204.9 Notes

Chapter 205

Constructor Functions for <tt>Key</tt> and <tt>KeySet</tt>

205.1 Problem

The structs for `Key` and `KeySet` are opaque, i.e., only the `typedef`s are part of the public headers, the actual `struct` definitions are in a private header. Because of that, `libelektra-core` must provide a way to construct a new `Key` or `KeySet`.
However, since both `Key` and `KeySet` are rather complex structures consisting of multiple parts (name, value, metadata, and collection of `Key`s respectively), it is not straightforward to create the best API for these functions.

205.2 Constraints

- Elektra has many bindings. The constructor functions must be callable from all these bindings, with no additional effort. In particular, that means the constructors must not use macros.
- Constructor functions should use as little resources as feasible. `malloc` and `memcpy` calls should be kept to a minimum and temporary allocations should be avoided.
- In accordance with the Namespace and Name of KeysNamespace and Name of Keys" decision", the constructor functions must not use the escaped name. Only the unescaped name may be used, but the namespace may be passed as a separate parameter, if this has benefits.

205.3 Assumptions

- There will be additional builder functions built on top of the constructor functions. These builder functions will live outside `libelektra-core`, but user code is expected to use builders rather than calling constructors directly.
- Sometimes, it may be desirable to have a function that can construct a `Key`, including name, value and metadata, or even whole `KeySet` in a single call. We assume these cases overlap almost entirely, with the cases where we expect builder functions to be used. Therefore, it doesn't matter much, if the constructor functions don't allow this can of `Key`/`KeySet` construction and extra calls to e.g., `keySetValue` are needed.

205.4 Considered Alternatives

205.4.1 No Arguments

The constructor functions are just specialized allocators:

```
Key * keyNew (void);
KeySet * ksNew (void);
```

While this would work decently well for `ksNew` (don't allocate array, only allocate array on first `Key` insertion), there is an issue for `keyNew`. A `Key` must have a name. Therefore, `keyNew` must set some default name in the `Key` it returns.

Even if there was a suitable default name, it would still be wasteful, since in many (almost all) cases, the name will soon be replaced via a `keySetName` call.
A clear advantage of this option is that the very simple API means the functions are callable without issue from basically any language.

205.4.2 Minimal Arguments

Instead of taking no arguments at all, the constructor functions take the minimal number of arguments:
```
// ns is separate to allow usage of ELEKTRA_NS_* constants
// size is needed because name can contain \0
Key * keyNew (ElektraNamespace ns, const char * name, size_t size);
KeySet * ksNew (size_t alloc);
```
This solves the default name issue for `keyNew`. For `ksNew` there was no issue and the `alloc` parameter isn't strictly speaking needed, but it can still be helpful. For example, if the caller knows they will insert 100 keys, the can call `ksNew (100)` to avoid later allocations to resize the array in the `KeySet`.
The API is also still simple enough that it can be called from any binding.

205.4.2.1 Sidenote: Bundle struct

Depending on the rest of the `libelektra-core` API, it may make sense to use a public struct to bundle the arguments of `keyNew`:
```
typedef struct {
  ElektraNamespace ns;
  const char * name;
  size_t size;
} KeyName;
Key * keyNewStruct (KeyName name);
// called as e.g.: keyNew ((KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo\0bar", .size = 7 })
Key * keyNewStructAlt (const KeyName * name);
// called as e.g.: keyNew (&(KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo\0bar", .size = 7 })
```
Whether the struct would be passed by value or as a pointer also depends on the rest of the API.
An important distinction between `Key` and `KeyName` in this solution is that `Key` can be seen as more of class, while `KeyName` is just a bundle of fields. That is why `Key` is opaque and `KeyName` would be public. The fields of a `Key` are implementation details, but `KeyName` is just a kind of alias for its fields.

205.4.3 Common Arguments

Lots of `Key`s will have a value from the moment they are created, e.g., `meta:/` keys are rarely created without a value. Therefore, it might make sense if `keyNew` took an optional (=nullable) argument for the value of the key:
```
typedef struct {
  const void * value;
  size_t size;
} KeyValue;
Key * keyNew (const KeyName * name, const KeyValue * value);
// called as e.g.:
// keyNew ((KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo\0bar", .size = 7 }, NULL)
// keyNew ((KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo\0bar", .size = 7 }, (KeyValue){.value = "1234",
       .size = 5})
```

> **Note**: Because this solution is much easier with the bundle structs, we use them here. It would work without them as well, but we'd need two optional arguments (pointer and size) for the value. Similarly, passing the bundle struct by value would mean you have to pass `(KeyValue){ .value = NULL, .size = 0 }` instead of just `NULL`.

For `KeySet`, we can pass a list of `Key *` to initialize the `KeySet` with:
```
// every variadic argument must be a Key *, the last argument must be NULL
KeySet * ksNew (size_t alloc, ...);
```
This API is easy to call in C:
```
KeySet * ks = ksNew (3, key1, key2, key3, NULL);
```
However, many languages for which we provide bindings can't use variadic arguments. So it would make more sense to have a function like this only in a C-specific library (`libelektra-lowlevel-c`), and have the version from above in `libelektra-core`.
An alternative would be to use an array argument:
```
// last element of `keys` must be NULL
KeySet * ksNew (size_t alloc, Key * keys[]);
```
This is easier to call from other languages, but it's slightly more cumbersome in C:
```
KeySet * ks = ksNew (3, (Key*[]){key1, key2, key3, NULL});
```

The array parameter also has some other advantages. We could copy it with a single `memcpy` and use something like `qsort`, instead of copying the `Key *` one by one from the variadic arguments. Also, a `Key *[]` provides more type safety compared to a variadic arguments.

205.4.4 Full Arguments

For `ksNew` the above solution already uses the full set of arguments to initialize a `KeySet` fully.
For `Key` we'd also need to take metadata.

> **Note**: This solution is described only for completeness's sake. We assume that Builder Functions for `Key` and `KeySet`builder functions"" exist outside of `libelektra-core`. With this solution those builders would be superfluous.

205.4.4.1 Variadic Arguments

This could be done by using a system of variadic arguments, called like so:

```
// system:/foo/bar with value 1234 and metadata: meta:/type=long, meta:/length/min=4
keyNew (ELEKTRA_NS_SYSTEM, "foo", "bar", NULL, KEY_VALUE, 5, "1234", KEY_META, "type", NULL, "long",
        KEY_META, "length", "min", NULL, 4, NULL);
// system:/foo/bar with value 1234 and no metadata
keyNew (ELEKTRA_NS_SYSTEM, "foo", "bar", NULL, KEY_VALUE, 5, "1234", NULL);
// system:/foo/bar with no value and no metadata
keyNew (ELEKTRA_NS_SYSTEM, "foo", "bar", NULL, NULL);
```

However, as discussed above, such functions are hard to call from many other languages. You also loose type safety in C and the function is not particularly intuitive to use. To emphasize this last point, consider that the signature for the above `keyNew` would likely be:

```
Key * keyNew (ElektraNamespace ns, ...);
```

Things can be slightly improved, by passing the keynames as a single string, but it still a bad API, with lost of potential for misuse.

205.4.4.2 Metadata Array

A better option would be to build on the bundle structs option from above, by adding an new struct for metadata:

```
typedef struct {
  // no namespace, because that is always ELEKTRA_NS_META
  const char * name;
  size_t nameSize;
  KeyValue value;
} KeyMeta;
Key * keyNew (const KeyName * name, const KeyValue * value, const KeyMeta metadata[]);
```

This could be called as:

```
// system:/foo/bar with value 1234 and metadata: meta:/type=long, meta:/length/min=4
keyNew (
  &(KeyName){.ns = ELEKTRA_NS_SYSTEM, .name = "foo\0bar", .size = 8 },
  &(KeyValue){.value = "1234", .size = 5},
  (KeyMeta[]){
    {.name = "type", .nameSize = 5, .value = {.value = "long", .size = 5}},
    {.name = "length\0min", .nameSize = 11, .value = {.value = "4", .size = 2}},
  });
```

205.5 Decision

Suggestion: Minimal Arguments

205.6 Rationale

- The "Minimal Arguments" solution can still be considered minimal, while the more complex solutions are not really minimal anymore.
- For `libelektra-core` "Minimal Arguments" is enough. The more complex APIs can be provided as builder functions in other libraries.

205.7 Implications

- You need an extra library beyond `libelektra-core` to construct `Key`s and `KeySet`s in a single call.

205.8 Related Decisions

- Namespace and Name of Keys
- Builder Functions for `Key` and `KeySet`

205.9 Notes

Chapter 206

Man Pages

206.1 Problem

Our man pages are written as Markdown in `doc/help` and then converted to roff and stored in `doc/man`. These are the only generated files in our version control system. Having such files is a problematic workaround, which was introduced because `ronn-ng` is not available on most distributions. The poor availability of the package `ronn-ng` is a problem because distributions usually build packages by exclusively relying on other packages of the distribution. E.g. `dpkg-buildpackage` must work with only `deb` packages installed (and not any packages via `gem`, as would be needed to get `ronn-ng v0.10.1`).
We have a mechanism to automatically disable (re)building man pages. But we want to avoid that distributions build packages without man pages, hence we added the generated files.
Storing generated files, however, is problematic, as it requires:

- developers to always update generated files if the sources are changed
- developers not committing irrelevant changes to generated files (e.g. as may occur with different `CMAKE_INSTALL_PREFIX` etc.)
- require extra effort for continuous integration, e.g. `#4542`

TODO: everything below is draft, please don't comment on it.

206.2 Constraints

1. we want beautiful rendered man pages, e.g., OPTIONS section looks like normal man pages, see in Notes[1] below
2. we cannot require rare tools for the build process: the man pages must be present in every package

206.3 Assumptions

206.4 Considered Alternatives

0. staying with `ronn-ng` but add the man pages only in the release tarballs but not to `git`

1. Write a tool that converts our specification, similar to pythongen
2. Write a tool that parses gopts `--help` output
3. `help2man`
4. Doxygen:
    - Constraint 1 probably broken
5. Pandoc:

- has a few standard dependencies
- would need rewrite of the current documentation in doc/help:
  - To fulfill Constraint 1 `definition lists` would be needed
  - would need YAML metadata/front matter for every file (It would be possible, but not advisable, to:
    * also pass information as command-line arguments via `--variable` but then we would move meta-information about man pages to the build system
    * that we use the current (non-standard) front matter and convert it to Pandoc's frontmatter but this makes the build system more complicated)

206.5 Decision

Not yet done except spelling of man pages, see #4567.

206.6 Rationale

206.7 Implications

206.8 Related Decisions

- [](doc_decisions_0_drafts__README_md)
- [](doc_decisions_0_drafts__README_md)
- [](doc_decisions_0_drafts__README_md)

206.9 Notes

[1] ronn-ng converts:

```
- `-H`, `--help`:
  Show the man page.
- `-V`, `--version`:
  Print version info.
- `-p`, `--profile <profile>`:
  Use a different kdb profile.
```

to:

```
OPTIONS
       -H, --help
              Show the man page.
       -V, --version
              Print version info.
       -p, --profile <profile>
              Use a different kdb profile.
```

Chapter 207

Metakey semantics

207.1 Problem

Metakeys (i.e., keys with namespace `KEY_NS_META`) have special requirements beyond what other namespaces want. This is because metakeys are used for specifications.
These extra requirements are:

1. Copying metadata from `spec:/` keys to other namespaces must share as much memory as possible. Without sharing memory there would be massive unnecessary duplication of data.
2. It must be possible to identify whether a metakey was copied from `spec:/` key, or was set directly. Additionally, it must be possible to detect if the value of a copied metakey has changed.
3. Changing the value of a copied metakey must not affect the original `spec:/` key (and its metadata).

Unrelated to the specification use case (see also Use Case: Validating Configuration with SpecificationValidating Configuration with Specification""), metakeys must also be prevented from having metadata of their own. As `keyMeta` now "leaks" (compared with the previous API) the `KeySet` of metadata is unprotected and the requirements above are not fulfilled anymore. For example, changes of metadata values could confuse the spec plugin and lead to invalid configuration passed to applications.

207.2 Constraints

- The API should be hard to misuse.
- Elektra should protect against incorrect operations, that would lead to undefined behavior.
- As a specialized hook plugin `spec` could make use of specialized APIs. However, the need for such internal APIs should be limited as much as possible.

207.3 Assumptions

- Not allowing metadata on metakeys, can also be ignored here, because it can be dealt with in `keyNew()` and `keyMeta()`. `keyMeta()` can just return `NULL` and `keyNew()` can either ignore metadata arguments or fail if metadata is given.

207.4 Considered Alternatives

207.4.1 Completely Read-only

One option to enforce all requirements is to just make any `Key` that is created with the `KEY_NS_META` namespace entirely read-only after `keyNew()`.
Because the `Key` is entirely read-only, it's value cannot change. Therefore, we can find out, if it was copied from a given `spec:/` key by doing a pointer comparison:

```
KeySet * specMeta = keyMeta (specKey);
KeySet * otherMeta = keyMeta (otherMeta);
for (elektraCursor it = 0; it < ksGetSize (otherMeta); it++)
{
   Key * metaKey = ksAtCursor (otherMeta, it);
   if (metaKey == ksLookup (specMeta, metaKey, 0))
   {
      // copied from spec
   }
}
```

To update metadata you would have to create a new `Key` to replace the existing one:

```
ksAppendKey (keyMeta (key), keyNew ("meta:/type", KEY_VALUE, "string", KEY_END));
// would always fail, because metakeys are read-only
keySetString (ksLookupByName (keyMeta (key), "meta:/type", 0), "string");
```

Safely copying metadata would be very simple with this solution.

```
ksAppend (keyMeta (dest), keyMeta (source))
```

This copies metadata while sharing the memory for the individual `Key`s. Because the `Key`s are read-only they cannot be changed at all, so we don't have to worry about changes to `dest` affecting `source`. If we want to change the metadata of `dest` we have to create a new `Key`, which will not be used by `source`.

207.4.2 Read-only while in <tt>KeySet</tt>

A relaxation of the above solution would be to only make the `Key` read-only, while it is part of a `KeySet`.
The snippets from above would still work and changing the value of metakey directly as above would still fail. But now you don't have to create the `Key` directly with the right name and value.

```
Key * metaKey = keyNew ("meta:/", KEY_END);
keyAddBaseName (metaKey, somePart);
keySetString (metaKey, someValue);
```

The snippet above would not work, if the key is returned as read-only directly from `keyNew()`.
Copying metadata works the same as above. Since the `Key`s are still read-only as long as `source` uses them, we again cannot affect `source` by changing `dest`.

207.4.3 Utilize COW implementation

207.4.3.1 Issues with Read-only Solutions

An issue with the solutions above is that you always have to create a new `Key` to change metadata. This means we need to allocate all the memory for a new metakey. This new metakey will replace the existing one in `keyMeta(key)`. But that means, if the existing metakey was not shared with other keys, it will be deleted, when we could have just reused that memory.
Some of that problem is mitigated by `keyDup` using copy-on-write (COW). However, let's look at what is needed to change the value of `meta:/type` with a read-only solution.
The straightforward option doesn't work, because of the read-only nature of metakeys:

```
// FAILS, metakey is read-only
keySetString (ksLookupByName (keyMeta (key), "meta:/type", 0), "string");
```

To make proper use of COW you need to write something like this to change the value of `meta:/type`:

```
// Note: snippet could be even more complex, if you want to avoid the `keyDup`, in case meta is not in
      another KeySet
Key * meta = ksLookupByName (keyMeta (key), "meta:/type", 0);
meta = meta == NULL ? keyNew ("meta:/type", KEY_END) : keyDup (meta, KEY_CP_NAME);
keySetString (meta, "long");
ksAppendKey (keyMeta (key), meta);
```

However, most people would probably opt for the much simpler:

```
ksAppendKey (keyMeta (key), keyNew ("meta:/type", KEY_VALUE, "long", KEY_END));
```

For a one-time operation the difference might not be big, but if a metakey (probably not `meta:/type`), changes many times it will become significant. Every time the simpler solution is used, an entirely new `Key` is created instead of utilizing the COW approach.

207.4.3.2 Possible Fix

One option to solve that read-only problems, in "Read-only while in `KeySet`" is something like this:

```
Key * metaKey = ksLookupByName (keyMeta (key), "meta:/type", KDB_O_POP);
if (isInKeySet (metaKey)) // <- hypothetical API
{
   // still used, can't modify -> use new key
   keyDel (metaKey);
   ksAppendKey (keyMeta (key), keyNew ("meta:/type", KEY_VALUE, "string", KEY_END));
}
else
{
```

```
   keySetString (metaKey);
   ksAppendKey (keyMeta (key), metaKey);
}
```

> **Note**: Where the code above would live doesn't matter. It may be part of some Elektra library, or it may be user code. The code is clearly not ideal and that's why the solution here actually is to avoid the need for a `isInKeySet` function entirely.

However, that still has some obvious issues:

- It needs a public API `isInKeySet` to check whether a metakey is used by some metadata `KeySet`.
- We still need to take the metakey out of the metadata `KeySet` and reinsert it. That means shuffling around the array in the `KeySet`, which may be worse than the duplicate memory for the new metakey.

Clearly this is **not** a viable solution to the problem. But the good thing is we can solve these issues, because of the COW implementation.
To solve the issues mentioned above, `spec` needs to do a few things differently:

207.4.3.3 Copying Metakeys in <tt>spec</tt>

Instead of copying metadata with
```
ksAppend (keyMeta (dest), keyMeta (source))
```
we need to make a copy of all metakeys
```
KeySet * sourceMeta = keyMeta (source);
KeySet * destMeta = keyMeta (dest);
for (elektraCursor i = 0; i < ksGetSize (sourceMeta); i++)
{
   ksAppendKey (destMeta, keyDup (ksAtCursor (sourceMeta, i), KEY_CP_ALL));
}
```
This seems like a bad change, but because of the COW implementation it's not as bad as it looks. Only the `struct Key` will be duplicated, both the name and value data of the metakeys will still be shared between `source` and `dest`. Still worse than reusing everything and just adding a few new pointers, but not too bad.
This change means that modifying metadata in `dest` cannot possibly affect `source`, because they do not share any `Key *`s. At first, they do share all the name and value data, but through COW that stops as soon as either `Key` is modified. Therefore, we don't need to make the value read-only and anybody (including user code) can just do:
```
keySetString (ksLookupByName (keyMeta (key), "meta:/type", 0), "string");
```

207.4.3.4 Detecting & Removing Copied Metakeys in <tt>spec</tt>

The issue with not sharing `Key *`s of course is that a pointer comparison will no longer detect copied metakeys. That means we need a new way of detecting what `spec` needs to remove.
First we'd add a new `keyGetCOWValue` function:
```
KeyData * keyGetCOWValue (Key * key)
{
   return key->keyData;
}
```

> **Note**: This function wouldn't be part of `libelektra-core`. It would be in `libelektra-extra` or some other library on which `spec` would depend.

With this function, the check to detect copied metakeys becomes this straightforward snippet:
```
KeySet * specMeta = keyMeta (specKey);
KeySet * otherMeta = keyMeta (otherMeta);
for (elektraCursor it = 0; it < ksGetSize (otherMeta); it++)
{
   Key * otherMetaKey = ksAtCursor (otherMeta, it);
   if (keyGetCOWValue (otherMetaKey) == keyGetCOWValue(ksLookup (specMeta, metaKey, 0)))
   {
      // copied from spec
   }
}
```
The check above works, because changing the value of a `Key` will allocate a new `key->keyData`, if it is shared with another key. Therefore, the `keyData` pointers will only be the same if the value was not modified and was copied from the `spec:/` key.

Note: Adding `keyGetCOWValue` (anywhere) might cause problem with some of the current guarantees related to COW. A quick solution for this would be to instead only provide `keyHasSame↩ COWValue`:

```
bool keyHasSameCOWValue (Key * key, Key * other)
{
  if (key == NULL && other == NULL) return true;
  if (key == NULL || other == NULL) return false;
  return key->keyData == other->keyData;
}
```

This would still allow making the comparison needed by `spec`, but doesn't actually provide any access to `key->keyData`, so nothing no guarantees can be broken.

207.5 Decision

207.6 Rationale

207.7 Implications

207.8 Related Decisions

- Copy On Write
- Types of `KeySet`s

207.9 Notes

Chapter 208

Notifications

208.1 Problem

`kdbEnsure` (contracts) shifted expectations: Applications want control over if they receive notifications.

208.2 Constraints

- applications need to send notifications in any case, otherwise others cannot receive it

208.3 Assumptions

- multiple transport mechanism at once do not need to be supported

208.4 Considered Alternatives

208.5 Decision

Have a symlink to which transport notification plugin is used. This notification plugin is used by default to sent notifications.
If applications also want to receive notifications is up to the contract.

208.6 Rationale

208.7 Implications

The decision which transport mechanism, e.g. dbus or zeromq, must be chosen per computer node or container. Within one Elektra setup, they cannot be intermixed (unlike config file formats, where many different formats can be mixed).

208.8 Related Decisions

- hooks
- Ensure

208.9 Notes

Chapter 209

Operation sequences

209.1 Problem

The current public API allows various kind of sequences of `kdbGet` and `kdbSet` calls. There is no real guideline of allowed and/or prohibited sequences, other than `kdbGet` for a certain parent key must be called before `kdbSet`:

- initially after `kdbOpen`
- after conflict errors in `kdbSet`

We discovered that some sequences are currently legal but problematic for change tracking and maybe also other use cases.
Currently, notification plugins assume that a `kdbSet` operation for a certain parent key always directly follows the `kdbGet` operation for that same key. However, this is not enforced in any way within Elektra, nor is it documented that it should.
In applications, `kdbGet` and `kdbSet` of different parent keys might be interwoven in any way. This is problematic, as plugins may store the result of `kdbGet` and use the stored data in the following `kdbSet` to calculate a change-set. Plugins have to do this, as there isn't any other Elektra-provided mechanism currently to do change tracking for them.
Note, that erroneous behavior only occurs if plugins are used globally, i.e. as `notification-send` hook plugins. If they are mounted for specific backends (via `kdb mount`), this behavior will not occur as plugins from different backends are isolated.

209.1.1 Reproducible Example

The following example will demonstrate the problematic sequence as real, runnable code. This example will show the erroneous behavior for the `dbus` and `logchange` plugins, if used as `notification-send` hooks. These both plugins just serve as a general demonstration, and there may be more plugins that do change tracking this way.
First, run some setup steps:

```
# enable the dbus plugin as a notification-send hook
kdb set system:/elektra/hook/notification/send/plugins/#0 dbus
# enable the logchange plugin as a notification-send hook
kdb set system:/elektra/hook/notification/send/plugins/#1 logchange
# mount user:/a and user:/b as two backends
kdb mount a.ecf user:/a
kdb mount b.ecf user:/b
# populate user:/a and user:/b with values
kdb set user:/a/brightness 1
kdb set user:/a/saturation 1
kdb set user:/b/foreground 1
kdb set user:/b/background 1
```

Then, run the following command to monitor the notifications by the `dbus` plugins:

```
dbus-monitor --session type='signal',interface='org.libelektra'
```

Finally, execute the following program:

```
#include <kdb.h>
#include <stdio.h>
int main (void)
{
        Key * root = keyNew ("/", KEY_END);
        KDB * handle = kdbOpen (NULL, root);
        Key * keyA = keyNew ("user:/a", KEY_END);
```

```
        Key * keyB = keyNew ("user:/b", KEY_END);
        KeySet * a = ksNew (0, KS_END);
        KeySet * b = ksNew (0, KS_END);
        kdbGet (handle, a, keyA);  // (A)
        kdbGet (handle, b, keyB);  // (B)
        // we need to change *something*, otherwise the backend would assume nothing changed
        // and never even execute all of kdbSet
        ksAppendKey (a, keyNew ("user:/a/test", KEY_VALUE, "hi", KEY_END));
        kdbSet (handle, a, keyA);  // (C)
        ksDel (a);
        ksDel (b);
        keyDel (keyA);
        keyDel (keyB);
        keyDel (root);
        kdbClose (handle, 0);
        return 0;
}
```

- At point `(A)` the plugins internally store the data below the parent `keyA`.
- At point `(B)` the same thing happens for the data below the parent `keyB`. This **replaces** the data from `(A)`.
- At point `(C)` plugins that do change tracking need to generate a changeset. The changeset will be calculated between the stored data and the new data in the KeySet `a`. Because the stored data is now from `(B)` and **not** from `(A)`, this changeset will be wrong. The calculation will wrongly assume everything in KeySet `b` was removed and everything in KeySet `a` is newly added.

Right after `(B)`, the two keysets are filled with the following values:

a	b
user:/a/brightness	user:/b/background
user:/a/saturation	user:/b/foreground

If a plugin calculates a changeset during `kdbSet` by comparing to the `KeySet` from the last `kdbGet` (as described above), the plugin will wrongly calculate the following changeset at `(C)`:

Added Keys	Removed Keys	Modified Keys
user:/a/brightness	user:/b/background	
user:/a/saturation	user:/b/foreground	
user:/a/test		

This is obviously wrong. Looking at the example above, the changeset should only contain `user:/a/test`, as nothing else has been changed in keyset `a`. So the correct changeset should look like:

Added Keys	Removed Keys	Modified Keys
user:/a/test		

The erroneous behavior can be noticed by the output of `dbus-monitor`:

```
signal time=1666993066.749997 sender=:1.364 -> destination=(null destination) serial=2
       path=/org/libelektra/configuration; interface=org.libelektra; member=KeyAdded
   string "user:/a/brightness"
signal time=1666993066.750161 sender=:1.364 -> destination=(null destination) serial=3
       path=/org/libelektra/configuration; interface=org.libelektra; member=KeyAdded
   string "user:/a/saturation"
signal time=1666993066.750551 sender=:1.364 -> destination=(null destination) serial=4
       path=/org/libelektra/configuration; interface=org.libelektra; member=KeyAdded
   string "user:/a/test"
signal time=1666993066.750628 sender=:1.364 -> destination=(null destination) serial=5
       path=/org/libelektra/configuration; interface=org.libelektra; member=KeyDeleted
   string "user:/b/background"
signal time=1666993066.750866 sender=:1.364 -> destination=(null destination) serial=6
       path=/org/libelektra/configuration; interface=org.libelektra; member=KeyDeleted
   string "user:/b/foreground"
```

The same behavior is present in the `logchange` plugin. Notice its output onto `stdout`:

```
added key: user:/a/brightness
added key: user:/a/saturation
added key: user:/a/test
removed key: user:/b/background
removed key: user:/b/foreground
```

As can be seen, the change tracking within the `dbus` and `logchange` plugins wrongly calculates everything

below `user:/a` are new keys that have been added, and everything below `user:/b` was removed.

209.2 Constraints

1. As far as possible, we must check for illegal sequences and raise an error when an illegal sequence occurs

209.3 Assumptions

1. 2. 3.

209.4 Considered Alternatives

1. Document which sequences are allowed without raising errors in all illegal cases. Least-effort approach, but at least offers some transparency for developers. Developers utilizing Elektra can still ignore it, and may cause problems in certain setups without knowing.

   (Violates constraint 1)

2. Enforce that the `parentKey` used in `kdbSet` is below the one used in the last `kdbGet`. If this is not the case, `kdbSet` will abort and report an error in `parentKey`. Developers might still wrongly mix the sequences, but they will get an error and have to fix it.

   - Adding a `Key * lastGetParent` to `struct _KDB`
   - Doing `keyCopy (handle->lastGetParent, parentKey, KEY_CP_NAME)` in `kdbGet`
   - And checking `keyIsSameOrBelow (handle->lastGetParent, parentKey) == 1` in `kdbSet`

```
KDB * kdb = ...;
Key * keyA = ...;
Key * keyB = ...;
KeySet * a = ...;
KeySet * b = ...;
// These sequences will work fine
assert (kdbGet (kdb, a, keyA) == 0);
assert (kdbSet (kdb, a, keyA) == 0);
assert (kdbGet (kdb, b, keyB) == 0);
assert (kdbSet (kdb, b, keyB) == 0);
// These will not
assert (kdbGet (kdb, a, keyA) == 0);
assert (kdbGet (kdb, b, keyB) == 0);
assert (kdbSet (kdb, a, keyA) == -1); // Error! keyA is different from last used key in kdbGet (keyB)
```

1. Enforce that `ks` used in `kdbSet` is the same as in the last `kdbGet` Similar to (2), but we check for the pointer of the `KeySet` instead of having a copy of the parent key, which saves some memory.

   - Adding a `KeySet * lastGetKs` to `struct _KDB`
   - Doing `handle->lastGetKs = ks` in `kdbGet`
   - And checking `lastGetKs == ks` in `kdbSet`

2. Change the API so that a single instance of KDB can only contain a single instance of a `KeySet`. This solution completely eliminates problematic sequences.

```
// same as before but uses handle->data instead of the ks argument
int kdbGet (KDB * handle, Key * parentKey);
// same as before but uses handle->data instead of the ks argument
int kdbSet (KDB * handle, Key * parentKey);
KeySet * kdbData (KDB * handle) {
    return handle->data;
}
```

1. Allow arbitrary sequences and let each plugin deal with it on a case-by-case basis. This alternative would put most of the burden onto the plugin authors. Depending on what the plugins do, every plugin may also need to deep-dup every keyset of every parent it ever receives via `kdbGet`. This will increase memory usage. However, this could be paired with the COW semantics so the memory toll would not be that big of a deal. The biggest problem with this approach would be the unnecessary duplication of the non-trivial change tracking algorithm.

2. Don't restrict sequences further and provide a common framework to handle change tracking correctly.

   As the problem has only been observed with plugins doing their own change tracking, we could provide a general change tracking framework within Elektra. This way, we have only one such algorithm in a central place, and plugin authors don't have to think about the sequences their plugins are called by developers.

   This approach can also be paired with COW semantics, so that memory toll will be kept low. A separate decision for change tracking is currently in progress.

   Should we observe this problem with use cases other than change tracking, we can provide general frameworks for those too.

209.5 Decision

209.6 Rationale

209.7 Implications

-
-
-

209.8 Related Decisions

- Change Tracking
- Internal KeySet Cache
- [](doc_decisions_0_drafts__README_md)

209.9 Notes

- Issue #4514 uncovered a problem in the current change tracking approach relating to sequences of `kdbGet` and `kdbSet`
- Issue #4512 explored some possible solutions

Chapter 210

Plugin Contract Function

210.1 Problem

Contract for a plugin is currently retrieved by calling the plugin's `get` function with a special `parentKey`. This makes the `get` function awkward to write. You must always first check for the special `parentKey`. It also makes the `get` function mandatory, even if the plugin doesn't implement any real `get` functionality.
Additionally, the actual public API exported from shared library to make it a plugin is the `ELEKTRA_PLUGIN_EXPORT` function. This function always calls `elektraPluginExport`, which allocates and half initializes a `struct _Plugin`. Specifically, only the `name` field and the `kdb*` function pointers are set. This same information is also provided via the contract. The function pointers are all listed in `system:/elektra/modules/<plugin>/exports/*` and the `<plugin>` part of the name is the `name` of the plugin.

210.2 Constraints

210.3 Assumptions

210.4 Considered Alternatives

1. Create a separate `kdbContract` that returns the contract of a plugin.

```
KeySet * ELEKTRA_PLUGIN_FUNCTION(contract) ();
// OR
void ELEKTRA_PLUGIN_FUNCTION(contract) (KeySet * ks);
```

This solves the problem of the awkward `get` function, but it does not address the issue of the duplicate information in `ELEKTRA_PLUGIN_EXPORT`.

1. Change the `ELEKTRA_PLUGIN_EXPORT` to return the contract of the plugin instead of creating half a `struct _Plugin`.

```
KeySet * ELEKTRA_PLUGIN_EXPORT ();
// OR
void ELEKTRA_PLUGIN_EXPORT (KeySet * ks);
```

This also removes the duplicate information inside the plugin.

1. (combined with 2) Change what the `KeySet * modules` stores.

   Currently, `modules` stores data dependent on how modules are loaded. When dynamic linking is used, the keys store the handle from `dlopen` and a pointer to the `ELEKTRA_PLUGIN_EXPORT` function. With static linking, the keys store a custom struct with function pointers.

   Instead, we would now store the contracts directly in `KeySet * modules`.

   The implementations of `elektraModulesLoad` would simply find the correct function and call it.

```
typedef void (*elektraPluginExportFn) (KeySet * ks);
void elektraModulesLoad (KeySet * modules, const char * name, Key * errorKey)
{
     Key * moduleKey = keyNew ("system:/elektra/modules", KEY_END);
```

```
keyAddBaseName (moduleKey, name);
if (ksLookup (modules, moduleKey, 0) != NULL)
{
        // already loaded
    return;
}
    elektraPluginExportFn exportFn;
    // [...] find exportFn for plugin
    exportFn (modules); // inserts the contract with all the exported functions
    if (ksLookup (modules, moduleKey, 0) == NULL)
{
        ksAppendKey (modules, moduleKey);
}
    else
    {
        keyDel (moduleKey);
    }
}
```

Note to avoid allocated and deleting a temporary `KeySet *` the `void ELEKTRA_PLUGIN_EXPORT (KeySet * ks)` API works best here.

210.5 Decision

210.6 Rationale

210.7 Implications

210.8 Related Decisions

- Commit Function
- Plugin Struct

210.9 Notes

Chapter 211

Readonly Keynames

211.1 Problem

It might be a good idea to prevent changes to the keyname after creation. Currently, the name of a key is automatically made read-only, when it is currently part of or at some point in the past was part of at least one `KeySet`.
There are discussions about making key names writable again, when the key is removed from all `KeySet`s.
There may also be situations where changing the name of a key after its creation is required. In some situations, it may be wise to reuse a key instead of deleting it and creating a new one.

211.2 Constraints

1. `keyAddName` et al. still have to work up to a certain point 2. 3.

211.3 Assumptions

1. 2. 3.

211.4 Considered Alternatives

211.4.1 Separate API for keynames

Use the proposed `ElektraBuffer` struct to create a separate API for keynames independent of the `Key` API.

211.4.2 Read-only keynames

The key name should be permanently read-only after creation.
TBD: How would we dynamically create keynames with this approach?

211.4.3 Re-entrant lock for the key name

Since we assume a single threaded context, this can be implemented as a simple counter.

```
struct _Key {
    // [...] other stuff
   uint16_t nameLock; // if zero, name is writable, otherwise name is readonly
};
void keyLockName (Key * k) {
    k->nameLock++;
}
void keyUnlockName (Key * k) {
    if (k->nameLock == 0) return;
    k->nameLock--;
}
void ksAppendKey (KeySet * ks, Key * k) {
    keyLockName (k);
    // [...]
}
void ksRemove (KeySet * ks, elektraCursor cursor) {
    keyUnlockName (ks->array[cursor]);
```

```
    // [...]
}
```

211.4.4 Alternative C

211.5 Decision

211.6 Rationale

211.7 Implications

-
-
-

211.8 Related Decisions

- [](doc_decisions_0_drafts__README_md)
- [](doc_decisions_0_drafts__README_md)
- [](doc_decisions_0_drafts__README_md)

211.9 Notes

- Issue 2202 talks about how unexpected it is that keys will be readonly once in a keyset, but they don't get unlocked when removing them from a keyset.

Chapter 212

Transformations

212.1 Problem

Certain plugins, e.g. `rename`, perform transformations on keys and keysets within Elektra. Those transformations include, but are not limited to:

- Changing the names of the keys back and forth
- Changing values back and forth for normalization, e.g. `true` -> `1`, `1` -> `true`

While these features are useful, they do create feature-interaction problems. More specifically, problems have been observed in conjunction with the following (overlapping) types of plugins:

- notification plugins
- plugins that do change tracking

The problem, in general, can be described as: Which representation of the KDB should be used for notifications/change tracking?
We differentiate between:

- *persistent* name/value/metadata: How it is actually stored, i.e. the state returned by and passed to the `storage` plugins.
- *transient* name/value/metadata: How it is at runtime, i.e. what is returned by `kdbGet` and passed to `kdbSet`.
- *intermediate* name/value/metadata: Any state inbetween the two.

212.1.1 Observed problems with changing key names

Suppose there is a plugin that changes key names. It converts to lowercase for the runtime representation, and to uppercase for the stored representation. The plugin is executed in the post-storage phase for the `get` operation and the pre-storage phase for the `set` operation. This results in the keys being in UPPERCASE in the configuration files, but they are presented in lowercase to other plugins and applications using the Elektra API.
For example, here is a configuration file with a hypothetical format:

```
/DISPLAY/BRIGHTNESS = 100
```

As can be seen, the keys are in UPPERCASE within the configuration file. In Elektra keys are case-sensitive. As operations on keysets such as `ksLookup` operate with *runtime* data after the post-storage phase, `kdb get /DISPLAY/BRIGHTNESS` will fail. For Elektra, the key `/DISPLAY/BRIGHTNESS` does not exist, as the `rename` plugin transformed this into the lowercase `/display/brightness`.
This leads to problems with the notification plugins. As notification plugins are executed after the post-storage phase of the `set` operation, they will receive a keyset with the already transformed keys. In this example, the notification plugins will receive all-UPPERCASE keys, and send out notifications with those all-UPPERCASE keys. An application listening to those notifications will not be able to query Elektra for those keys, as for Elektra those UPPERCASE keys do not exist.
Apart from the problems with notifications, the way key name changing plugins work also breaks change tracking in plugins like `dbus`. This is because key names are read-only when they are contained in a `KeySet`. In order

to change the name of a key, such a plugin has to create a new key with the changed name, and delete the key with the old name. The `dbus` plugin implements change tracking by checking the 'key needs sync' flag instead of comparing the values. As new keys by design have the 'key needs sync' flag set, the plugins that implement change tracking via the flag will always erroneously detect transformed keys as changed.

212.1.2 Observed problems with changing values (normalization)

Suppose we have a plugin that changes the value of a key to the hard coded value `1` during the storage phase of the `set` operation. If a plugin does change tracking, this will lead to false positives.

```
user:/limits/openfiles = 1
```

If the user changes the value, e.g. using `kdb set user:/limits/openfiles 23`, plugins will observe the new value `23`. The value-changing plugin, however, will reset that value back to `1`. So in practice, the configuration has not been changed. Plugins relying on change tracking plugins (e.g. notification plugins) will however think that it has.

212.2 Constraints

1. We want to enable some kind of transformations
2. Renaming keys should be possible, at least in storage plugins 3.

212.3 Assumptions

1. False positives for change tracking algorithms are only a minor problem. False positives are that changes are detected, even though nothing changed.
2. There is no reason to modify or delete existing `meta:/` keys.
3. Newly generated `meta:/...` keys can
   - either stay and get permanently stored during `kdbSet`
   - or be written as `meta:/generated/...`. The `meta:/generated/...` metakeys are never stored and are automatically removed during `kdbSet` before `storage` is called.

212.4 Considered Alternatives

212.4.1 Enable setting and unsetting of the <tt>keyNeedsSync</tt> flag

Require plugins that rename keys to remove the `keyNeedsSync` flag. This would require making the 'key sync' APIs public. Letting user code modify these internal flags may lead to serious bugs. This does not solve the problem of changing values.

212.4.2 Use change tracking to replace the <tt>keyNeedsSync</tt> flag

Eliminate the need of the `keyNeedsSync` flag by utilizing the planned change tracking API. This does not solve the problem of changing values and false positives.

212.4.3 Store original names and values in metadata.

For key name transformations require that transformation plugins set a metakey (e.g. `meta:/elektra/runtimename`) with the runtime name before they do any transformations in the `kdbSet` phase. This *must not* be done if this metakey already exists, i.e. another plugin already tranformed it beforehand. This allows notification and change tracking functionality to work determine and work with the runtime name of the key.
A similiar thing was already attempted for values, i.e. `meta:/origvalue`.

212.4.4 Create an API for transformations

Plugins must use a special function to transform key names, e.g.:

```
typedef const char * (*ElektraNameTransform)(const Key *);
int elektraApplyNameTransform (KeySet * ks, ElektraNameTransform transform);
```

How the `elektraApplyNameTransform` function marks the original name is an internal implementation detail. May be as `meta:/` or something else entireley.
Something similar could be done for the value of a key as well.

212.4.5 Introduce a new phase for transformations

We could also introduce a new phase between before/after storage exclusively for transformations. Then we can just do a "fake" call to that phase to get back the transient names for change tracking.

212.4.6 Call value transformation plugin(s) when <tt>keySetValue</tt> / <tt>keySetString</tt> is called

After a value has been set by the user, call the transformation plugin. We could store those callbacks as metakeys, i.e. `meta:/generated/transformation/value/callback/#1`. Alternatively, we could implement a linked list or other data structure for the callbacks.

```
struct callback {
    callback_fn function;
    struct callback * next;
}
```

There will be 3 possibilities where transformations take place:

1. For keys with a registered callback:
   - Within `keySetValue` / `keySetString` the callbacks are called.
2. For new keys, i.e. keys without a callback:
   - Within `kdbSet`, as it is now.
3. For new keys that override keys with an intact callback:
   - Within `ksAppend` and/or `ksAppendKey` the callbacks are copied from the original key and then executed on the new key.

```
kdbGet (kdb, ks, parent);
// ks contains two keys:
//  - user:/limits/openfiles
//  - user:/background/color
Key * background = ksLookupByName(ks, "user:/background/color", 0);
keySetString(background, "#ff00ff"); // (1) -> callbacks are called.
ksAppendKey (ks, keyNew ("user:/my/key", KEY_VALUE, "1234", KEY_END); // (2) -> transformation will happen
       inside kdbSet
ksAppendKey (ks, keyNew ("user:/limits/openfiles", KEY_VALUE, "23", KEY_END); // (3) -> replaces existing
       key, callbacks of existing key are copied and then executed
kdbSet (kdb, ks, parent);
```

We also need to provide a simple API to plugins to register such callbacks for a key.
As the transformations for existing keys will be applied before `kdbSet`, this will elimate false positives in change-tracking.
A drawback to this solution is that it adds some complexity to `libelektra-core`.

212.5 Decision

212.6 Rationale

212.7 Implications

-
-
-

212.8 Related Decisions

- Hooks
- Change Tracking
- Operation Sequences
- Internal Cache

212.9 Notes

- Issue #404 - dbus and rename plugin do not work together
- Issue #955 - dbus: non-UTF8 key names
- Issue #3626 - origvalue

Chapter 213

Global Validation

213.1 Problem

Links (`fallback` and `override`) and validation constraints might point to keys not loaded with the respective `kdbGet`.

213.2 Constraints

- no workaround such as `kdb get -a`

213.3 Assumptions

- it is too expensive to always load all keys (as some backends are not cacheable)

213.4 Considered Alternatives

- global plugin can register additional backends to load
- split `kdbGet` into multiple steps:
  1. do `kdbGet` on `system:/elektra` to update mount points
  2. first do `kdbGet` on the `spec:/`-namespace
  3. then calculate which backends are needed
  4. then fetch all backends as needed
- split loops in `kdbGet` not only according placements but also according namespace (get spec first)

213.5 Decision

Implementation delayed after 1.0.
For now, the specification can only refer to what applications request by `kdbGet`.

213.6 Rationale

- problems in implementing to always get everything
- it would make the parameter `parentKey` to `kdbGet` useless

213.7 Implications

213.8 Related Decisions

- Internal Cache

213.9 Notes

- Was part of `improved KDB logic #1291`

Chapter 214

Plugin Struct

214.1 Problem

`struct _Plugin` contains some unnecessary data.

214.2 Constraints

214.3 Assumptions

214.4 Considered Alternatives

Note: Not all of these are alternatives, most of them could be combined.

1. Replace `KeySet * global` and `KeySet * modules` with `KDB * kdb`.

   This would require changing the `elektraPluginOpen` API to take a `KDB * kdb`, or a `Plugin * existing` from which we extract the `KDB *` (the first plugin could use a stack allocated `Plugin` with just `KDB *` set).

2. Replace all the `kdb*` function pointers and `char * name` with a `KeySet * contract`.

   The contract also contains the function pointers, so this avoids duplication and should reduce memory usage. This could potentially affect runtime performance, because instead of having a pointer directly, we need to first do a lookup in a keyset.

3. Remove `size_t refcounter`.

   Plugins are never shared between mountpoints. There is also no way of increasing `refcounter` via the API.

4. (Based on all above) Change how `struct _Plugin` is used.

   This is the most comprehensive change, it combines all the above to remove as much duplication as possible. It consists of these parts:

   - Create a `struct _PluginData` that will be used in `struct _BackendData` instead of `struct _Plugin`.

     ```c struct _PluginData { KeySet * config; // config of plugin void * data; // private data of plugin char * name; // name of the plugin }; ```

     The `name` is needed to get the correct functions from `kdb->contracts` (see below).

   - Change `struct _Plugin` to the following and use stack allocated instances created when calling a plugin function.

     ```c struct _Plugin { KeySet * config; // copied from struct _PluginData void * data; // copied from struct _PluginData KDB * kdb; // pointer to KDB instance calling this plugin right now, may be copied from other plugin calling this plugin }; ```

- Add a `KeySet * contracts` to `struct _KDB`. This contains all the contracts for all the plugins currently loaded.

  To call a plugin function you need an existing `Plugin *` from which the `KDB * kdb` will be taken. Then the function will be looked up in `kdb->contracts`. Inside `libelektra-kdb` the existing `Plugin *` is faked with a stack allocated `struct _Plugin` containing only a `KDB *`.

  > **Note** If the changes to `KeySet * modules` from the Plugin Contract FunctionPlugin Contract Function"" decision are implemented, then `kdb->contracts` becomes `kdb->modules`.

5. Introduce a new `KeySet * shared` for sharing data among plugins of a single backend.

   This is similar to `KeySet * global`, but not shared between different mountpoints. The `KeySet * shared` would be a copy of the one in `struct _BackendData` (also new field), which owns the `KeySet`.

   This solves the issue that `libelektra-kdb` must currently use `KeySet * global` to communicate extra data like the plugins belonging to a backend (for access in backend plugins).

   When combined with the stack allocated `struct _Plugin` from option 4, this would not affect memory usage as much as otherwise. In that case, it would only be one `struct _KeySet` per backend, when otherwise we would also add one pointer for every loaded plugin instance.

214.5 Decision

Don't do anything for 1.0.

214.6 Rationale

214.7 Implications

214.8 Related Decisions

- Plugin Contract Function
- Commit Function

214.9 Notes

Chapter 215

Plugin Variants

215.1 Problem

Some plugins are generic in the sense that they cannot fully define their contract statically. (Note: if they can decide it statically, you should prefer compilation variants.) Instead their contract is based on their configuration. We will call every combination of plugins+configuration, where we get a different contract **plugin variant**.
The current issue is that there is no way to enumerate possible plugin variants as needed to list all functionality of Elektra.

215.2 Constraints

- User Customizability:
  - Users must be able to add new plugin variants
  - Users should be able to suppress plugin variants
- Modularity:
  - The plugins themselves yield a list of plugin variants
  - The code defining variants is part of the plugin's code
  - To get a full list of all variants of all plugins, one simply has to iterate over all plugins and ask the plugins for possible variants.
- libtool should provide a convenience layer for easy access of all plugin variants. In this case there should also be a performance optimization, that plugins do not need to be reopened required multiple times.

215.3 Assumptions

- The number of variants needs to be bounded: plugin variants need to be fully enumerable

215.4 Considered Alternatives

A specification of the plugin's configuration and a tool that can enumerate all possible essential configurations. Problems here are:

- Plugins might need to self-reference (a validation plugin might have plugin variants, too)
- Elektra's specification language was not complete nor consistent at point of writing
- Does not fit with the `checkconf` (`see here`) approach.

215.5 Decision

Implementation delayed after 1.0.
Best implementation candidate was:

1. Provide a function `int genconf (KeySet * ks, Key * errorKey)` where `ks` is filled with a list of all variants with the essential configuration (subkeys `config`) and the changed parts of the contract (subkeys `infos`).
2. Keys defined in `system:/elektra/plugins/<plugin>/variants/<variantname>` have the same content, but take precedence. If a variant with the same name is defined, only `config` or `infos` from `genconf` are considered if they are not mentioned in `system:/elektra/plugins/variants`.
3. If the bool key `override` (for a plugin or a variant) is true, it will be overwritten (content of `genconf` ignored, but instead a plugin or variant as given is created).
4. If the bool key `disable` (for a plugin or a variant) is true the plugin or a variant of the plugin will not be available.
5. Empty `config` and `infos` mean:
   - `config`: The "default variant" (without any parameter) should be also available (has useful functionality)
   - `infos`: It is not possible to determine the changes of the contract, the user need to instantiate the plugin and enquiry the contract.

215.5.1 Example

`genconf` for augeas yields:
```
system:/access
system:/access/config
system:/access/config/lens = Access.lns
system:/access/infos
system:/access/infos/provides = storage/access
system:/aliases
system:/aliases/config
system:/aliases/config/lens = Aliases.lns
system:/aliases/infos
system:/aliases/infos/provides = storage/aliases
```
`genconf` for python might yield:
```
user:/configparser/config
user:/configparser/config/script = python_configparser.py
```
The user:/admin specifies:
```
system:/elektra/plugins/jni/disable = 1
system:/elektra/plugins/augeas/variants/access
system:/elektra/plugins/augeas/variants/access/disable = 1
system:/elektra/plugins/augeas/variants/aliases
system:/elektra/plugins/augeas/variants/aliases/infos
system:/elektra/plugins/augeas/variants/aliases/infos/status = 10000
system:/elektra/plugins/python/variants/configparser
system:/elektra/plugins/python/variants/configparser/override = 1
system:/elektra/plugins/python/variants/configparser/config
system:/elektra/plugins/python/variants/configparser/config/script = mybetter_configparser.py
```
As result we get:

1. `access` of plugin `augeas` is not available
2. `aliases` of plugin `augeas` as defined from `genconf`, but with changes in contract (`infos/status`)
3. `configparser` of plugin `python` is completely redefined (result from `genconf` will not be considered) but it will be considered as specified.
4. the plugin `jni` will not be available

To have a space-separated simpleini one would use:
```
system:/elektra/plugins/simpleini/variants/spacesep
system:/elektra/plugins/simpleini/variants/spacesep/config
system:/elektra/plugins/simpleini/variants/spacesep/config/format = "% %"
```

215.6 Rationale

- The `genconf` API was chosen to be consistent with `checkconf`. The pluginhandle might be missing for some scenarios. If it is needed, we should change both APIs for consistency reasons.
- The very verbose (deeply nested) subkeys (and names instead of arrays) provide better customizability and extensibility.

215.7 Implications

- `genconf` needs to be implemented for the plugins:
  - augeas
  - jni
  - lua
  - python
- `PluginDatabase` needs an extension to list all plugin variants
- `kdb plugin-list` should be able to list all variants, e.g. like: `augeas lens=Access.lns` so that a user can copy and paste this for the `kdb mount` command.

215.8 Current state

Currently only overrides of plugin configuration (`config/...`) is possible, overrides of other plugin information (contract information) does not work yet.
It is also not possible to add additional information to a variant, only overrides work. E.g.

```
system:/elektra/plugins/augeas/variants/aliases
system:/elektra/plugins/augeas/variants/aliases/override = 1
system:/elektra/plugins/augeas/variants/aliases/config
system:/elektra/plugins/augeas/variants/aliases/config/lens = Aliases.lns
system:/elektra/plugins/augeas/variants/aliases/config/otherparam = 0
```

works, while

```
system:/elektra/plugins/augeas/variants/aliases
system:/elektra/plugins/augeas/variants/aliases/config
system:/elektra/plugins/augeas/variants/aliases/config/otherparam = 0
```

gets ignored.

215.9 Related Decisions

- CMake Plugins

215.10 Notes

Discussions took place here.

Chapter 216

Validation

216.1 Problem

Validation plugins operate as independent blackboxes. For every backend each mounted validation plugin iterates over the whole keyset, checks every key for its trigger metakey, and validates the key.
Currently all needed validation plugins need to be specified at mount-time; if additional validation is required, the backend has to be remounted with the required plugins and plugin configuration.
If validation of a key fails, each plugin decides on its own how to handle the issue and proceed in ways that might be different from what is expected or desired.

216.2 Constraints

216.3 Assumptions

While plugins should always fail and return an error if validation fails on kdbSet, there are several different requirements for what should happen on kdbGet and handle problems e.g.

- only issue warnings

  we want to read the whole configuration, but issue warnings if keys fail to validate instead

  problems are handled external by an application, user, ...

- remove invalid keys

  we want to read the whole configuration, but drop invalid keys

  invalid keys might be replaced by default values, requested from the user, ...

- fail with error

  we only want to read valid configurations, and fail with an error if the configuration is invalid

216.4 Considered Alternatives

- Extend validation plugins to allow us to specify what should happen if a key fails to validate
- Export a validation function that allows us to use an additional plugin to decide what should be done

216.5 Decision

Don't do anything for 1.0.
Later, maybe introduce a wrapper plugin to iterate over the keyset and delegate the validation of each key to the corresponding validation plugin.

216.6 Rationale

- Validation plugins don't need to know what should be done if the validation fails
- We can run multiple validations on every key and improve error messages
- Different ways of handling errors only need to be implemented once

216.7 Implications

Validation plugins have to export their validation routine
`static int validateKey(const Key * key, const Key * errorKey)`
Returning 1 if validation succeeded, 0 on failure

216.8 Related Decisions

216.9 Notes

Chapter 217

CMake Spec

217.1 Problem

The compilation variants of plugins blow up the number of plugins. Additionally there is the concept of default storage + resolver that is needed for bootstrapping plugins.

217.2 Constraints

- full default resolver need to be different from other default resolver for testing
- there is no standard resolver, they always should state their configuration

217.3 Assumptions

- keep it not too difficult to configure, even though most people will go for the defaults

217.4 Considered Alternatives

- many CMake variables for every case KDB_DEFAULT_STORAGE, KDB_DEFAULT_RESOLVER for dynamic and static cases
- Have a PLUGINS field that state what is the default dynamic and static plugin, e.g. ! for storage ? for resolver ds?resolver_b_u_b;ds!dump

217.5 Decision

Rejected: keep default plugins as-is

217.6 Rationale

217.7 Implications

217.8 Related Decisions

217.9 Notes

Chapter 218

Pub/Sub Communication

218.1 Problem

To develop a Web UI, we need to be able to remotely configure Elektra via a network socket.
The idea is to use a Pub/Sub concept to synchronize actions which describe changes in the Elektra state.

218.2 Constraints

- We need to be able to synchronize all changes in Elektra with the Web UI.
- This needs to be done via a network socket due to limitations of the Web.
- That means we need to run an Elektra daemon (`elektrad`) to be able to connect to Elektra at any time.

218.3 Assumptions

218.4 Considered Alternatives

- ZeroMQ: small and popular library for pub/sub
- nanomsg: from the same author as ZeroMQ, even smaller - https://nanomsg.org/documentation-zeromq.html
- redis: requires a running redis server
- kafka: seems too big for Elektra
- ZeroMQ with JSMQ.

218.5 Decision

Don't use pubsub at all.
Instead use REST API implemented by `elektrad`.

218.6 Rationale

nanomsg sounds interesting, but isn't as popular as ZeroMQ, which is why there are no browser JS bindings available (only Node.js, which cannot be easily used for the Web UI).

218.7 Implications

218.8 Related Decisions

218.9 Notes

Chapter 219

Null Pointer Checks

219.1 Problem

Currently all functions do proper argument checking which might degrade performance.

219.2 Constraints

219.3 Assumptions

219.4 Considered Alternatives

- Removing all null pointer checks and do assert on debug code
- Removing some null pointer checks
- adding functional high-level methods that avoid most null pointer checks

219.5 Decision

Rejected (keep checks) due to time constraints

219.6 Rationale

- also passing NULL pointers should have consistent, defined behavior

219.7 Implications

- ABI, API

219.8 Related Decisions

219.9 Notes

- Benchmarks needed

Chapter 220

Key Name

220.1 Problem

Often a `Key` argument is used when you just need a key name. This is because with a `Key` we know the name is valid and we get an unescaped name. Using a `Key` here makes the API a bit confusing.
There could be a richer API for manipulating key names without relying on the escaped name (e.g. concatenating two full key names). With the current situation, all these functions would need to be part of the API for a key. Adding such functions to the key API is certainly not minimal.

220.2 Constraints

220.3 Assumptions

220.4 Considered Alternatives

- add separate `struct KeyName`

220.5 Decision

Continue keeping 3 classes: `Key`, `KeySet` and `KDB`.

220.6 Rationale

- A minimal `Key` ideally only consists of a key name and the goal is to keep `Key` small, so introducing `Key↩ Name` would go the wrong direction.

220.7 Implications

- Thus, operations working on key names, directly use `Key` as argument.
- The key name is actually, in every implementation, a plain string. This is also required, because implementations must use the same memory layout.

220.8 Related Decisions

- Null
- Namespace and Name of Keys

220.9 Notes

Text partly copied from @kodebach https://github.com/ElektraInitiative/libelektra/pull/4201#pullre

Chapter 221

Vendor Spec

221.1 Problem

Vendors (distributors, administrators) might want to modify the specification. `gsettings` has a similar feature.

221.2 Constraints

There are many constraints in providing such a feature because it is possible to get an inconsistent or unusable specification.

221.3 Assumptions

Developers who elektrify their applications do care about good integration and being administer friendly.

221.4 Considered Alternatives

- implementing a new namespace that gets merged
- merge specification files during installation

221.5 Decision

As found out during implementation of specload, only a very limited subset can be modified safely, e.g.:

- add/edit/remove `description`, `opt/help` and `comment`

221.6 Rationale

- Elektra wants to reduce fragmentation, and vendor specific changes obviously is a severe kind of fragmentation
- providing vendor overrides/fallbacks might be an excuse to not provide or general override/fallback features

221.7 Implications

Provide means for a single specification to be very good integrated in every system.

221.8 Related Decisions

221.9 Notes

Chapter 222

Error Handling

222.1 Problem

There are ambiguous cases where the same return value can refer to multiple problems:

- name modifications which can be either invalid name or locking the key name
- getting values of (non-)binary keys

222.2 Constraints

We also have to consider that many parts of the API very constrained in what values they can use for error codes. If a function returns a pointer the only possible error value by definition is NULL.

222.3 Assumptions

222.4 Considered Alternatives

- Change return types for many functions across the API

222.5 Decision

- Update documentation in `doc/dev/error-*` and link to them in the documentation for the module `kdb`
- Add second channel for getting information about errors
- Return error codes directly from functions where failures are expected, e.g. `kdbGet`, `keySetName`
- Harmonize return values from all functions and move error reporting to second channel

222.6 Rationale

This simplifies and unifies error reporting across all functions. It also introduces a default way for the developers to handle error reporting. Developers of bindings for Elektra can rely on the existing error reporting feature without having to introduce custom error messages, that can vary wildly between different bindings.

222.7 Implications

222.8 Related Decisions

- Binary metadata vs flag #4194

222.9 Notes

Chapter 223

Error Semantics

223.1 Problem

While we have a classification of errors and warnings, it remains unclear when plugins actually should emit errors and warnings.

223.2 Constraints

- Should not be contradicting to specified behavior in storage plugin tutorial.

223.3 Assumptions

- Users want a uniform behavior within Elektra, so plugins must behave uniformly.

223.4 Considered Alternatives

- freedom to plugin writers
- strict rules and conformance tests for plugins

223.5 Decision

Provide guidelines in the form as tutorials, covering:

- prefer errors to warnings
- that any not understood metadata (e.g. types), should lead to an error
- that wrong specifications, like `kdb meta-set /tests/ipaddr/ipv4 check/ipaddr ipv8` should be rejected
- if the value does not confirm **exactly** to the specified type, an error should be emitted (e.g. only `0` or `1` as boolean)
- anything else that is beyond the capabilities of a plugin (not implemented), should lead to an error

Violations against these guidelines can be reported as bug and then either:

- the bug gets fixed
- the plugin get a worse `infos/status` but still get shipped with 1.0
- the plugin gets removed

223.6 Rationale

It is easier for developers if there are clear expectations on how a plugin should behave. And it is much easier for overall Elektra if there is more consistency.

223.7 Implications

- more checks&errors in storage plugins are needed

223.8 Related Decisions

- Metadata in Spec Namespace
- Capabilities
- Boolean

223.9 Notes

- `Issue #1511`

Chapter 224

keyString() return value

224.1 Problem

When using keyString() on empty / binary values the return values are the literal strings (null) / (binary). This seems very awkward and unintuitive from a user's perspective.

224.2 Constraints

Because the return value of the function is `const char*`, there are not so many possibilities for a change to this behavior, as every possible return value - except for a NULL pointer - could just be the value of the string of the Key.

224.3 Assumptions

On one hand always returning a string allows things like `strlen(keyString(k)) == 0` to check for an empty string. But at the same time people might expect that `strlen(keyString(k)) < MAX_LEN` also works. But this again might silently break code down the line, if `k` is binary and `strlen("(binary)") < MAX_LEN`.

224.4 Considered Alternatives

An alternative would be to always store a zero byte after all key values, even if they are binary (might be done already). Then we can safely return `k->data.c == NULL ? "" : k->data.c`. It may contain incomplete data and the `MAX_LEN` problem from above still applies, but there are no segfaults and you don't get return values that have nothing to do with the actual data.

224.5 Decision

- `key == NULL` return 0, error code via second channel
- `key->value == NULL` return 0, error code via second channel
- `key == <binary>` return 0, error code via second channel
- everything else as is

224.6 Rationale

- 0 seems like the most intuitive value to return in case of an error, although this introduces the possibility of segfaults for users of the library. printf doesn't cause segfaults anymore, which improves this problem a lot.
- With the introduction of a second channel for reporting errors, users can check the error messages in case of segfaults - which alleviates this issue. The first thing in case of an error should be checking the error message.

224.7 Implications

224.8 Related Decisions

224.9 Notes

Chapter 225

Spec's Expressiveness

225.1 Problem

Currently, you can easily come into wrong assumptions that something would work in the specification.
We need to find minimal requirements to implement a sane spec plugin.

225.2 Constraints

225.3 Assumptions

225.4 Considered Alternatives

225.5 Decision

- no defaults for `sw/_/key` specifications (default will not work for `ksLookup(/sw/sthg/key)`)
- plugins are not allowed to create keys (may change in future; depends on plugin positions)

The spec plugin should yield errors when it detects such situations.

225.6 Rationale

225.7 Implications

225.8 Related Decisions

225.9 Notes

Chapter 226

Metadata in Spec Namespace

226.1 Problem

To make storage-plugins suitable for `spec` they need to be able to store all the metadata as specified in `METADATA.ini`. Most file formats do not have support for that.
If metadata is merged from different namespaces, e.g., `spec:` and `user:`; metadata from one namespace might end up in keys of other namespaces, e.g.; metadata from `spec:` might end up in `user:`.

226.2 Constraints

- We do not touch what is already described in the storage plugin tutorial.

226.3 Assumptions

- Metadata can always safely merged from `spec:/` to other namespace.

226.4 Considered Alternatives

- Store metadata in the comments like the `ini` plugin: this exposes internal metadata into the comments and can drastically affect the readability of a storage file. Comments should never be touched by a parser.
- Designate metadata to be only for `spec:/` or to only for restoring configuration file formats. Rename metadata <type> in `spec:/` to `check/<type>` or `make/array`. This way no merging of metadata would be needed and by the name alone it would be clear to which namespace it belongs.
- Do not support `array` or `type` if the underlying configuration file format does not support it. `spec-mount` could make sure that the storage plugin supports everything the underlying format needs.

226.5 Decision

Do not store metadata unrelated to the configuration file structure in any namespace except in `spec:/`.

- Trying to store any other metadata in any other namespace leads to an error. E.g. `kdb set-meta user:/data metadata_not_suitable_for_storage_plugins something` would fail (validated by `spec` plugin).
- Metadata that is designed to be stored by storage plugins to preserve configuration file structure. E.g. `comment` or `order`, might be stored in any namespace.

Sometimes, the same metadata can be used in several namespaces but with different meanings and ways of serialization, e.g. `type` and `array`:

- In `spec:/` the metadata `array=` (empty value) means "this is an array". If you give it a value e.g. `array=#4` it means "this is an array with default size X" (e.g. `#4` = size 5).

- In any other namespace `array=` means "this is an empty array" and e.g. `array=#4` means "this is an array with max index #4". `array=#4` is not stored literally but inferred.
- Either the storage plugin does not support arrays, then the metadata will be discarded on `kdbSet` but `spec` will keep on adding it for every `kdbGet`.
- Or, if the storage plugin supports arrays, the data will be serialized as array (even if the metadata comes from `spec`) and as such available in the next `kdbGet` from the storage plugin to be validated by `spec`.

Use different storage plugins, or plugins with different configurations, for the `spec:/` namespace:

- `ni`
- TOML with `meta` configuration

The `kdb mount` tool will add the `meta` plugin configuration when mounting a storage plugin to `spec:/`.

226.6 Rationale

- We do not need a storage plugin suitable for everything.
- The problems that internal metadata ends up in configuration files disappears.
- The short names `type` and `array` can be used in specs.
- Also simple storage plugins (like properties) can still be used with realistic specifications (that include array and/or type).

226.7 Implications

- We need to have different (default) plugins in `spec:/` than in the other namespaces.
- It can be confusing about which metadata we are speaking, e.g., `array` in `spec:/` has a quite different meaning to `array` in other namespaces.
- The spec:/ meaning of array= must obviously be stored. The other meaning should never be stored explicitly and instead should be inferred from the stored data. Either by the storage plugin (e.g. JSON array) or by the spec plugin.

226.8 Related Decisions

- Spec Expressiveness
- Arrays

226.9 Notes

Chapter 227

Binary

227.1 Problem

Binary

- is the only metadata that allows more values (instead of making the validation stricter)
- is the only API that modifies metadata
- creates some inconsistencies in the API (e.g. `KEY_FLAG_RO_VALUE` is linked to `KEY_FLAG_RO_META`)

227.2 Constraints

- simplify API

227.3 Assumptions

- byte-array values are a rarely used feature in configuration settings
- absence of values, i.e., null keys however, is quite common

227.4 Considered Alternatives

- flag for indicating if a value is binary

227.5 Decision

- make keys with untyped value per default (no assumption about type, i.e., byte-array)
- all current types are not binary, so type = string is the way to indicate a string doesn't
  - contain null bytes
  - is not null (not only indicator of structure without value)
  - uses some well-defined but unspecified text encoding (i.e. not necessarily ASCII or UTF-8)
- remove binary functionality from keyNew keyVNew keyIsBinary keyIsString keyValue keyGetValueSize keyString keyGetString keySetString keyGetBinary keySetBinary
- keySetRaw -> keySetValue

227.6 Rationale

227.7 Implications

- metadata can also be binary
- binary types would be possible (e.g. nullable-string)

227.8 Related Decisions

- null

227.9 Notes

Chapter 228

Functions with buffers

228.1 Problem

Currently the way functions like `keyGetName()` work is by passing a buffer with a `maxSize` and if the buffer is large enough, the value gets copied into the buffer. This leads to the user having to write a lot of surrounding boilerplate code, checking for the size of every value / name they want to copy into a buffer.

228.2 Constraints

228.3 Assumptions

228.4 Considered Alternatives

228.5 Decision

- Remove Functions:
  - keyGetName()
  - keyGetUnescapedName()
  - keyGetBaseName()
  - keyGetString()
  - keyGetBinary()
- add documentation in API documentation about life-time and add in release notes that you should use strncpy() / memcpy() instead:

```
// str values
strncpy(..., keyName (k), ...)
// binary values
memcpy(..., keyValue (k), ...)
```

228.6 Rationale

The functions clutter the API and try to replace existing builtin functionality for little to no gain. This makes the API leaner while also retaining its functionality.

228.7 Implications

228.8 Related Decisions

228.9 Notes

Chapter 229

Iterators

229.1 Problem

The internal iterator inside KeySets seems to cause more problems than it solves.

229.2 Constraints

We could embed one such iterator into KeySets, but recommend that people use an external instance (int) in new code.

229.3 Assumptions

@raphi011 made benchmarks showing that external and internal iterators have about the same speed. So it should be a safe choice to not provide the internal iterators for 1.0 and only the external instead.

229.4 Considered Alternatives

229.5 Decision

- remove all functions related to the internal iterator:
  - ksRewind
  - ksNext
  - ksCurrent
  - ksGetCursor
  - ksSetCursor
  - ksHead
  - ksTail
  - keyRewindMeta
  - keyNextMeta
  - keyCurrentMeta
- change `ksAtCursor` to `ksAt`
- add implementation / documentation / tests for the external iterator
- start using external iterators in new code
- remove documentation about internal cursor from all functions.

229.6 Rationale

- The only function that returns a `cursor_t` is `ksGetCursor`. Its documentation is completely broken:
- `ksRewind` and `keyNextMeta` cannot be used on the same variable (`ks`).
- The documentation (not just for the above function, but all over the `ks*` functions) also contains lots of warnings (use only cursor from same keyset, may be invalid, may become invalid, etc.) that make it seem like there is something special about these cursors, when in fact they are simply indices. There is no information (at least that I could find), how cursors can be modified. Most of the time it even seems like modifying cursors would be a bad idea.

229.7 Implications

229.8 Related Decisions

229.9 Notes

Chapter 230

Types of <tt>KeySet</tt>s

230.1 Problem

A `KeySet` can be used in different ways. Among these are at least:

- As originally intended, a set of configuration data
- Metadata of a `Key`
- General associative array data structure

Only `KeySet`s with one kind of keys makes sense. These types have slightly different requirements, and therefore need to be treated differently. Currently, keys of different types can be arbitrary intermixed, leading to many error variants and error code that is time-consuming (full iteration is needed).
The `KeySet` for metadata (returned by `keyMeta`) also has the special requirement, that it must be restricted to metadata keys, even when it is empty.

230.2 Constraints

- API must be minimal
- API must be safe to use for each of the different types
- `KeySet` returned by `keyMeta` must always be limited to `meta:/` keys

230.3 Assumptions

- The types are not different enough to warrant the overhead of entirely different APIs and data structures.
- Introducing a type safe API with different C-types for each type of `KeySet`, is not worth the necessary API duplication.

230.4 Considered Alternatives

230.4.1 No restrictions

Leave `KeySet` entirely unrestricted, so it can be used for each use case.
Sometimes there can be odd edge cases. For example, what should a storage plugin do, if it receives a `KeySet` with both configuration data and metadata.

230.4.2 Restriction based on first <tt>Key</tt>

To effectively create different kinds of `KeySet`s, we restrict which `Key`s can be inserted into a `KeySet` based on the first `Key` in a `KeySet`.

Any Key except cascading keys can be inserted into an empty KeySet. Only Keys matching the type of a KeySet can be inserted into a non-empty KeySet. The type of a KeySet is determined by the first Key, i.e., ksAtCursor(ks, 0). Transitively, this means the type of a KeySet is fixed when the first Key is inserted and when a KeySet is cleared, it becomes "type-less" again.
The different types of KeySet will be:

1. Metadata: Can only contain Keys with namespace KEY_NS_META.
2. General data: Can only contain Keys with namespace KEY_NS_MISC.
3. Config data: Can only contain a mix of Keys with namespace KEY_NS_SPEC, KEY_NS_PROC, KEY_NS_DIR, KEY_NS_USER, KEY_NS_SYSTEM and KEY_NS_DEFAULT.

Importantly, Keys with namespace KEY_NS_CASCADING can never be inserted into a KeySet. This is a restriction that simplifies the logic for cascading lookups. Instead, the KEY_NS_MISC namespace should be used.
Cascading lookups always try the namespaces allowed in the type of KeySet in order, but never try KEY_NS_SPEC. This means cascading lookups work for all types of KeySet the way a user would expect.
The special restriction for the KeySet returned by keyMeta can be solved by adding a new flag to struct _KeySet:

```
bool forceMeta : 1;
```

230.4.3 Different structs and APIs

Extract the implementation of KeySet into an internal API (and possibly data structure). Expose a different type and API for each of the use cases.
This creates unnecessary overhead and at least some code duplication, even if it is minimized by extracting the implementation.

230.4.4 KeySets also have Namespaces

Add a ksSetNamespace API which allows to change the namespace of a KeySet. This requires an alias KS_NS_CONFIG to the namespaces KEY_NS_SPEC, KEY_NS_PROC, KEY_NS_DIR, KEY_NS_USER, KEY_NS_SYSTEM and KEY_NS_DEFAULT.
Only keys of the namespace set to ksSetNamespace can be added. ksSetNamespace fails if called after keys were added.
While the term "namespace" (NS) is used here, it may be better to use "type" to avoid confusion.
This alternative conflicts with the constraint of minimal API.

230.5 Decision

230.6 Rationale

230.7 Implications

230.8 Related Decisions

- Namespace for miscellaneous data

230.9 Notes

Chapter 231

Simplify API

231.1 Problem

According to symbols.map the public core API has 124 symbols[1], which is arguable too much for a key-value API which has as highest goal simplicity.
In particular following areas have many functions and are not simple:

- binary keys
- memory functions
- comparison functions
- error/warnings

The urgency of this decision is that API can be easily introduced later but we cannot get rid of it after 1.0.

231.2 Constraints

- To not disturb main features as described in use cases.

231.3 Assumptions

- Binary data is not a core feature, if needed the plugin system can also work without (properly tagged) binary data in key sets.

231.4 Considered Alternatives

- flags for binary data

231.5 Decision

The exact API changes are not listed here, because it would just a long list that could just as easily be found in the Git history.
However, the API changes follow these rules:

- *Remove* all functions related to key metadata, except those listed below
- *Remove* all functions related to keyset cursors, as well as `ksHead` and `ksTail`
- *Remove*/_Change_ all functions related to (binary) key values as described in (Binary)
- *Remove* all functions that use a user-provided buffer to return keyname/value/etc.

- *Change*/_Add_ `keyMeta`/`keySetMeta` to directly read/write the metadata KeySet of a Key
- *Rename* `keyGet*Size` to `key*Size`
- *Add* `ksRemove` function to remove a Key at a specific index
- *Remove from public API* all the helper functions that use `elektraMalloc` (e.g. `elektraFormat`), as well as all the helper functions that only enhance standard APIs with additional error checks (e.g. `elektra↩StrCmp`)
- *Remove from public API* `ksCut`, `ksDeepDup`, `ksCopyInternal` and other functions that should never have been public
- *Remove* all functions that can be replaced by others (e.g. `ksPop`)
- *Rename* all functions to start with `elektra` as described in Elektra Prefix, and use `KeySet` instead of `ks` (also applies to names above)

231.6 Rationale

231.7 Implications

231.8 Related Decisions

- Binary
- Elektra Prefix

231.9 Notes

[1] the 124 symbols are (as found by @kodebach):

- 6 for the KDB stuff
- 6 for the plugin system
- 48 for Key
- 31 for KeySet
- 15 other helper functions
- The other 18 symbols are the public constants for the error API.

Chapter 232

Elixir Bindings

232.1 Problem

When creating Elixir bindings one needs a way to use some kind of foreign function interface. In Erlang (which is the basis for Elixir) such a foreign function interface exists in the form of NIFs (native implemented functions). The question is how this NIF interface should be implemented.

232.2 Constraints

1. Elektra's C API is given.
2. The implementation should be completed in a given time frame of around 20 to 40 person-hours.

232.3 Assumptions

1. The compatibility of the Elixir bindings is not very important.

232.4 Solutions

232.4.1 By hand

Implement the NIFs by hand. That is, every function of the Elektra C API which one wishes to expose to the Elixir binding has to be explicitly implemented as a NIF. The big disadvantage is, of course, that this process is error-prone. Furthermore, any change in the C API has to be manually carried over to the NIF.

232.4.2 Use Nifty code generation tool

Use the existing code generation tool `Nifty`.
Reasons for not choosing this approach are:

- Installation process required several patches of Nifty.
- Resource management is not that supplied by the Erlang NIF library but a custom one.

232.4.3 Write own code generation tool

Write a custom generation tool.
In principle one should be able to take a C header file and convert this to a NIF in an automatic fashion.
Solving this problem in general clearly would be a daunting task. However, when restricting oneself to the use case for creating an interface for, say, Elektra's KDB this might actually have been feasible.
The main difference to Nifty would be that one could adopt the memory management tools provided by the Erlang NIF library directly. Furthermore, it would be possible to write an improved interface for the types, e.g. binary and string.
The main reson for not choosing this approach were time constraints.

232.5 Decision

The chosen approach was to use "Alternative A", i.e., to implement the NIFs by hand.

232.6 Rationale

The main reason for this choice were time constraints.
Since I have assumed that the compatiblity of the Elixir bindings is not very important, it is acceptable if future changes break these bindings.

232.7 Implications

- Every incompatible change in the C API needs manual changes in the Elixir binding.

232.8 Related Decisions

- GitHub PR: Elixir Bindings

232.9 Notes

- If there were a guarantee that the C API does not change, then the chosen approach would be unproblematic.

Chapter 233

Change Tracking

233.1 Problem

Currently, a range of Elektra plugins are implementing some sort of change tracking for configuration data. This includes, but is not limited to, the internalnotification and dbus plugins. In the near future, Elektra shall also be extended with session recording.
KDB itself also has some rudimentary change tracking (via the `keyNeedsSync` flag) to determine whether `kdb↩ Set` needs to actually do something.
These competing change tracking implementations create multiple problems:

1. pointless waste of resources, as data is duplicated in each plugin,
2. multiplication of code, generating a maintenance burden.
3. various subtle differences in change tracking behavior, e.g., `kdbSet` might write a config file but notification is not sent out
4. the current approach to change tracking in plugins is fragile, which is outlined in a separate decision about valid kdbGet/kdbSet s

For `KeySet` we need to track which of the keys:

- have been removed
- have been added

For `Key` we need to track:

- original value of the key
- size of the original value (for binary keys)
- metadata, which is a combination of the tracking of keysets and keys
- tracking should only be done on the following namespaces:
  - `system:/`
  - `user:/`
  - `dir:/`
  - `meta:/`
  - `spec:/`

233.2 Constraints

Change tracking must:

- be transparent for applications using the public Elektra API

- be transparent to users changing configuration data
- if overhead is not negligible: only do tracking as required, i.e., a plugin specifically requests it
- have negligible overhead if disabled
- not duplicate data for each plugin that wants change tracking
- work with all allowed sequences of `kdbGet` and `kdbSet` as per this decision

We only want to track changes that are done by the user, not changes that are done by plugins. I.e. the scope of change tracking is on what happens *outside* of `kdbGet` and `kdbSet`.
The library `libelektra-core` must be kept minimal.

233.3 Assumptions

- It is possible to do change tracking with reasonable memory and computation overhead
- It is possible to design a single change tracking API that is useful for all existing and future plugins
- False positives are okay
  - this may happen when some keys have been changed by the user, but have subsequentially been "unchanged" by a transformation plugin Scenario: plugin that converts `false<->0` and `true<->1`
    * `system:/background` is stored with value `false`
    * user gets key `system:/background` with value `0` (after conversion by plugin)
    * user changes it to `false`
    * changetracking detects that value has been changed, because `false` != `0`
    * plugin changes value from `false` to `0`
    * consumers of the changetracking API will now get a false positive if they query whether `system:/background` has been changed
    * We assume consumers of the changetracking API have access to both the previous and the new value of a changed key. Therefore, they could detect these false positive cases, if they need to.
- False negatives (missed changes) are not okay

233.4 Solutions - Storage

233.4.1 Utilize already existing <tt>backendData->keys</tt>

We already store which keys have been returned by `kdbGet` for each backend within KDB. Currently, those are shallow copies. To be useful for changetracking, we need to perform deep copies instead. As keys and keysets are already utilizing copy-on-write, this would not add too much memory overhead.
A problem with this approach is that the internally stored keys are recreated as new instances every time `kdbGet` is called.

233.4.2 Combine with internal cache

We already decided that we want to have an internal deep-duped keyset of all the keys we returned. See internal cache decision.
The difference to `backendData->keys` is that this cache is not recreated each time `kdbGet` is called.

233.4.3 Have a seperate storage for changetracking

Have a global keyset with deep-duped keys that is purely used for changetracking and nothing else.

233.4.4 Store changes as meta keys

When something changes for the first time, store the original value as a metakey to every key. Not yet clear how we handle changes to metadata then. Also not really possible for keysets.

233.4.5 Implement changetracking as a mechanismn on the <tt>Key</tt> and <tt>KeySet</tt> datastructures

The idea here is that we extend the `KeySet` and `Key` structs with additional fields.
The tracking itself would be done within the `ks*` and `key*` methods, after checking if it is enabled. It would also transparently work for metadata, as metadata itself is implemented as a keyset with keys.
Downsides of this approach:

- It adds functionality to `libelektra-core` which may violate the constraint above. It may, however, be debatable whatever 'minimal' means in this context.
- Adding fields to the structs causes a slight memory overhead, even with tracking turned off. While negligible for `KeySet` due to the low amount of keysets in typical applications, it may be noticable for `Key`. On a 64-bit system we'd add 16 bytes to it. 8 bytes for the pointer to the original value, 8 bytes for the size of the original value. To put this in perspective, the current size of the `Key` struct is 64 bytes, so we'd add 25% overhead to an empty key. However, this percentage will be much lower in a real-world application, as the usefulness of an empty key is very low.
- Another downside here is that it is not so easy to determine what the "original" value is. Some part of `libelektra-kdb` would need to mark the keys as original (after transformations etc.)

233.5 Solutions - Implementation

233.5.1 Implement directly within <tt>libelektra-kdb</tt>

Implement all the logic for changetracking directly within `libelektra-kdb`.

233.5.2 Implement as a seperate plugin

Implement changetracking as a hooks plugin that will be called within `kdbGet` and `kdbSet` accordingly.
The following hooks will be needed:

- `tracking/get`: will be called at the end of `kdbGet`, directly before the result is returned.
- `tracking/set`: will be called at the beginning of `kdbSet`.
- `tracking/changeset`: compute the changeset for the requested parent key and return it.

As every hook plugin can define its own contract, in theory all storage forms mentioned in the previous chapter should be possible to implement. We could just point the plugin to `backendsData->keys` or the internal cache if we go down that route.

233.6 Solutions - Query

233.6.1 Provide an API within <tt>libelektra-kdb</tt>

The API should be usable by plugins and by applications utilizing Elektra. It does not matter whether the changetracking is implemented as part of `libelektra-kdb` or as a separate plugin. The API may look something like this:

```
bool elektraChangeTrackingIsEnabled (KDB * kdb);
ChangeTrackingContext * elektraChangeTrackingGetContext (KDB * kdb, Key * parentKey);
KeySet * elektraChangeTrackingGetAddedKeys (ChangeTrackingContext * context);
KeySet * elektraChangeTrackingGetRemovedKeys (ChangeTrackingContext * context);
KeySet * elektraChangeTrackingGetModifiedKeys (ChangeTrackingContext * context); // Returns old keys
       (pre-modification)
bool elektraChangeTrackingValueChanged (ChangeTrackingContext * context, Key * key);
bool elektraChangeTrackingMetaChanged (ChangeTrackingContext * context, Key * key);
KeySet * elektraChangeTrackingGetAddedMetaKeys (ChangeTrackingContext * context, Key * key);
KeySet * elektraChangeTrackingGetRemovedMetaKeys (ChangeTrackingContext * context, Key * key);
KeySet * elektraChangeTrackingGetModifiedMetaKeys (ChangeTrackingContext * context, Key * key); // Returns
       old meta keys (pre-modification)
```

233.6.2 Provide query methods as part of a separate plugin

This solution only makes sense if changetrackig is implemented as part of a seperate plugin. It will be a bit challenging to use for applications, as it would require that applications have access to the plugin contracts.
The changetracking plugin needs to export at least functions for the following things in its contract:

- Get added keys
- Get removed keys
- Get modified keys
- Get added meta keys for a key
- Get removed meta keys for a key
- Get modified meta keys for a key

233.7 Decision

Plugin and application developers can declare that changetracking is required via a contract. Store deep-duped copy-on-write returned keys in a separate keyset, which we might also use as internal cache. The whole change-tracking logic lives within `libelektra-kdb`. We provide an API for developers with `libelektra-kdb`.

233.8 Rationale

This decision meets all the constraints.
We are not altering the behavior of already existing functions, so everything is transparent for applications using the public Elektra API. As changetracking is an opt-in feature, it will create zero runtime overhead and negligible memory overhead in usecases where it is not needed. There is only a single implementation and storage, so no duplication for each plugin that wants change tracking. As we are storing our own tracking data, all sequences of `kdbGet` and `kdbSet` should work.

233.9 Implications

- Documentation for changetracking APIs needs to be written
- `struct _KDB` needs to be extended to contain the keyset for changetracking
- Replace all custom changetracking code within Elektra with this unified implementation. This should also include the `keyNeedsSync` flag. It also includes switching all notification plugins to use the new API.

233.10 Related Decisions

- Valid kdbGet/kdbSet sequences from `#4574`.
- Copy On Write

233.11 Notes

- Issue `#4514` uncovered a problem in the current change tracking approach
- Issue `#4520` already explored some of the considered alternatives

Chapter 234

Capabilities

234.1 Problem

Only plugins like `dump` and `quickdump` are able to represent any KeySet (as they are designed to do so). Limitations of other storage plugins are e.g., that not every structure of configuration is allowed.
Some of these limitations were documented `infos/status`, others were not.

234.2 Constraints

- tooling should be able to query which limitations exist

234.3 Assumptions

234.4 Considered Alternatives

- Implementing plugins that work around the limitations (e.g. escape the characters or rewrite directory values) is too complex and lead to new problems (e.g. escaping of the rewritten values and interactions of plugins, e.g. renaming and notification).
- Capabilities API was also found too complex, as application developers usually do not exactly know the requirements of their underlying format, especially if some parts of the configuration is extensible or derived from user-input.

234.5 Decision

Add `infos/features/storage` to document limitations of storage plugins. Ideally, storage plugins should throw an error in `kdbSet` for unrepresentable KeySets.
Elektra cannot guarantee that any configuration file format can be mounted anywhere. Developers, maintainers and administrators are responsible for what they mount. They need to test the setup.

234.6 Rationale

234.7 Implications

234.8 Related Decisions

- Base Name

234.9 Notes

See also #3504:

Chapter 235

Remove elektraMalloc() et al.

235.1 Problem

A problem with having `elektraMalloc` et al.[1] is, that:

- it makes it impossible to call libc functions that allocate data with malloc (e.g. strndup) or
- others that expect a pointer that can be passed to realloc.

You might even do that by accident and never notice the problem, until someone replaces elektraMalloc and then you could get some pretty bad memory bugs.

[1] Affected functions:

- `elektraMalloc`
- `elektraCalloc`
- `elektraFree`
- `elektraStrDup`
- `elektraStrLen`

235.2 Constraints

Proprietary apps are sometimes delivered together with a libc (either static or a startup script makes sure their own libs are found). If they would use Elektra, and the person wants the app to use the global Elektra it is a valid use case for the dedicated functions.

235.3 Assumptions

The only safe way to keep elektraMalloc et al. is to define that they will always call their libc counterpart (malloc et al.) and their purpose is simply to add assertions. Then we can also make them private, because you can just call free to free any pointer returned by Elektra.

235.4 Considered Alternatives

- Completely replace `elektraMalloc` / `elektraCalloc` with simple calls to malloc
- Fix all places where non-Elektra functions get used

235.5 Decision

- keep current state with the custom functions (allocators)
- make it optional for plugin developers to use language specific allocators or our custom allocators
- remove everything except `elektraFree` from public API
- remove all functions that don't actually involve any memory allocations (e.g. `elektraStrLen`, `elektraStrCmp`)
- add `elektraStrNDup` and other `stdlib` equivalents that do memory allocation.

235.6 Rationale

While the current state might cause some problems with compiler optimizations and discoverability, those issues probably are only very minor in practice. Removing all functions poses a considerable amount of work, which would have to be undone in case we need those functions one day anyway. The current downsides don't justify such a big procedure.

235.7 Implications

235.8 Related Decisions

235.9 Notes

@kodebach wrote: Ideally, I'd remove the functions, but it seems unlikely this will be accepted.

Chapter 236

Elektra Prefix

236.1 Problem

Some names, such as `kdbOpen`, `keyNew` are so generic that other libraries might also use them. Furthermore, it is inconsistent that some functions have prefixes (e.g. `elektraKsFilter`) and others don't have (e.g. `ksNew`).

236.2 Constraints

- In C such libraries, containing the same external identifier, cannot be used together.
- The `C99 standard, section 5.2.4.1` gives following limit: 31 significant initial characters in an external identifier (each universal character name specifying a short identifier of 0000FFFF or less is considered 6 characters, each universal character name specifying a short identifier of 00010000 or more is considered 10 characters, and each extended source character is considered the same number of characters as the corresponding universal character name, if any)

236.3 Assumptions

236.4 Considered Alternatives

- leave it as is
- use macros to have short names for actually longer external identifiers
- use `e_` or `el_` as shorter prefix

236.5 Decision

- Ensure all functions start with `elektra`.
- Ensure all macros and constants start with `ELEKTRA_`.

236.6 Rationale

- This makes it clear which functions come from Elektra.
- Unifies all function names within Elektra (including core).
- Avoids collisions with other libs.

236.7 Implications

- Changes in basically every application and tool, but this is automated with a refactoring tool @kodebach writes.
- To be 100% C99 compatible we cannot introduce identifiers which the same 31 character prefix (e.g. `ELEKTRA_WARNING_VALIDATION_SYNTACTIC_LEXER` and `ELEKTRA_WARNING_↩ VALIDATION_SYNTACTIC_PARSER` would be a problem). Otherwise, it is not guaranteed that the code can be compiled with every C99 compiler.

236.8 Related Decisions

236.9 Notes

Chapter 237

Header File Structure

237.1 Problem

kdb.h contains the public API for both libelektra-core and libelektra-kdb. It is confusing and makes it hard to see what library a function actually belongs to.
The big problem is kdbprivate.h. It has two main problems:

1. It contains stuff from many different libraries. I found at least libelektra-core, libelektra-kdb and libelektra-highlevel.
2. It contains things that are different levels of "private".

237.2 Constraints

- Each library should have at least one separate header file:

  Without this constraint, we would constantly include things that aren't needed. This is inconvenient as it clutters the auto-suggestions of IDEs. It also slows down the compile process, since the compiler has to parse everything that was included.

- Non-public API and public API should be kept in separate files
- Some non-public APIs need to be accessible for testing, such headers should not be packaged/installed

237.3 Assumptions

- There are different categories of "private":

  1. Some parts are truly private, i.e., shouldn't be used outside the library that defines them. These things are only there because a library was split into multiple `.c` files. This includes e.g., `opmphmNew` or `struct _BackendData`.

     Symbols belonging to this category should not appear at all in the `symbols.map` file. Headers declaring such symbols must never be installed.
  2. Other things are truly private, but must be tested. This includes e.g., most `backends*` functions or the `elektraKeyName*` functions.

     Symbols belonging to this category should not appear in at all in the `symbols.map` file. Headers declaring such symbols must never be installed.
  3. Some things are not part of the public API and will never be part of the public API. This includes the `struct _Key` and `struct _KeySet`, but `elektraMalloc` and the high-level functions needed for codegen. These things will never be public API, because we don't want to make all the guarantees associated with that. Nonetheless, they cannot truly be private, because functions in other libraries need access. Each of these functions/structs/symbols has a specific "target audience" that needs access.

     Symbols belonging to this category should not appear in a public section of the `symbols.map` file. Headers declaring such symbols generally should not be installed.

4. Finally, there things that aren't part of the public API, but may be in future. This includes e.g., `ksFind↩Hierarchy` or `elektraReadArrayNumber`. These functions could be public, but for various reasons are not. Maybe they are not well-tested, or maybe we just don't want to commit to the function yet.

   Symbols belonging to this category should not appear in a public section of the `symbols.map` file. Headers declaring such symbols can be installed, if symbols are in a kind of experimental pre-release cycle. If there is no concrete plan to make symbols public, the headers should, however, not be installed.

237.4 Considered Alternatives

237.5 Decision

The decision is split into 4 subsections "Libraries", "Plugins", "Tools" and "Tests", because the different parts of Elektra have different requirements.

237.5.1 Libraries

A library can be monolithic or modularized. Monolithic libraries should be small and bigger libraries with large APIs should be modularized.
A monolithic library `foo` may have these headers (covering categories 3 & 4 from above):

- `src/include/elektra/foo.h`: Contains the full public API of `libelektra-foo`. Will be installed as `<include-root>/elektra/foo.h`.
- `src/include/internal/foo.h`: Contains the internal API of `libelektra-foo`. Will not be installed.

A modularized library `foo` may have these headers (covering categories 3 & 4 from above):

- `src/include/elektra/foo.h`: Declares the public API of `libelektra-foo`, by including `#include <elektra/foo/*.h>`. Will be installed as `<include-root>/elektra/foo/public.h`.

  Such a header may only contain `#include <elektra/foo/*.h>` lines.
- `src/include/elektra/foo/*.h`: Additional public API header of `libelektra-foo`. Will be installed as `<include-root>/elektra/foo/*.h`.

  Any one `installed.h` of these installed headers must be included from `src/include/elektra/foo.↩h` via a line `#include <elektra/foo/installed.h>`.
- `src/include/internal/foo.h`: Declares the internal API of `libelektra-foo`, by including `#include <internal/foo/*.h>`. Will not be installed.

  Such a header may only contain `#include <internal/foo/*.h>` lines.
- `src/include/internal/foo/*.h`: Additional internal header of `libelektra-foo`. Will not be installed.

  Any one `installed.h` of these installed headers must be included from `src/include/internal/foo.↩h` via a line `#include <internal/foo/installed.h>`.

Additionally, all libraries may also have private headers:

- `src/lib/foo/**/*.h`: Additional headers may be present. These headers may only be used by other files within `src/lib/foo`, according to the rules in Including Headers.

237.5.2 Plugins

Plugins do not declare their API via header files. Their headers are never installed and can be named any way the developer wants.
Importantly, however, headers belonging to plugins must not be used outside the plugin.

237.5.3 Tools

Tools do not have a C API, so their headers are also never installed. Consequently, there are no naming rules for header files belonging to tools.
Just like with plugins, the headers belonging to tools must not be used elsewhere.

237.5.4 Tests

For category 2 from above (private but needs to be tested), one of the following should be done:

1. Declare functions as `static` in a `.c` file and `#include ""` this file from the test.
2. Add a private non-installed header (e.g., `src/lib/foo/testapi.h`) that declares the API that needs testing, `#include ""` that and compile the test sources together with the `.o` files from the library (static linking).

If a symbol is needed in only one file and for tests, option 1 should be preferred. For symbols that are used in multiple files, a header needs to exist anyway. In any case, these unit tests should not be installed and should statically link the library into the test executable. This way we don't pollute our `symbols.map` files and keep the number of exported symbols down.

237.6 Rationale

- This structure makes the `#includes` simple and works nicely with our directory structure.
- Having a uniform naming convention simplifies things, both for developers writing Elektra and those using Elektra.
- Requiring libraries must either be fully modularized (main headers are only `#includes`) or completely monolithic, avoids the situation where a library has some parts in `foo.h` and other parts in extra headers. This is bad, because it encourages users to simply include `foo.h`, which defeats the point of modularized headers.

237.7 Implications

- The location of a header file within the source tree determines what API it contains:
  - Public: Will be installed, must be stable, can be used anywhere
  - Internal: Will not be installed, doesn't have to be stable, can be used anywhere, but public headers
  - Private: Will not be installed, doesn't have to be stable, can only be used in the same library
- Some libraries are currently neither fully modularized nor fully monolithic. The headers for these libraries must be restructured.
- There is no room for experimental APIs, i.e., headers that should be installed, but are not (yet) stable. Currently, we don't plan to have any such APIs in the 1.0 release. The plan is that anything that is not internal/private in 1.0 is stable. If the need arises to introduce experimental APIs, a new decision must be reached about this.

237.8 Related Decisions

- Including Headers
- Library Directory Structure

237.9 Notes

- The Linux kernel also has many header files with many cross-dependencies. Recently people started to look into the effect of this "dependency hell" on compile-time (see LKML).

Chapter 238

Including Headers

238.1 Problem

In C you can include a header file with
`#include "header.h"`
or
`#include <header.h>`
In the C standard, section 6.10.2, paragraphs 2 to 4 state:

> 2. A preprocessing directive of the form
> `#include <h-char-sequence> new-line`
>
> searches a sequence of implementation-defined places for a header identified uniquely by the specified sequence between the < and > delimiters, and causes the replacement of that directive by the entire contents of the header. How the places are specified or the header identified is implementation-defined.
>
> 1. A preprocessing directive of the form
>
> `#include "q-char-sequence" new-line`
>
> causes the replacement of that directive by the entire contents of the source file identified by the specified sequence between the " delimiters. The named source file is searched for in an implementation-defined manner. If this search is not supported, or if the search fails, the directive is reprocessed as if it read
> `#include <h-char-sequence> new-line`
>
> with the identical contained sequence (including > characters, if any) from the original directive.
>
> 1. A preprocessing directive of the form
>
> `#include pp-tokens new-line`
>
> (that does not match one of the two previous forms) is permitted. The preprocessing tokens after include in the directive are processed just as in normal text. (Each identifier currently defined as a macro name is replaced by its replacement list of preprocessing tokens.) The directive resulting after all replacements shall match one of the two previous forms. The method by which a sequence of preprocessing tokens between a < and a > preprocessing token pair or a pair of " characters is combined into a single header name preprocessing token is implementation-defined.

238.1.1 Definitions:

- h-char: any member of the source character set except the new-line character and >
- q-char: any member of the source character set except the new-line character and "

In short, the whole search process of the `#include` mechanism is implementation-defined. However, because `#include` works (mostly) like a literal copy-paste, deciding between "" and <> within a public header file can affect user code.

238.2 Assumptions

- Modern Compilers work like GCC:
  - `#include "[header]"` treats `[header]` as file path relative to the current file (as defined by the standard, if such a file doesn't exist there is a fallback to the <> behavior)
  - `#include <[header]>` treats `[header]` as file path relative to one of the pre-defined include-paths
- We can write a tool that enforces all rules in this decision, in such a way that new contributors can learn the rules as they go.

238.3 Considered Alternatives

- Let developers decide:

  This would just create inconsistent code.
- Put all headers into one directory and use the same relative layout as will be installed. Then use `""`:

  This makes for an inconvenient development experience. It is also pretty hard to achieve for plugins and private headers would clutter the include-directory.
- Only use `#include <>` and add everything to include path.

  This makes the build setup more complicated. It also makes it much easier to accidentally include a non-installed header in an installed one.

238.4 Decision

The rules for including headers are:

- To include a private non-installed header (i.e., a file that is only available in the source repo) use:
  ```
  #include "./header.h"
  // or
  #include "./subdir/header.h"
  ```
  It should not be necessary, to include a private non-installed header from another directory. This ensures that libraries only use their own private APIs. Going up one (or more) directories, but staying in the same library would be okay, but is considered an indication of bad code structure. A library can always be restructured to avoid needing a `#include "../"`.
- Internal non-installed headers (i.e., a file that is not installed, but contains APIs exposed to other libraries within Elektra) can be included with
  ```
  #include <internal/somelib.h>
  // or
  #include <internal/somelib/header.h>
  ```
  Since these internal headers are not installed, they must only be included from other headers that are not installed or from code files.
- Installed headers are included with their full path as if they are installed already:
  ```
  #include <elektra/somelib.h>
  // or
  #include <elektra/somelib/header.h>
  ```
- System headers or headers for external libraries are included with:
  ```
  #include <stdlib.h>
  #include <dbus/dbus.h>
  // etc.
  ```

Only `build/include`, which is a copy of `src/include` after CMake processing, is added to the include path. The entire `build/include/elektra` directory is installed as-is, with the exact same directory structure and without any further processing.
This is enough for `#include <>`s to work.
But we will create an additional script that enforces that `internal/*` headers are not included from headers in `src/include/elektra`. We will also enforce that the path in a `#include ""` always starts with a `./` and does not contain any `/../`.

These include-rules do not apply to tests. Tests can include anything (including `.c` files) from anywhere within the code base to allow testing private APIs.

238.5 Rationale

The decision highlights the difference between installed and non-installed headers. The main driving factor for using `""` at all was that including a non-installed private header with <> would be unexpected, since non-installed headers shouldn't be in the (standard) include-path. This distinction also makes it easier to ensure that non-installed headers are not accidentally included in installed ones.
See also considered alternatives.

238.6 Implications

- All installed headers (and only those) must be put into `src/include/elektra`.
- To enforce the restrictions on the path in an `#include ""`, we will use a simple `grep`-based script that runs as a test case and as an early part of the CI (like e.g., the formatting check).
- There must not be any `#include <internal/...>`s anywhere within `src/include/elektra`. This will be enforced by the same `grep`-based script as the paths in `#include ""`.

238.7 Related Decisions

- Header File Structure
- Library Directory Structure

238.8 Notes

Chapter 239

Internal KeySet Cache

239.1 Problem

`kdbGet` might return more or fewer keys than requested. This was found confusing several times.

239.1.1 More Keys

Even if calling `kdbGet` with a parent key below a mountpoint, `kdbGet` will nevertheless return all keys of the mountpoint. Pseudo code example, assuming there is a mountpoint at `/mountpoint` and a key `/mountpoint/other`:

```
kdbGet (kdb, ks, keyNew("/mountpoint/below"));
assert (ksLookup (ks, "/mountpoint/other") == NULL);
```

It was found unexpected that this assert will fail.

In a similar fashion, calling `kdbSet` without the seemingly superfluous keys causes Elektra to unintentionally delete them from disk.

```
kdbGet (kdb, ks, keyNew("/mountpoint/below"));
KeySet * below = ksCut (ks, keyNew("/mountpoint/below"));
ksSet (kdb, below, keyNew("/mountpoint/below"));
// suddenly /mountpoint/other has been removed from the configuration file on disk, even if the user
       explicitly stated to only change stuff in /mountpoint/below
```

239.1.2 Fewer Keys

When doing a second `kdbGet` with a new keyset no keys will be returned when no backends report changed data, because kdb internally thinks the data is already up-to-date:

```
static void test_doubleGet (void)
{
        printf("running %s\n", __func__);
        // Setup
        Key * parentKey = keyNew("/somewhere", KS_END);
        KeySet * ks = ksNew(0, KS_END);
        KDB * kdb = kdbOpen (ksNew(0, KS_END), parentKey);
        kdbGet (kdb, ks, parentKey);
        ksAppendKey (ks, keyNew ("user:/somewhere", KEY_VALUE, "abc", KEY_END));
        ksAppendKey (ks, keyNew ("user:/somewhere/key", KEY_VALUE, "xyz", KEY_END));
        kdbSet (kdb, ks, parentKey);
        kdbClose (kdb, parentKey);
        // Scenario
        kdb = kdbOpen (ksNew(0, KS_END), parentKey);
        KeySet * ks1 = ksNew (0, KS_END);
        KeySet * ks2 = ksNew (0, KS_END);
        kdbGet (kdb, ks1, keyNew("/somewhere", KEY_END));
        succeed_if (ksLookupByName (ks1, "/somewhere/key", 0) != NULL, "should find key (1)");
        kdbGet (kdb, ks2, keyNew("/somewhere", KEY_END));
        succeed_if (ksLookupByName (ks2, "/somewhere/key", 0) != NULL, "should find key (2)");
}
```

It was found unexpected that the second assert - should find key (2) - will fail.

239.2 Constraints

- memory consumption must be low for `kdbGet`, see 4. Goal: Performance, in particular, deep duplication is too expensive

- For non-optional parts of Elektra, also non-POSIX systems must be supported

239.3 Assumptions

239.4 Considered Alternatives

239.4.1 Keep Current Situation

Improve documentation to make people more aware of these two problems:

- add a tutorial about `kdbGet` semantics
- add full examples how to correctly work with `kdbGet`

239.4.2 Changing <tt>parentKey</tt> according to mountpoints

"More keys":
Upon returning from `kdbGet`/`kdbSet`, set the keyname of parentKey to the key that actually is parent of the data that was loaded. I.e. to the mountpoint of the backend that contains parentKey. `parentKey` is already an inout-type argument, since we use both it's value and metadata to return some information.
A few people found this behavior just as unexpected as the current problem. E.g. a sequence of `kdbGet` and `kdbSet` with the same parent key might lead to different outcomes depending on the mountpoints.
"Fewer keys":
This is a bug in the logic of the "nothing changed" optimization. A partial solution would be to copy the keys from `backendData->keys`, so they are actually there, and we don't just assume they are there. Still some extra steps are required to make it work in all cases, e.g.:

```
TEST_F (Simple, NothingToDo2)
{
        using namespace kdb;
        KDB kdb;
        KeySet ks;
        auto parentKey = "system:" + testRoot;
        EXPECT_EQ (kdb.get (ks, parentKey), 0) << "nothing to do in get";
        ks.append (Key (parentKey + "a", KEY_VALUE, "x", KEY_END));
        kdb.set (ks, parentKey);
        EXPECT_EQ (kdb.get (ks, parentKey), 0) << "nothing to do in get";
        EXPECT_TRUE (ks.lookup (parentKey + "a")) << "key a not found";
        EXPECT_EQ (ks.lookup (parentKey + "a").get<std::string> (), "x") << "key a with wrong value";
        // TODO: adding the line below breaks the test
        //       This is because the Key instance is shared between `ks` and the internal data of `kdb`.
        // ks.lookup(parentKey + "a").set("y");
        KeySet ks2;
        EXPECT_EQ (kdb.get (ks2, parentKey), 0) << "nothing to do in get";
        EXPECT_TRUE (ks2.lookup (parentKey + "a")) << "key a not found";
        EXPECT_EQ (ks2.lookup (parentKey + "a").get<std::string> (), "x") << "key a with wrong value";
}
```

A very simple way to make it work would be to make the keys returned by kdbGet read-only. If we do not do this, we need a deep-copy or a copy-on-write copy.

239.4.3 Cachefilter Plugin

Naively one would simply cache the whole keyset and use `ksBelow` to always get the keyset.
This idea was implemented and later on `discarded: a3d95f`
The main problems are:

- very high memory consumption (duplication of KeySets)
- problems specific to hooks, see `#1072`

239.4.4 MMAP Cache with parent key

We make the mmap cache non-optional so that we always have a keyset of configuration data internally. The cache will be used to do change tracking. From this keyset, we use `ksBelow` to return the correct keyset.
Disadvantage: mmap implementation for Windows would be needed.

The cache should be updated at the end of every kdbGet that was a cache miss. So during kdbSet (assuming there is no external modification, i.e. conflict) the on-disk data of the cache should always be up-to-date. The idea would be to just read the cached keyset from disk and diff against the current keyset in kdbSet.
This would mean enabling change tracking also enables the cache (or at least updating the cache, we don't have to use it in kdbGet). If that's not wanted or if the cache data cannot be used directly for some other reason, the same approach could still be used. We'd just have to write the keyset to disk during kdbGet and use it during kdbSet. Since disk space is far less precious than RAM, we could even create separate files for every parent key. If we do go down this route, kdbClose should clean up the files created by this KDB instance to avoid wasting disk space.

239.4.5 MMAP Cache without parent key

We make the mmap cache non-optional and only use a single cache, caching everything. We remove the parent key of `kdbGet` and `kdbSet` and always return the keyset of the whole KDB.

- Disadvantage: mmap implementation for Windows would be needed
- Then also symlinks (fallback, override etc.) and constraints for keys outside of the parentKey would work. It would make the `-a` option of `kdb get` unnecessary.

It is still unclear whether this would actually be a good default behavior. Normally it is expected for mountpoints to be isolated If symlinks work like this, the isolation is partially broken. One could argue that the problem right now is that such "broken" symlinks are not prevented. The proposed solution doesn't completely fix the problem either
For example:

- `system:/foo` is a mountpoint, there are no other mountpoints
- `spec:/foo/bar` refers to `system:/abc` via `meta:/override`

Now in a plugin that is part of the mountpoint `system:/foo` doing a `ksLookupByName (ks, "/foo/bar", 0)` would NOT use `system:/abc`, because that key is never part of the keyset passed to this plugin.
However, in an application that used `system:/foo` as the parent, the `meta:/override` would work with your proposal. This seems like very confusing behavior, because both the application and the plugin seemingly use the same parent key.

239.4.6 Data restrictions

Make all the keys returned by kdbGet completely read-only. To change the data it is required to append an entirely new key to replace the existing one. Then we just need to keep a shallow copy internally.

239.4.7 API restrictions

Change the API and remove KeySet from kdbGet and kdbSet also option 4 in the operation sequences decision. If the keyset is owned by the KDB handle, it should not be such a big surprise, if there is extra data in there.
This only fixes the "Fewer Keys" issue.

239.4.8 Copy On Write

We keep a duplicated keyset in-memory and tag the keys as copy-on-write (COW). From this keyset, we use `ksBelow` to return the correct keyset. If the user tries to change the value or metadata of these keys, the data gets duplicated. I.e. the original keyset is not changed. The name is not relevant. It is always read-only, because the key is in at least one keyset (the internal one).
Possible copy-on-write implementations are described in another decision.

239.5 Decision

Implement the copy-on-write approach.
We keep a copy of all keys returned by the backends in memory. We use `ksBelow` to only return a copy-on-write copy of keys the user requested on `kdbGet`.

If the user tries to change the value or metadata of these keys, the data gets duplicated (copy-on-write). I.e. the data of the original keys is not changed. The name is not relevant. It is always read-only, because the key is in at least one keyset (the internal one). Possible copy-on-write implementations are described in another decision.
In `kdbSet` we use the user-provided `KeySet` for all backends strictly below `parentKey` as before. For the backend that contains `parentKey`, we start with the internally cached data. We then remove everything that is at or below `parentKey` (via `ksCut`) and replace it with the data from the user-provided `KeySet`. Keys not at or below `parentKey` therefore remain untouched.

239.6 Rationale

Semantics can be provided without additional code or overhead in the core. As we need copy-on-write for efficient change tracking anyway, it makes sense to also use this approach for the internal cache. The copy-on-write solution also does not require any changes or restrictions to the current API.

239.7 Implications

- Before we can implement this decision, we need to implement the copy-on-write decision.
- `kdbGet` will only return copy-on-write copies of keys below `parentKey` from the internal cache.
- `kdbSet` will use the keys within the internal cache to supplement all the keys above `parentKey` so that backends can write the correct data.

239.8 Related Decisions

- Global Validation
- Copy On Write

239.9 Notes

Problem "Fewer Keys" was found to be a `"horrible problem"`
Issues where the problem described here was found confusing:

- `#760`
- `#1363`

@mpranj wrote about the performance for `MMAP Cache without parent key`:

> in my benchmarks the pointer correction was never a bottleneck.

Chapter 240

Iterating Keyname Parts

240.1 Problem

While iterating over the parts of a keyname is easy for an experienced developer (just jump from `\0` to `\0` until you hit the name size), there is no good API that newcomers can use.

240.2 Constraints

No changes to the underlying APIs, while maintaining reasonable performance.

240.3 Assumptions

240.4 Considered Alternatives

- Put the new function into `libelektra-core`.

240.5 Decision

Add this new function to a separate library (name TBD):

- –

  Returns

  – the "next" (see above) key name part after `currentPart`. ∗/ const char ∗ keyGetNextPart (Key ∗ key, const char ∗ currentPart);

240.6 Rationale

The function is not required for a minimal API, but it is useful for people who don't know everything about the internals of Elektra.

240.7 Implications

240.8 Related Decisions

240.9 Notes

Chapter 241

<tt>keyIsBelow</tt>

241.1 Problem

There are multiple very similar functions to check the "is below" relation between to keys:

- Replacement `keyIsBelow`
- Replacement `keyIsBelowOrSame`
- Replacement `keyIsDirectlyBelow`

241.2 Constraints

Keep (at least) the same functionality

241.3 Assumptions

241.4 Considered Alternatives

241.5 Decision

Merge `keyIsBelow` with `keyIsBelowOrSame` and `keyIsDirectlyBelow` to create this new API:

```
int keyIsBelow (const Key * base, const Key * other);
```

The replacements for the old functions are:

- old keyIsBelow: `keyIsBelow > 0`
- keyIsBelowOrSame: `keyIsBelow >= 0`
- keyIsDirectlyBelow: `keyIsBelow == 1`

241.6 Rationale

241.7 Implications

241.8 Related Decisions

241.9 Notes

Chapter 242

Namespace and Name of Keys

242.1 Problem

A `Key` in Elektra is identified by its name, which consists of a namespace and a number of name parts. There are two common representations for the name: escaped and unescaped.
The unescaped form is essentially a single namespace byte plus and arbitrary-length sequence of arbitrary bytes, in which a `\0` byte separates parts. The escaped name is a `\0`-terminated string that maps 1:1 onto unescaped names. More details can be found in the relevant docs.
There is a conflict between these two forms in terms of API convenience and efficient execution. Generally, the escaped form is more convenient to use, since it is entirely human-readable and just a "normal" string. Implementing many tasks (like order comparisons) is, however, much simpler when using the unescaped name. Additionally, using the unescaped name often results in more performant code as well.
A particularly common example that highlights the difficulties in handling escaped names is splitting the name into parts. In the escaped name, this task requires correct handling of escape sequences, whereas in the unescaped name parts are always delimited by a `\0` byte.
Before this decision, we stored both versions in every `Key`. However, this resulted in too much memory use, so we need to find another solution.
The question now is, which representations should be used by `libelektra-core` and how.

242.2 Constraints

- Because `KeySet` is ordered by name and stores `Key`, the order comparison between the name of two `Key`s must be "fast enough". (see assumption below)
- We need a single pointer to a single buffer that contains the entire name of a `Key`. While there are other options, some of which could even save memory (e.g., split into parts and deduplicate), much of the `KeySet` internals rely on the fact that the name is a single buffer. Changing this would require major redesigns.

242.3 Assumptions

- In most cases the escaped name is used for convenience and not because of actual requirements.
- The most common case for using the escaped name is UI: reading names from or displaying them in a user interface (e.g., `kdb` CLI)
- In the constraint about order comparisons above, we assume that "fast enough" means "comparable to a single `memcmp`". Profiling for previous implementations, not based on a single `memcmp` of unescaped names, showed the comparison as a bottleneck, while the current single-`memcmp` implementation does not show the bottleneck. That said, it may be possible to find a solution slower than the current one that is still fast enough to avoid the previous bottleneck.

242.4 Considered Alternatives

242.4.1 Only escaped name

Because the escaped name is a simple `\0`-terminated string, it can be represented as a single `char *`.
Storing the name as a single `char *` would be the most space efficient. But resizing would require counting the length every time. Therefore, for storage the better solution may be a `char *` and a `size_t`.
However, in the API the name could always be a single `char *`, making for a very easy to use API.
The biggest problem with this approach is that comparing two escaped names is not trivial. The comparison needs to account for namespaces, parts and escaping. Previous benchmarks showed that it is very hard or even impossible to make the comparison of escaped names fast enough for our use cases.
Similarly, iterating over the individual parts of a name (and/or manipulating them) is non-trivial, because it requires logic to handle escape sequences.

242.4.2 Only unescaped name

The unescaped name contains `\0` bytes. It therefore must be represented as a pointer and a size.
This can make for less convenient API, but there are mitigation strategies using additional types. Using unescaped names in code can be inconvenient, especially regarding the namespace. Without a namespace a name could be written as e.g., `"foo\0bar"`. But with a namespace it would be something like `"\1\0\foo\0bar"` and developers would need to remember what namespace `\1` is. Using the `KEY_NS_*` constants like this is not easily possible.
Both order and hierarchy comparisons are very simple in this case and can be implemented with a single `memcmp` and a tiny amount of extra logic (e.g., to handle cascading names). Iterating over the individual parts is also trivial, since all parts are separated by `\0` bytes.

242.4.3 Only unescaped name, with separate namespace

In the above solution, the entire unescaped name (including the namespace) would always be considered one unit. As such, there would only be a single pointer and a size in an API that needs a name. This can be inconvenient, because it makes using the `KEY_NS_*` constants more difficult.
This solution enhances the above, by considering the namespace a separate thing. Above the namespace is intrinsically part of the name. It is essentially just a restriction on the first part of the name and sometimes the namespace must be considered specially. In this solution, we consider the namespace a separate entity from the start. A key does not have a name, which starts with a namespace. Instead, a key has a namespace *and* a name.
This is mostly a theoretical distinction, but it makes it easier to argue in favor of APIs that use separate arguments for the namespace. It also makes it more obvious that sometimes the namespace on its own can have an influence on the behavior of a function.
In the API the name could now be given as separated into namespace and the rest of the name. Instead of taking a single pointer and size, which receive values like `"\1\0foo\0bar"` and `10`, the API would take a namespace, a pointer, and a size, with values like `KEY_NS_CASCADING`, `"foo\0bar"` and `8`.
Internally, we don't necessarily need to store this as separate fields. The namespace could be combined into one buffer with the rest of the name, and stored as a single pointer and size. However, depending on the API there can also be benefits to keeping the namespace as a separate field.
Even with a separate namespace field, most benefits of "Only unescaped name" are retained. The memory consumption is near minimal (alignment padding can cause a difference). Comparisons are exactly the same, just with an additional namespace byte comparison beforehand.

242.4.4 Both escaped and unescaped name

The previous approach used both to combine the advantages of escaped and unescaped name.
The API could largely rely on the escaped name, while e.g., comparisons can use the unescaped name.
The issue with this approach is the insane memory consumption. Keynames can already be quite long and `Key` is at the base of Elektra. Storing every name twice in only slightly different forms essentially doubles the memory consumption.

242.4.5 Both escaped and unescaped name, but only unescaped stored

Instead of storing both escaped and unescaped name, only the unescaped name could be stored.

APIs that use the escaped name would do conversion on the fly.
This approach has several downsides. First, while the conversion may be optimized, it will never be free in terms of runtime. But more importantly, if an escaped name should be returned by an API, it must be stored somewhere. This means extra allocations and crucially somebody needs to do the cleanup. In other words, it complicates the API.

242.4.6 Escaped and unescaped name in single buffer

Another variant of the above. The escaped and unescaped name are stored in a single buffer. This avoids extra allocations and extra pointers and sizes in structs.
The escaped name could also be stored lazily only when needed. This would solve the cleanup problem.
While this may seem like the ideal solution, there are still some downsides. The biggest problem is the API design. If the API uses escaped names a lot (because it is more convenient), then this essentially degrades into the "↩ Both escaped and unescaped name" solution. Even if APIs exist for both escaped and unescaped names, the convenience benefit, will lead to more use of escaped names. This means the escaped name will be stored for many keys and therefore the benefit of the lazy allocation is negated.
Without the lazy allocation benefit, the only difference to "Both escaped and unescaped name" is that we have fewer pointers and sizes in structs. This saves some amount of memory and allocations, but makes internal code more difficult to write and understand.

242.5 Decision

Go with "Only unescaped name, with separate namespace" from above:

- Store only unescaped name with size inside `struct _Key`
- API of `libelektra-core` will use unescaped name exclusively
- Convenience functions using escaped names, will be provided via other libraries
- Where appropriate the API will take the namespace as a separate argument to allow using `KEY_NS_*` constants.
- Whether namespace is stored separately in `struct _Key` will be decided at a later point, when the scope of all API changes and changes to `struct _Key` is clear.

242.6 Rationale

- Largest memory savings among the proposed options
- Option to use separate namespace argument leads to more convenient API (`KEY_NS_*` constants).
- Simple internal code
- Escaped name requirements can easily be solved by an additional library (e.g., `libelektra-ease`, `libelektra-extra` or new standalone library for names), because not every caller will need those functions.
- Full API and internal struct layout aren't designed yet, so deciding how to store namespace is difficult.

242.7 Implications

- `keyNew` needs to change
- `keyName` returns unescaped name
- functions for escaped name move out of core

242.8 Related Decisions

242.9 Notes

242.9.1 Printing unescaped name in GDB

In GDB (and probably others) the unescaped name of a `Key * key` can be printed with (assuming the name is in `key->ukey` and its size in `key->keyUSize`):

```
p *key->ukey@key->keyUSize
```

This prints `key->ukey` as a fixed-length string of length `key->keyUSize`, e.g., for `user:/abc` it prints:

```
$1 = "\006\000abc"
```

Chapter 243

Library Directory Structure

243.1 Problem

Currently `src/libs/elektra` contains the source of several libraries. Each of the libraries should have its own folder. Furthermore, we need to allow rust source to exist in parallel.

243.2 Constraints

- It must be possible to set up include-paths via CMake so that `#include <elektra/foo/header.h>` works the same in the build directory and when installed.

243.3 Assumptions

243.4 Considered Alternatives

- One folder per language, then split by code type (`src/<lang>/plugins`, `src/<lang>/libs`, etc.):
  - Pro: Makes the per language setup easier, since everything is contained in a single directory and that directory is for a single language, just as if it was any other standalone project in that language.
  - Con: When navigating the source-code you need to know what language e.g. a plugin is written in to find the correct folder.
- First split by code type, then one folder per language (`src/plugins/<lang>`, `src/libs/<lang>`, etc.)
  - Similar to the variant above, but has more downsides. "single folder per language" no longer applies and you still need to know the language to find the folder.

243.5 Decision

We use this directory tree (YAML syntax for highlighting):

```
doc:
benchmarks:
examples:
scripts:
src:
  include:
   # headers that don't belong to a specific library
  libs:
    core-c: C implementation of libelektra-core.so
    core-rust: Rust implementation of libelektra-core.so
    opts: libelektra-opts.so has only a single implementation, so no suffix
  # plugins and tools use the same suffix idea as above
  plugins:
  tools:
  bindings:
    rust: Rust bindings
    cpp: C++ bindings
```

```
    java: Java bindings
  tests:
    shell: shell script tests
    cframework: framework for C tests
    gtest-framework: framework for C++ tests
    abi: ABI compatibility tests
  # build tool configuration files to create a single root project for every language
  # this is improves IDE support
  CMakeLists.txt
  build.gradle
  Cargo.toml
```

Unit tests (*) (e.g. the old `tests/ctest`, `tests/kdb`) should be bundled with the code they are testing: e.g. in Rust test code is in the same file, in C/C++ it should be in the same folder and in Java a separate source set in the same module is used. This makes it easier to find the tests for a library, and also creates a uniform structure (in Rust or Java it would be harder to put tests into the current structure). System tests that test multiple components live in the `src/tests` folder.

> (*) The term "unit test" is used very loosely here. Most tests for e.g. the `kdb` tool will not, strictly speaking, be unit tests, since they don't replace dependencies with mocks/fakes and instead test the whole `kdb` tool.

For now benchmarks will remain in the top-level `benchmarks` directory, unless there is a language specific reason to have them next to the code they benchmark (like unit-tests). In future this may change to facilitate automated performance-regression tests.

243.6 Rationale

This setup, allows to find a plugin/library/etc. just by knowing its code type (library/plugin/etc.) and its name. At the same time, using language suffixes allows having multiple implementations in different languages. Finally, whenever possible there should be project config file in the top-level `src` directory for every language. This will improve IDE support and make it possible to treat all code from one language as a single project. However, it is important that modularity is not lost in the process. It must still be possible to depend on individual libraries in CMake, without having to build everything else that is written in the same language.
Therefore, the top-level project should only exist, if the language allows projects that only include sub-modules. Otherwise, fully separate projects for every plugin/library/etc. should be used to preserve modularity.

243.7 Implications

243.8 Related Decisions

- Header File Structure

243.9 Notes

Chapter 244

Library Split

244.1 Problem

Only `libelektra-core` is supposed to access private data, but this contradicts the goal to keep the library minimal. `kdbprivate.h` was too generic, it contained many other parts next to the struct definitions of Key/KeySet.

244.2 Constraints

- The `C99 standard, section 5.2.4.1` gives following limit: 4095 external identifiers in one translation unit
- Some parts of Elektra, like `mmapstorage` need access to private data structure.
- Elektra does not support several different struct definitions of Key/KeySet. Alternative implementations that want to coexist (e.g. `mmapstorage` should still work) must use the same struct definitions of Key/KeySet.

244.3 Assumptions

244.4 Considered Alternatives

- keep current situation

244.5 Decision

Also allow `libelektra-extra` (and maybe other explicitly documented libraries) to access private Key/KeySet.
Put struct definitions of Key/KeySet in a separate header file, which gets included by parts that need it.
All currently planned libraries and their respective API prefixes are listed in the `Notes` below.

244.6 Rationale

- allows various users (plugins, applications) to link to (more or less) exactly what they need
- allows symbol versioning on different levels (for different evolving libraries)
- allows alternative implementation of parts of Elektra, e.g. a libcore written in Rust
- facilitates code reuse between plugins

244.7 Implications

- we need to clearly communicate which plugins must be exactly in the version of the `libelektra-core`
- all libraries will share a versioning scheme and are only supported if used in the same version

244.8 Related Decisions

244.9 Notes

- `libelektra-core`: The core minimal API of Elektra. Defines what `ElektraKey` and `ElektraKeyset` are, and contains the minimal API for manipulating them.

  Prefixes: `elektraKey*` and `elektraKeyset*`

- `libelektra-kdb`: The main API for interfacing with the KDB.

  Prefix: `elektraKdb*`

- `libelektra-lowlevel-c`: Additional C APIs that are useful when working with `ElektraKey` and `ElektraKeyset`, but not are not minimal API. Specifically targets C and not intended for use via bindings.

  Prefixes: `elektraCKey*`, `elektraCKeyset*`

- `libelektra-highlevel-c`: The C high-level API for reading/writing configuration with Elektra. Intended for use by applications. Partially intended for use with code-generation. Specifically targets C and not intended for use via bindings.

  Prefix: `elektraHlc*` (Note: Even though there isn't any other `highlevel` implementation, `c` suffix to show this lib is only for C)

- `libelektra-opts`: The API for parsing command-line arguments according to Elektra's spec.

  Prefix: `elektraOpts*`

- `libelektra-notification`: The API for setting up callbacks and automatically updated variables linked to changes in the KDB.

  Prefix: `elektraNotification*`

- `libelektra-io`: Elektra's API for asynchronous operations.

  Prefix: `elektraIo*`

- `libelektra-merge`: The API for merging and/or detecting conflicts between two keysets.

  Prefix: `elektraMerge*`

- `libelektra-plugin`: The plugin API. It forms the base for all plugins and contains the functions required to implement a plugin. It also contains the API for interacting with (other) plugins. This includes loading plugins and calling exported functions.

  Note: This library comes with two separate headers. One for implementing a plugin and another for loading/calling plugins.

  Prefix: `elektraPlugin*` (Note: previously `libelektra-plugin` merged with `libelektra-invoke`)

- `libelektra-pluginload`: Internal static library, linked into `libelektra-plugin` (re-exported) and `libelektra-kdb` (not exported). Contains the code for loading plugins.

  Prefix:: `elektraPlugin*` (shared with `libelektra-plugin` because exported there)

- `libelektra-mount`: The API for manipulating mountpoints.

  Prefix: `elektraMount*`

- `libelektra-type`: The API that defines Elektra's type system.

  Names: `elektra<TYPE>ToString` and `elektraKeyTo<TYPE>` (Note: extracted from `libelektra-ease`)

- `libelektra-extra`: Contains extra APIs for `ElektraKey` and `ElektraKeyset` beyond the minimal API of `libelektra-core`. These APIs are things that could be considered part of the classes for `ElektraKey` and `ElektraKeyset`, but which we do not consider minimal. The APIs should not specifically target C (see `libelektra-lowlevel-c` for that) and should be usable via bindings (if appropriate for the other language).

  Prefix: `elektraExtra*` (Note: includes `elektraExtraKeysetCut`, etc.; may also include stuff from old `libelektra-ease` or `libelektra-meta`)

- `libelektra-ease`: A collection of various other APIs that help when interacting with `ElektraKey` and `ElektraKeyset`. These APIs go beyond what could be considered part of the "classes" for `ElektraKey` and `ElektraKeyset` (e.g., SHA256 hashes). The APIs should not specifically target C (see `libelektra-lowlevel-c` for that) and should be usable via bindings (if appropriate for the other language).

  Prefix: `elektraEase*` (Note: rest of old `libelektra-ease`, merged with `libelektra-meta`; cleanup needed)

- `libelektra-utility`: Standalone helper functions that don't depend on `libelektra-core`

  Prefix: `elektraUtil*`

- `libelektra-base`: Internal static library, linked into `libelektra-core` (not exported) and `libelektra-utility` (partially re-exported). Contains helper functions for e.g., memory allocations, string formatting, etc.

  Prefix: `elektra*` (for internal), `elektraUtil*` (for exported)

Chapter 245

Namespace for miscellaneous data

245.1 Problem

A `KeySet` is powerful data structure, which could be use for data other than configuration data. Especially with the hashmap implementation, using a `KeySet` for generic data can be useful. For this generic data we do not care about the namespace-related features.
However, because a `Key` cannot exist without a namespace, we need to choose a namespace.

245.2 Constraints

- The outcome must be compatible with the Types of `KeySet`sTypes of `KeySet`s" decision".

245.3 Assumptions

- The Types of `KeySet`sTypes of `KeySet`s" decision" leads to separate types of `KeySet`s that are restricted w.r.t. the namespaces they may contain.

245.4 Considered Alternatives

245.4.1 Use cascading <tt>Key</tt>s

We could use the existing `KEY_NS_CASCADING` for one of the types of `KeySet`s.
This may be confusing to users of the APIs. The purpose of `KEY_NS_CASCADING` becomes muddied. It is no longer just for lookup, but would now have a secondary purpose.

245.4.2 Introduce separate <tt>KEY_NS_MISC</tt> namespace

We introduce a new `KEY_NS_MISC`, which is used exclusively for generic and miscellaneous data. It implies a separate type of `KeySet`.
This leaves `KEY_NS_CASCADING` for its original purpose, while still solving the issue with generic data.

245.5 Decision

Introduce separate `KEY_NS_MISC` namespace.
In the unescaped form this will be `\x09` (byte with decimal value `9`). In the escaped form it will be `misc:/`.
`KEY_NS_MISC` has a separate type of `KeySet`. Such a `KeySet` can only contain `KEY_NS_MISC` keys. It may not be used for metadata or with `kdbGet`/`kdbSet`.
A cascading lookup is still allowed for such `KeySet`s. However, the `KEY_NS_CASCADING` will simply be replaced with `KEY_NS_MISC` and then an exact name lookup is performed, as if the `KEY_NS_MISC` namespace was given directly.

245.6 Rationale

Introducing a new namespace is much cleaner than reusing one of the existing ones. It fits better with the separate types of `KeySet`s.

245.7 Implications

- Some code already uses cascading keys for general data. It must be updated to the new `KEY_NS_MISC`.
- The cascading logic for `ksLookup` needs to adapted for `KEY_NS_MISC`.

245.8 Related Decisions

- Types of `KeySets`Types of `KeySet`s"".

245.9 Notes

Chapter 246

Private API

246.1 Problem

Only `libelektra-core` is supposed to access private data, but this contradicts the goal to keep the library minimal. `kdbprivate.h` was too generic, it contained many other parts next to the struct definitions of Key/KeySet. Theoretically everything in `kdbprivate.h` is supposed to be private, but lots of code still uses it when it shouldn't. `kdb.h` is also the only header that is definitely public. All other headers are anybody's guess.

246.2 Constraints

- The `C99 standard, section 5.2.4.1` gives following limit: 4095 external identifiers in one translation unit

246.3 Assumptions

246.4 Considered Alternatives

- keep current situation, as described above

246.5 Decision

- Also allow other libraries (e.g. a new `libelektra-operations`) to access non-public API. Such libraries need to have a good reason (e.g. performance, impossible otherwise, etc.) why they access non-public API and they need to be kept up-to-date. If a library cannot provide a stable API on top of the unstable non-public API, it clearly needs to state which APIs are not stable.
- Put struct definitions of Key/KeySet in a separate header file, which gets included by parts that need it (see also Header File Structure).

246.6 Rationale

- allows various users (plugins, applications) to link to (more or less) exactly what they need
- allows alternative implementation of parts of Elektra, e.g. a libcore written in Rust
- facilitates code reuse between plugins
- The `extern const` version constant(s) can be used for compatibility checks. In most cases, compatibility should be ensured via package management systems or manually by the user. Sometimes it may be possible to write code (e.g. in a third-party library) such that it is compatible with multiple versions of Elektra. In those cases, we need to know what version of Elektra is installed, so that the correct code can be executed. For example, if a new version accepts an argument to `foo()` that previously wasn't allowed. An external library may be able to use this when available and use other code as a fallback in older versions.

246.7 Implications

- Any library that uses non-public API, but itself exposes a public API, must either
  - hide breaking changes of the underlying non-public API
  - increment it's major version, if an underlying non-public API causes an incompatible change
- For some libraries (e.g. `libelektra-operations`) it might be easiest to require a specific version of `libelektra-core`. This cannot easily be enforced, so it must be well documented.
- Plugins can safely be used as before, but loading may fail if there is a version mismatch (only important for third-party plugins).

246.8 Related Decisions

- Header File Structure

246.9 Notes

Chapter 247

Arrays

247.1 Problem

Currently it is inefficient to detect the length of an array and it is impossible to know if a key (without subkeys) should be an array or not.

247.2 Constraints

- None

247.3 Assumptions

247.4 Considered Alternatives

- Metadata arrays simply work by convention as they are not serialized in special ways nor they get validated.
- `###empty_array` as in `yajl`, problem: does not allow efficient access of first element
- store length (and not last element), problem: needs prepending of `#_...`
- store element after last element (C++-Style), would not fit nicely with key-value
- use value and not the metadata `array`, problem: is ambiguous
- use metadata on all children

247.5 Decision

Store length in metadata `array` of key, or keep metadata `array` empty if empty array. Only children that have `#` syntax are allowed in a valid array. The index start with `#0`. Both `keyAddName("#12")` or `keyAddBaseName("#_12")` is allowed to add the 13th index.
For example (`ni syntax`, sections used for metadata):

```
myarray/#0 = value0
myarray/#1 = value1
myarray/#2 = value2
myarray/#3 = value3
myarray/#4 = value4
myarray/#5 = value5
[myarray]
  array = #5
```

It is not allowed to have anything else than `#5` in the metadata `array`, e.g. even `#4` would be a malformed array.
With the metadata `array` in `spec:/` the `spec` plugin will add the `array` marker with the correct length. This needs to be happening early, so that plugins can rely on having correct arrays.
For example:

```
spec:/myarray      # <- has array marker
user:/myarray      # <- either has correct array marker or no array marker
user:/myarray/#0
```

```
user:/myarray/#1
```

Here, the `spec` plugin would add `array=#1` to `user:/myarray` if it was not there before.
To look up an array, first do `ksLookupByName (ks, "/myarray", 0)` on the parent. With the last index you get from its metadata `array`, iterate over the children. A cascading lookup on every individual child is also needed to make sure that overrides on individual elements are respected.
For example:

```
spec:/myarray    # <- contains the specification for the array
spec:/myarray/#  # <- contains the specification for the array elements
dir:/myarray/#0  # <- not an array, just an override for user:/myarray/#
user:/myarray    # <- with metadata array=#0, this would be the array we get
user:/myarray/#0
system:/myarray  # <- not found in cascading lookup, as user:/myarray exists
```

The `spec` plugin should check if it is a valid array, i.e.:

- that the parent key always contains the metadata `array`,
- that the correct length is in `array`,
- that the array only contains `#` children, and
- that the children are numbered from `#0` to `#n`, without holes.

247.6 Rationale

- Is very similar to `binary` metadata but without its problems.
- The key alone suffices to know if it is an array.
- One can distinguish an array with keys that are called by chance e.g. `#0`.

247.7 Implications

- The plugin yajl needs to be fixed
- metadata library needs to be adapted
- spec plugin needs to be fixed, a lot of work needed there. Most amount of work is to detect misstructured nested arrays (`#` intermixed with non `#` keys) which is a possibility also in all the alternatives of this decision.
- A `user:/` or `dir:/` key can change the semantics of a `system:/` array, if not avoided by `spec`.
- user-facing documentation should contain a note like: "Mixing array and non-array keys in arrays is not supported. In many cases the trivial solution is to move the array part into a separate child-key."

247.8 Related Decisions

- Hooks
- Global Validation
- Base Names
- Metadata in Spec Namespace
- Spec Expressiveness
- Binary.

247.9 Notes

https://github.com/ElektraInitiative/libelektra/issues/182

Chapter 248

Definition of Bool

248.1 Problem

Inconsistent use of booleans in various parts of Elektra.

248.2 Constraints

- needs to be string

248.3 Assumptions

248.4 Considered Alternatives

- only check presence or absence (no cascading override of already present key possible)
- use booleans as in CMake, which allows on/off, true/false, ... (would need convenience across the code)
- do not accept a specification with `type = boolean` without a default

248.5 Decision

Only the strings `0` and `1` are allowed in the `KeySet` for `type = boolean`, for both values and defaults. Everything else should lead to errors in checkers (in `kdbSet`).
A spec with `type = boolean` without a specified default should be interpreted as `default = 0`.
Example for an implementation in C in an application:

```
if (k != NULL && strcmp(keyString(k), "1") == 0) {/*true*/} else {/*false*/}
```

Storage plugins are allowed any representation as suitable, e.g., a JSON plugin might render `1` as `true`.
The type checker plugin should allow

- non-presence
- the string "0"
- the string "1"

248.6 Rationale

- most easy to implement
- allows non-presence to be false
- plugins allow us to convert to any other behavior

248.7 Implications

- Storage plugins are only allowed to emit `0` or `1` as key values
- Applications either get `0` or `1`, or (without a key) can safely assume that false is meant

248.8 Related Decisions

248.9 Notes

Chapter 249

Decision Process

249.1 Problem

Simply discussing in an issue and then implementing a solution is okay for non-substantial changes. Substantial decisions, however, must be made in a transparent and participative way.

- Discussing fundamental problems in forum-like threads showed to be repetitive and ineffective.
- Decisions by supervisors are undemocratic.
- Decisions in meetings are nontransparent for the outside world.

249.1.1 Terminology

- `Decision`: A text file which contains the content as explained here.
- `Decision PR`: A pull request that contains substantial changes for a decision.
- `Decision author`: Is the person who creates the decision PR.
- `Decision reviewers`: Are the people who review the decision PR.

249.1.2 Main Purpose

The main purpose of decisions is

- to have clear descriptions of technical problems and solutions.
- to get a common understanding of the problems and the impacts of possible solutions.

249.2 Constraints

- All relevant information about decisions must be within Elektra's repository.
- All decisions must go through at least two review rounds, with a merge in between.
- At least two people need to approve the decision in each round.
- Documentation guidelines apply to decisions.
- During the decision process, the PRs constantly get updated:
  - Make changes as new commits to the pull request.
  - Questions in the PRs are answered by:
    1. Update the PR and incorporate the review.
    2. Reply on GitHub:

        * Give a short summary of what you did in a single comment, and reply to individual questions if necessary.
        * Link to commit SHA-IDs for details.
      3. Mark all GitHub threads as resolved, if you incorporated the feedback as suggested. Committing a suggestion directly on GitHub does this automatically.
    - As generally recommended in Elektra, do not squash commits after they are visible in a pull request.
    - Rebase only if the decision was already accepted and has a merge conflict.

- For decision reviewers:
    - Prefer to directly give suggestions how to change sentences.
    - General questions should be asked in the root of "Conversation" and not at vaguely related sentences in the review.

- Changes not changing the decision step or the direction of the decision are not decision PRs.

- The person merging the decision PR must be someone other than the person that created the decision. There is no claim that decisions contain everything that was said. In particular corrections of wrong decision text is, if at all, only visible via Git history. Rather it is important that decisions:
    - contain everything relevant, and
    - what is written is technically correct.

- Usually no decisions are needed for libraries, plugins, tools or bindings if:
    - they are similar to already existing modules (e.g. yet another checker plugin)
    - if they reimplement some existing module (e.g. reimplementation in other programming languages)

- Reviewers are only required to read the files in the PR. They may not read the discussions surrounding the decision. It is encouraged that at least one decision reviewers provides a review *without* participating in the discussion. This ensures that there aren't any unintentional shared assumptions between discussion participants.

- Reviews focus on the "Problem" section first, and only when that is agreed upon focus in the other parts.

- The decision author invites several decision reviewers. Ask for help in the decision PR if you do not know whom to invite.

249.3 Assumptions

- People want to be informed about or even participate in what Elektra looks like in the future.

- People actually use Elektra and propose only what they need for their use cases.

- Decision authors have some scientific background and want decisions based on science, and not only on opinions.

- If assumptions, including this ones written here, are broken, decisions will be redone.

- Decision authors and reviewers want Elektra to improve, so they also want to accept (acceptable) decisions. In general people want change if it brings Elektra towards its goals.

- All decision reviewers want to help the decision authors to write a good decision.

- Decision authors are the main force behind a decision and possibly also of specific solutions. Nevertheless they don't want to avoid other solutions, are open to arguments/facts/etc. and incorporate all input of decision PR fairly.

- For important decisions, we will able to reach an consensus even if it requires that the core or plugins get multiple implementations. We accept indefinite postponement of lesser important decisions, where there is strong disagreement about the problem or the best solution. Thus we don't need a vote (besides the approved review) or a benevolent dictatorship.

- Different to initiatives like Rust, most contributors in Elektra are not experts in configuration management or programming languages. So we do not expect that a clear problem or solution is in the decision writer's mind beforehand. Instead the decision process is a supported learning process.
- People focus on getting the best solutions and not to wish for the impossible.
- People creating decision PRs have strong motives to also finish it. They take extra effort on them to be clear about the problem and find the best solution.
- The decision process itself isn't a barrier for people to write their first decision.

249.4 Solutions

- Create a decision process tailored to Elektra
- Issues like https://issues.libelektra.org/4521 (often did not result to a PR that documented the outcome)
- GitHub discussions (-||-)
- Votings (low participation and people often didn't inform themselves what the choices are)
- maintainer decides (was the situation before)
- PEPs: https://peps.python.org (tailored for programming language specifications, not ideal for architectural decisions)
- Rust RFCs: https://www.ietf.org/standards/rfcs/ (-||-, see also https://forge.rust-lang.org/compiler/mcp.html)

249.5 Decision

We use a decision process tailored to Elektra, in which decisions need to:

- be implementable within the next major release
- be according to Elektra's goals
- first be decided upon using the decision process described here

We use the template TEMPLATE.md. Explanations of the template are in EXPLANATIONS.md.
Steps are described in STEPS.md:

- Decision authors are the main force to improve the text to get a decision forward.
- Decision authors and decision reviewers together decide about which step a decision currently has.
- Each mandatory step requires two reviews and the merging of the decision PR.
- In each step we directly update the decision text with the different opinions. Discussions should focus on the decision text so that the text evolves and improves.

249.6 Rationale

- The process is lightweight and simple.
- The template makes sure important points are not forgotten.
- Every decision is by design in its own file with its own Git history.
- PRs allow to better support the constraint that everything must be within Elektra's repository (also rejected decisions).

- "Decision", "Rationale" and "Implications" are filled out later to keep the discussion unbiased
- PRs allow to suggest changes and review individual sentences of the decision.
- As "Decided"/"Implemented" decisions need a special decision PR, we cannot merge a "Decided"/"↩ Implemented" decision PR by accident.
- Several "Related Decisions" are very important even if everyone agrees on one solution. They allow reviewers and future readers of the decision to understand which options were considered and why they were rejected.
- The decision process is focused around the decision text (and not forum-like discussions), so that:
  - The resulting text is understandable without reading any discussions.
  - There is a common understanding after only reading the decision text.
  - To avoid any gaps of reading discussions and the decision.

249.7 Implications

- Proposal issues are obsolete.
- The decision process creates at least:
  - two chances to comment decisions, and
  - two commits in the Git history.
- Decision PRs might be merged quite often until they reach the step "Implemented". They might even be merged several times within the same step.
- Decisions encourage to write documentation before actually writing code.
- The proposal encourages healthier discussions despite cognitive biases we all have, see in Notes below.

Partially implemented because of open tasks:

- rename Considered Alternatives to Solutions
- rename Steps

249.8 Related Decisions

This is the only non-technical decision, so no issues are related.

249.9 Notes

- The first idea often is not the best, don't fixate on it. Abraham Luchins called this the “Einstellung effect.” Thus we encourage to generate as many ideas as possible for any problem (interrupt effect).
- We have a tendency to add: Take courage to also propose solutions that (mostly) remove code. See Leidy Klotz et al., e.g. “People systematically overlook subtractive changes“.
- Failures are not a bad thing. It is a good thing to have "rejected" decisions for future references and to recheck if assumptions change.
- Decisions are about "Why?" and not about "Who?" or "When?". Such discussions should be in separate issues.
- Discussions in issues/discussions are not prohibited. They don't bring a decision forward, though. To not waste time, it is recommended to start with the decision process as described here asap.
- Make sure that moving between the stages is detected by git. E.g. rename in a commit that doesn't rewrite too much of the decision.

- See also change requests: https://en.wikipedia.org/wiki/Change_request

Written by Markus Raab.

- First discussion round starting 10.10.2022.
- Second discussion round starting 28.10.2022.
- Third discussion round starting 6.11.2022.

Chapter 250

Hooks in KDB

250.1 Problem

Some components of `kdbGet`/`kdbSet` should be optional. We use the plugin system for that. However, some of these cases cannot be tied to a mountpoint. This was the idea of global plugins, but that idea proved problematic. In the old global plugins implementation:

- Notification does not happen once after final commit, but for every plugin.
- Problems in spec plugin

These problems can be traced back to the placement of the plugins. We need to clean up and simplify the placement.

250.2 Constraints

- Plugin interface should be the same. Many plugins, where appropriate, e.g. dbus, should work as global plugins w/o any change in code (i.e. only changes in contract)
- Global plugins might depend on specific applications or specific mount points (it should be possible to enforce global plugins for specific applications).

250.3 Assumptions

- Only following plugins need hook functionality:
  - mmap
  - spec
  - gopts
  - receiving of notifications (internalnotification)
  - sending of notifications (dbus, ...)
  - recording of changes

250.4 Considered Alternatives

- generic placements like /prerollback /rollback /postrollback /getresolver /pregetcache /pregetstorage /getstorage /postgetstorage /postgetcache /setresolver /presetstorage /setstorage /precommit /commit /postcommit which can be executed at: /init /deinit /foreach proved to be too complicated and difficult to test.

250.5 Decision

Have hooks and APIs specific for plugins listed in assumptions. These hooks are not shared, so no `list` plugin is needed.
Installed plugins will be used.
We'll hard code the names of the plugins. For changing those plugins symlinks will have to be used.

250.6 Rationale

- allows adding more types of plugins later, also post-1.0
- much clearer semantics for each type, no need to compromise
- easier to implement

250.7 Implications

- remove `global-mount` command
- command for mmap/notification/... enable disable (like current `kdb cache` tool)
- remove `list` plugin
- remove plugins that stop working or disallow global positioning for them
- call `spec` as needed several times
- remove current global plugins mechanism
- rename `global` to `hook` in contract

250.8 Related Decisions

- Array
- Ensure

250.9 Notes

Chapter 251

Reference Counting

251.1 Problem

- locking is not reset when ref counting again gets 0 (adding to keyset and pop again) #2202
- C++ API for KeySet and Key has unexpected differences: also use ref counting for KeySets (also suggested in #1332)

251.2 Constraints

251.3 Assumptions

251.4 Considered Alternatives

- make ref counting thread safe (probably useful for JNI)
- start with 1 for reference counting and let keyDecRef do keyDel

251.5 Decision

- add second counter to Key
- One counter is for references, the other one is for locking the keyname. The keyname is thereby protected with a re-entrant lock.
- introduce reference counter for KeySets (for external keyset references, e.g. in bindings)
- limit number of references to `UINT16_MAX - 1` and use `UINT16_MAX` as an error value
- return error on reference count overflow
- no error on underflow (decrement when zero), instead stay at zero
- use fixed sized types (`uint16_t`) for reference counters
- increment/decrement references before/after passing instances to plugins

251.6 Rationale

- Adding a second reference counter to Key and reducing the size of both significantly (`size_t` to `uint16_t`) actually saves memory (32 vs 64bit on 64-bit machines) compared to the previous solution.
- The added complexity of maintaining two reference counters is worth the trade-off for the gained functionality.

251.7 Implications

251.8 Related Decisions

251.9 Notes

Not implemented yet:

- Update bindings to use KeySet reference counter (especially C++)
- Second counter for automatic keyname (un)locking

Chapter 252

Arrays

252.1 Problem

Currently multiple warnings are saved in an elektra non-conforming array notation which is limited to 100 entries. The notation of `#00` is against the design decision made.

252.2 Constraints

252.3 Assumptions

252.4 Considered Alternatives

252.5 Decision

The format should be aligned with the correct array notation, starting with `#0`. The maximum number of warnings will stay at 100 entries (`#0` - `#_99`).

252.6 Rationale

252.7 Implications

To keep the ordering we add an underscore _ once we go to beyond 10 warnings (`#_10`).

252.8 Related Decisions

- Array

252.9 Notes

Chapter 253

Backend Plugin

253.1 Problem

- Old backends (before Elektra 0.9.12) store plugins in arrays which have a fixed number of slots for each plugin role. The number of plugins which can be assigned is limited, making it easy to reach the limit if many plugins are in use.
- As structs, old backends are separate from the plugin interface and integrated into the core of Elektra. This makes it difficult to perform operations such as nesting plugins, or to develop other implementations for backends.

253.2 Constraints

- All existing plugins, except hooks, should continue working as before.

253.3 Assumptions

253.4 Considered Alternatives

- Multiple storage plugins within a single backend
- Improve backend to contain more plugin slots
- Making number of plugins per slot unlimited

253.5 Decision

- The current backend implementation was redeveloped into a backend plugin. The core of Elektra accesses backends through the standard plugin interface.
- The new backend plugin supports an unlimited number of plugins in any position where more than one plugin is sensible, e.g., unlimited plugins in `poststorage`, but only a single one in `storage` and `resolver`

253.6 Rationale

- Making backends also plugins detaches their implementation from the core of Elektra, making it possible to develop new kinds of backends without major changes to the core itself.
- As plugins, backends can contain further backends, making it possible to nest plugins and enabling new kinds of plugin combinations such as fallback storage options.

253.7 Implications

The structure of `system:/elektra/mountpoints` has to change to accommodate different kinds of backend plugins.

253.8 Related Decisions

253.9 Notes

https://issues.libelektra.org/2963

Chapter 254

Base Name

254.1 Problem

A key name is made out of a sequence of key part names, and can be constructed with `keyAddBaseName/keySetBaseName`. Both applications and configuration file formats might need arbitrary strings to be encoded within a key name part.
For example:

- an application uses names of internal components as sections within the configuration.
- a parser reads an empty string, to be encoded as base name.

254.2 Constraints

- `keySetBaseName (key, keyBaseName (key))` should be a NOP, which is needed for round-trips: If a storage plugin serializes what it gets with `keyBaseName`; `keySetBaseName` must lead to the same key.
- support any configuration file format (i.e., any valid file of some format can be transformed to a KeySet, e.g. TOML supports empty key part names).

254.3 Assumptions

254.4 Considered Alternatives

- restrict what `keyAddBaseName/keySetBaseName` can accept: has the downside that applications would suddenly fail when trying to set some key base names
- have additional `keySetBaseName*` functions that make strings safe to be accepted in `keyAddBaseName/keySetBaseName`: seems to be a too big change in the storage plugins

254.5 Decision

`keyAddBaseName/keySetBaseName` never fail with any argument, so any character sequence can be escaped except of NULL bytes. The argument goes unmodified to the unescaped key name.
For arrays there is no escaping needed because an array is only an array if the metadata `array` is appended to the direct parent key. See array.

254.6 Rationale

- hard to use it wrong API: having only the functions `keyAddBaseName/keySetBaseName`, without any size argument.

- applications and storage plugins can pass any C string to `keyAddBaseName/keySetBaseName` without any further consideration

254.7 Implications

254.8 Related Decisions

- Array

254.9 Notes

Chapter 255

Bootstrap

255.1 Problem

Currently, the default backend (default.ecf) will also be used for bootstrapping. There are two problems with this approach:

1. Thus the default backend first will be read with parentKey `system:/elektra` and later with parentKey `system:/`, it needs to store absolute paths and thus won't work with most of the plugins (except dump).
2. When `system:` is large without mount points, everything is reread twice during bootstrapping.

255.2 Constraints

- Bootstrap should be fast and not unnecessarily read large files

255.3 Assumptions

255.4 Considered Alternatives

- Implement a hack so that `system:/elektra` is actually read as `system:/`. (Will not solve problem 2.)
  - It's a hack.
  - Its confusing and does not play well with persistent data with relative key names.
- Split up without compatibility mode: would need to migrate all mount points by exporting (with old version!) and then importing (with new version!)
  - I consider this too error-prone, people might easily forget to export with the old version and then discard their mount points unintentional.

255.5 Decision

Split up the concepts of default (`default.ecf`) and bootstrap (`elektra.ecf`) backend. During bootstrap only `elektra.ecf` is read. The default backend reading `default.ecf` is only relevant as long as no root backend is mounted.
Algorithm:

1. get system:/elektra using the file elektra.ecf (KDB_DB_INIT)
2. mount `elektra.ecf` (the init backend) to system:/elektra

255.6 Rationale

255.7 Implications

255.8 Related Decisions

255.9 Notes

Added scripts/upgrade-bootstrap to migrate from previous setups to upgrade to new system, either:

- touch /etc/kdb/elektra.ecf (loses old mount points)
- or do kdb export system:/elektra/mountpoints, kdb rm -r system:/elektra/mountpoints, kdb import system↩ :/elektra/mountpoints

Chapter 256

CMake Plugins

256.1 Problem

- plugin names and plugin folders not always exactly match (`resolver_*`, `crypto_*`)
- plugin should be able to register new variants
- there should be only one place to define a new plugin
- multiple categories should be possible per plugin, defined in README.md
- on some OS some plugins won't work (simpleini)
- some unit tests depend on bindings

256.2 Constraints

- should work with supported CMake
- It should be easy to add trivial plugins (without dependency and variants)
- It should be possible to add complex plugins (with dependencies on plugins/unit tests and many variants)

256.3 Assumptions

256.4 Considered Alternatives

- have one more mapping that describes folder <-> plugins (bad coherence)
- directly use README.md to also describe CMake deps (too limited in expression)
- split up AddPlugin.cmake and CMakeLists.txt (does not work well with variants)
- simply adding all directories in src/plugins and decide within the `add_plugin` if we should drop the plugin (see below in Rationale)

Names for flag:

- `DEPENDENCIES` -> might be used for `add_plugin`
- `ASSEMBLE_DEPS`
- `FIND_DEPS` -> not only finding happens
- `PROCESS_DEPS`
- `HANDLE_DEPS`

- DEPS_MODE -> too generic, new terminology
- CHECK_DEPS -> not only checking happens
- DO_DEPS -> sounds funny
- FIND_PACKAGES -> not only packages are subject to this phase
- COLLECT -> needs NOT
- FIND_PHASE -> needs NOT
- DEPENDENCY -> ambiguous

256.5 Decision

Introduce a CMake process where all plugins are processed three times. Following CMake variables are used for the phases:

- COLLECTION_PHASE .. collect all add_plugins
- DEPENDENCY_PHASE .. resolve all dependencies, do add_plugins again
- ADDTESTING_PHASE .. (reserve for potential 3rd phase)

1. Collection phase (COLLECTION_PHASE is ON), add_plugin internally builds up:
   - ADDED_PLUGINS
   - REMOVED_PLUGINS
   - ADDED_DIRECTORIES
2. assemble dependency phase (DEPENDENCY_PHASE is ON, only considering ADDED_DIRECTORIES), with:
   - find_libraries, actually search for libraries on the system (only relevant libraries of plugins that are considered for inclusion)
   - add_plugin, with *actually adding* the plugins
3. assemble all unit tests (ADDTESTING_PHASE is ON), either
   - with ADD_TEST in add_plugin, or
   - with add_plugintest (for unittests that have dependencies to bindings)

256.6 Rationale

Solves all the issues without adding too much complexity for actually adding plugins.
Maintaining additional mappings is very time-consuming. Simply adding all plugins directories is problematic. It would:

- clutter the CMake output (especially in the case of errors)
- introduce more variables into the CMakeCache which are irrelevant for the user
- maybe even find libraries in wrong versions which are incompatible to what other plugins need

256.7 Implications

- need to adopt all CMakeLists.txt

256.8 Related Decisions

- Plugin Variants

256.9 Notes

256.10 Limitations

- `ADDED_DIRECTORIES` of variants will be kept.
- Typos in plugin names are currently not checked, strings that are not plugin names are simply ignored.

Chapter 257

Copy On Write

257.1 Problem

One of Elektra's core goals is low memory usage. Currently, there are many places within Elektra where keys and keysets are duplicated and copied around. Most of those copied keys are never modified, but are required to be detached from the lifetime of the original instance. We want to introduce an in-memory copy-on-write mechanism to lower our memory usage.
In the near future, Elektra will also gain facilities for change tracking and session recording, both of which will potentially again duplicate keys. There are also aspirations to create a new, simple internal cache that would also benefit from a copy-on-write mechanism.

257.2 Constraints

1. The lifetime of a `Key` and a `KeySet` must be unaffected by copy-on-write.

257.3 Assumptions

257.4 Considered Alternatives

257.4.1 <tt>mmapstorage</tt>-like copy-on-write implementation

There is already some kind of copy-on-write semantics within `libelektra-core` to support the `mmapstorage` plugin. We can build on this and add a more generic copy-on-write to it.

```
Key * key = keyNew ("dir:/something", KEY_VALUE, "my value", KEY_END);
keyCopy (key_dup, key, ELEKTRA_CP_COW);
assert (keyString(key) == keyString(key_dup));
keySetString (key_dup, "other value"); // COW done here
assert (keyString(key) != keyString(key_dup));
assert (keySetName (key_dup, "dir:/valid") == -1); // must fail, as we have a COW key
assert (keyName(key) == keyName(key_dup)); // stays always valid
```

This is already implemented for the MMAP cache, so the implementation should be straightforward: Do the same COW duplications as done for MMAP but with a different flag.
For the metadata, however, also COW KeySets might be needed (at least with the current API). Example:

```
keyCopy (cow, key, ELEKTRA_CP_COW);
KeySet * cowMeta = keyMeta (cow);
ksAppendKey (cowMeta, keyNew ("meta:/whatever", KEY_VALUE, "abc", KEY_END));
ksRemoveByName (cowMeta, "meta:/type");
```

Pros:

- Elektra doesn't require MMAP

Cons:

- Lifetime of a copied COW key MUST be less than the key it was copied from. We can not track how many keys point to the same data and name this way, so we can only free data and name if the key does not have the COW flag. If the original key gets deleted, using a COW key that points to the same data and name will lead to corrupt data. The same is true for updating values of the original key.

This is only problematic if we want to use COW for keys outside of KDB. If it is only for use within KDB, especially for usage as internal cache and in change tracking, we could always guarantee that the original keys are going to last as long as the KDB instance. However, we need to document for the users of Elektra that keys returned from kdbGet are only valid until kdbClose. If they want to continue using them afterwards, they'd have to deep copy them.

Triggering the delete problem:

```
Key * originalKey;
Key * copiedKey;
keyCopy (copiedKey, originalKey, ELEKTRA_CP_COW);
assert (keyString (copiedKey) == keyString (originalKey));
assert (keyName (copiedKey) == keyName (originalKey));
keyDel (originalKey);
keyString (copiedKey); // Error! Original value has been deleted. Pointer to data in copiedKey points to
       freed memory
keyName (copiedKey);   // Error! Original name has been deleted.
```

Triggering the update problem:

```
Key * originalKey;
Key * copiedKey;
keyCopy (copiedKey, originalKey, ELEKTRA_CP_COW);
assert (keyString(copiedKey) == keyString(originalKey));
keySetString (originalKey, "new value!");
keyString(copiedKey); // Error! Original value has been deleted. Pointer to data in copiedKey points to
       potentially freed memory
```

The same problems in principle exist for mmapstorage where kdbSet frees (munmap) the keyset. We can still let users access the flag ELEKTRA_CP_COW, we just need to clearly document what is forbidden. Maybe set the KEY_FLAG_RO_VALUE on the original key, so that the API itself detects the error. There is, however, no flag for keyDel that we could set.

257.4.1.1 Changes to <tt>libelektra-core</tt>

The struct _Key will be extended with two more pointers, if we want to eliminate the lifetime problem:

- struct _Key * origData: points to the key containing the referenced data
- struct _Key * origName: points to the key containing the referenced name

Those two pointers are needed for memory management. Each referenced key will also have its reference counter increased. This way, an original key can be keyDel() d without impacting the copied keys.
Three new key flags will be added:

- KEY_FLAG_COW_VALUE: the value points to a value of another key
- KEY_FLAG_COW_NAME: the name points to the name of another key
- KEY_FLAG_COW_META: metakeys are copy-on-write

A new copy flag will be added:

- KEY_CP_COW: instructs keyCopy to copy whatever it should copy as copy-on-write. This will NOT be part of KEY_CP_ALL. We don't want developers outside of Elektra to accidentally use this.

If keyCopy() is instructed to do a copy-on-write copy:

- dest->data.v and dest->data.c point to the exact same location as in the source. dest->data↩Size is set to the same value as source->dataSize. KEY_FLAG_COW_VALUE is set on dest->flags. KEY_FLAG_RO_VALUE is set on source->flags. dest->originalData is set to source. source->refs is incremented.
- dest->key points to source->key. dest->keySize is set to the same value as source->key↩Size. dest->ukey points to source->ukey. dest->keyUSize is set to the same value as source->keyUSize. KEY_FLAG_COW_NAME is set on dest->flags. KEY_FLAG_RO_NAME is set on source->flags. dest->originalName is set to source. source->refs is incremented.
- dest->meta points to a **new** keyset. The keys in dest->meta are also copied with KEY_CP_COW, i.e. they are also copy-on-write keys. KEY_FLAG_COW_META is set on dest->flags. KEY_FLAG_RO_↩META is set on source->flags.

The source key will remain as a read-only key. This constraint is needed, because the source key is the only key we can free the resources on. If the data or the name would change in the source key, all COW-copied keys would suddenly have another value. For the same reason, the source key will need to live longer than all COW-copied keys from it.
A `keyCopy()` without `KEY_CP_COW` from an COW key will create a deep copy of the key. These keys are "normal" non-COW keys and can live on their own.
Every `key*()` method that modifies data on a COW-copied key will need to allocate new memory for this data and remove the `KEY_FLAG_COW_DATA` flag. Every `key*()` method that modifies the name of a COW-copied key will need to allocate new memory for this name and remove the `KEY_FLAG_COW_NAME` flag. Every `key*()` method that modifies metadata needs to make sure that the same happens for metakeys.
Keysets are not copy-on-write. A `ksDeepDup()` of a keyset with COW keys will create a keyset with deep-copied non-COW keys. Internally we may need a `ksCowDup()` function to create a keyset with copy-on-write keys from another keyset. Whether this function will be part of the public API is a point for discussion.

257.4.2 Full-blown copy-on-write implementation

Make Elektra's `Key` and `KeySet` data structures copy-on-write. This requires some major refactoring of code within `libelektra-core`. Code that does only interact with the data structures via the public `libelektra-core` API should not notice any differences. The `mmapstorage` plugin needs to be updated.
Unlike "mmapstorage-like COW implementation" keyDup, keyCopy, ksCopy and ksDup will always use COW. `ksCopy` and `ksDup` is needed for (de)duplication of metadata. Furthermore, the API has better usability if Key and KeySet behave the same, especially for bindings where duplication might happen implicitly.

257.4.2.1 Changes to <tt>Key</tt>

For the `Key`, we need to extract everything for the data and name into their own structs. This is done for memory-management reasons, as we need to track how many keys point to the same data and/or name.

```
struct _KeyData {
    union {
        char * c;
        void * v;
    } data;
    size_t dataSize;
    uint16_t refs;
};
struct _KeyName {
    char * key;
    size_t keySize;
    char * ukey;
    size_t keyUSize;
    uint16_t refs;
};
struct _Key {
    struct _KeyData * keyData;
    struct _KeyName * keyName;
    KeySet * meta;
    keyflag_t flags;
    uint16_t refs;
};
```

@mpranj's thoughts regarding moving name and data to separate structures:

> 1. If they [key name and data] are a separate entity, `mmapstorage` will need a flag once again for each of those. This is used to mark whether the data is in an mmap region or not. (or we find some bit somewhere that we can steal for this purpose)
>
> 1. Adding more indirections is probably not going to help performance. (I understand that we save memory here)

257.4.2.2 Changes to <tt>KeySet</tt>

For `KeySet`, we need to split out everything to do with the stored keys into a separate datastructure. This includes the array itself, the sizes and the hashmap.
Why don't we just add the number of references to the original `KeySet`?

- If we delete a copied KeySet, we don't know which KeySet is the original, so we couldn't decrement the counter. This could be dealt with storing a pointer to the original KeySet.

- If the original KeySet is deleted, we don't know which other KeySets point at the data, so updating their count would not work
- In similar fashion, if you update the original KeySet, the copied KeySets will also contain the new data (if the memory address does not change). This is unexpected behavior.

```
struct _KeySetData {
    struct _Key ** array;
    size_t size;
    size_t alloc;
    Opmphm * opmphm;
    OpmphmPredictor * opmphmPredictor;
    uint16_t refs;
};
struct _KeySet {
    struct _KeySetData * data;
    ksflag_t flags;
    uint16_t refs;
};
```

257.4.2.3 Reference Counting

We need reference counting for the internal COW datastructures. We do it the same way reference counting currently works for `Key` and `KeySet`. One tweak though is that the refcount should never be 0, as this does not make sense for internal datastructures.
This means we always increment the refcount after creation and always decrement before deletion, so that the refcount is never zero. An example implementation is shown below:

```
static void keySetValue(Key * key, void * value, size_t size) {
  // [...] removal of current value from key
  struct _KeyData data = keyDataNew (value, size);
  keyDataIncRef (data);
  key->data = data;
}
static void keyDel(Key * key) {
  keyDataDecRef (key->data);
  keyDataDel (key->data);
  // [...] other cleanup
}
```

257.4.2.4 Variation 1 - RcBuffer

Instead of using different structs for `_KeyData`, and `_KeyName` use a more generic struct for reference counting. This would avoid some duplication on the reference counting code for the key. Keysets will still have their own data struct, as it contains more than just a pointer and a size.

```
typedef struct {
        void * data;
        size_t size;
        uint16_t refs;
} RcBuffer;
struct _Key {
        RcBuffer * uname;
        RcBuffer * ename; // will be removed soon
        RcBuffer * value;
        KeySet * meta;
        keyflag_t flags;
        uint16_t refs;
};
```

257.4.2.5 Possible Edge Cases

In general, it should be possible to always do copy-on-write. From a users perspective, copy-on-write copies of a key (and a keyset) should behave the same. There is, however, one edge case: the user modifying the value of a key directly. This is shown in the following example:

```
Key * key;
struct foo myFoo = {
  .x = 0
};
keySetBinary (key, &myFoo, sizeof(myFoo));
Key * dup = keyDup (key);
((struct foo *)keyValue (key))->x = 1;
// with COW
assert (((struct foo *)keyValue (dup))->x == 1);
// without COW
assert (((struct foo *)keyValue (dup))->x == 0);
```

This edge case can be accounted for by providing a private function `keyDetach`, that forces that the key has its very own copy of the data.

```
((struct foo *)keyValue (keyDetach(key)))->x = 1;
// with COW
assert (((struct foo *)keyValue (dup))->x == 0);
// without COW
assert (((struct foo *)keyValue (dup))->x == 0);
```

257.4.2.6 Compatibility with <tt>mmapstorage</tt> plugin

If we do change the internal data structures it makes much more sense to fix the cache and mmapstorage afterwards (or in tandem). The most important constraint for mmap is that any structure (or bytes) that is an allocation unit (e.g. we malloc() the bytes needed for KeySet struct, so this is an unit) needs to have a flag to determine whether those bytes are actually malloc()ed or they are mmap()ed. Thus all the newly added structures as proposed will need some kind of an mmap flag.
`mmapstorage` only calls `munmap` in some error cases, so basically `munmap` is almost never done and the keyset is never invalidated.
During `kdbSet` the storage plugins always write to a temp file, due to how the resolver works. We also don't need to mmap the temp file here: when doing `kdbSet` we already have the `KeySet` at hand, mmap-ing it is not needed at all, because we have the data. We just want to update the cache file. The `mmap`/`munmap` in kdbSet are just so we can write the KeySet to a file in our format. (`mmap()` is just simpler, but we could also `malloc()` a region and then `fwrite()` the stuff)
Therefore the only case where we return a `mmap()`ed KeySet should be in `kdbGet`.
When the `mmapstorage` was designed/implemented, not all structures had refcounters, so there was no way to know when a `munmap` is safe. This was simply out of scope at that point in time.
If refcounting is now implemented for all structures, we might be able to properly `munmap` in future.
Two ideas to deal with this in conjunction with our reference counting implementation:
If we have `free` function-pointer along side the refcount, `mmapstorage` (and also other plugins with different allocators) could set it to their own implementation. To mimic the current behavior of `mmapstorage` this would point to a no-op function. However, we could also improve things and keep track of when all data has been freed and only then call `munmap`.
Another simpler way to avoid the flag, which doesn't really allow for further improvements, would be using the refcount. `mmapstorage` could set the refcount to a value that is otherwise illegal. This would allow us to detect the keys. Depending on the refcount implementation good values would probably be 0 or UINT16_MAX. The special value would have to ignored by all refcounting functions (inc, dec, del) and turn the functions into no-ops.

257.4.2.7 Possible Optimizations

- This approach requires more allocations than previously. We have not fully benchmarked whether this is a big issue. One optimization could be an expanding "pool" of `_KeySetData`, `_KeyData` and `_KeyName`. We could then allocate multiple of them at the same time, and borrow and give back instance from and to the pool.
- Embed the `KeySet * meta` directly in `struct _Key`. This may help with performance in cases we need metadata. It will, however, increase memory usage. This should only be considered after some benchmarking shows this is a real issue.

257.5 Memory comparison of COW approaches

The following calculations are based on the AMD64 platform. All results are in bytes unless stated otherwise.
Example key: `user:/hosts/ipv6/example.com = 2606:2800:220:1:248:1893:25c8:1946`
We want to measure the following properties for the key:

- Empty Key: size of a simple malloc of the key struct
- Empty Key (with name): size of simple malloc of all structs, so that the key has a name, but without including the size of the name
- Empty Key (with name + data): size of a simple malloc of all structs, so that the key has a name and data, but without including the size of the name or data
- Single Example Key: a single instance of the key defined above

- Example Key + 1 Duplicate: two instances of the key defined above, one of them is a duplication of the first
- Example Key + 2 Duplicates: three instances of the key defined above, two of them are duplications of the first

Approach	Empty Key	Empty Key (with name)	Empty Key (with name + data)	Single Example Key	Example Key + 1 Duplicate	Example Key + 2 Duplicates
Current Implementation	64	64	64	153	306	459
mmapstorage-like COW implementation (without additional pointers)	64	64	64	153	217	281
mmapstorage-like COW implementation (with additional pointers)	80	80	80	169	249	329
Full-blown COW implementation	32	72	96	185	217	249
Full-blown COW implementation - Variant 1 (RcBuffer)	40	88	112	201	241	281

We want to measure the following properties for the keyset:

- Empty KeySet: size of a simple malloc of the keyset struct
- Empty KeySet (with data): size of a simple malloc of all structs
- Example KeySet: size of a keyset with 15 keys + NULL byte
- Example KeySet + 1 Duplicate: two instance of Example KeySet, one of them is a duplication
- Example KeySet + 2 Duplicates: three instances of Example KeySet, two of them are duplications

Approach	Empty KeySet	Empty KeySet (with data)	Example KeySet	Example KeySet + 1 Duplicate	Example KeySet + 2 Duplicates
Current Implementation	64	64	192	384	576
mmapstorage-like COW implementation (without additional pointers)	64	64	192	384	576
mmapstorage-like COW implementation (with additional pointers)	64	64	192	384	576

Approach	Empty KeySet	Empty KeySet (with data)	Example KeySet	Example KeySet + 1 Duplicate	Example KeySet + 2 Duplicates
Full-blown COW implementation	16	64	192	208	224

257.5.1 Calculations

Raw data size:

- keyname : `28 + 1` = `29`
- unescaped keyname (measured): `25`
- data: `34 + 1` = `35`

Current Implementation:

- Empty Key [measured via `sizeof`]: `64`
- Empty Key (with name): `64`
- Empty Key (with name + data): `64`
- Single Example Key = `Empty Key + keyname + unescaped keyname + data` = `64 + 29 + 25 + 35` = `153`
- Single Example Key + 1 Duplicate = `Single Example Key * 2` = `153 * 2` = `306`
- Single Example Key + 2 Duplicates = `Single Example Key * 3` = `153 * 3` = `459`
- Empty KeySet [measured via `sizeof`]: `64`
- Empty KeySet (with data): `64`
- Example KeySet: `Empty KeySet (with data) + 16 * pointer to keys` = `64 + 16 * 8` = `192`
- Example KeySet + 1 Duplicate: `Example KeySet * 2` = `192 * 2` = `384`
- Example KeySet + 2 Duplicates: `Example KeySet * 3` = `192 * 3` = `576`

mmapstorage-like COW implementation (without additional pointers):

- Empty Key [measured via `sizeof`]: `64`
- Empty Key (with name): `64`
- Empty Key (with name + data): `64`
- Single Example Key = `Empty Key + keyname + unescaped keyname + data` = `64 + 29 + 25 + 35` = `153`
- Single Example Key + 1 Duplicate = `Single Example Key + Empty Key` = `153 + 64` = `217`
- Single Example Key + 2 Duplicates = `Single Example Key + Empty Key * 2` = `153 + 64 * 2` = `281`
- KeySets are not COW in this approach --> same as current implementation

mmapstorage-like COW implementation (with additional pointers):

- Empty KeySet [measured via `sizeof`]: `64`
- Empty Key [measured via `sizeof`]: `80`

- Empty Key (with name): `80`
- Empty Key (with name + data): `80`
- Single Example Key = `Empty Key + keyname + unescaped keyname + data = 80 + 29 + 25 + 35 = 169`
- Single Example Key + 1 Duplicate = `Single Example Key + Empty Key = 169 + 80 = 249`
- Single Example Key + 2 Duplicates = `Single Example Key + Empty Key * 2 = 169 + 80 * 2 = 329`
- KeySets are not COW in this approach --> same as current implementation

Full-blown COW implementation:

- Empty Key [measured via `sizeof`]: `32`
- Empty Key (with name) [measured via `sizeof`]: `Empty Key + sizeof(KeyName) = 32 + 40 = 72`
- Empty Key (with name + data) [measured via `sizeof`]: `Empty Key + sizeof(KeyName) + sizeof(KeyData) = 32 + 40 + 24 = 96`
- Single Example Key = `Empty Key (with name + data) + keyname + unescaped keyname + data = 96 + 29 + 25 + 35 = 185`
- Single Example Key + 1 Duplicate = `Single Example Key + Empty Key = 185 + 32 = 217`
- Single Example Key + 2 Duplicates = `Single Example Key + Empty Key * 2 = 185 + 32 * 2 = 249`
- Empty KeySet [measured via `sizeof`]: `16`
- Empty KeySet (with data): `Empty KeySet + sizeof(KeySetData) = 16 + 48 = 64`
- Example KeySet: `Empty KeySet (with data) + 16 * pointer to keys = 64 + 16 * 8 = 192`
- Example KeySet + 1 Duplicate: `Example KeySet + Empty KeySet = 192 + 16 = 208`
- Example KeySet + 2 Duplicates: `Example KeySet + Empty KeySet * 2 = 192 + 16 * 2 = 224`

Full-blown COW implementation - Variant 1 (RcBuffer):

- Empty Key [measured via `sizeof`]: `40`
- Empty Key (with name) [measured via `sizeof`]: `Empty Key + sizeof(RcBuffer)*2 = 40 + 24*2 = 88`
- Empty Key (with name + data) [measured via `sizeof`]: `Empty Key + sizeof(RcBuffer)*3 = 40 + 24*3 = 112`
- Single Example Key = `Empty Key (with name + data) + keyname + unescaped keyname + data = 112 + 29 + 25 + 35 = 201`
- Single Example Key + 1 Duplicate = `Single Example Key + Empty Key = 201 + 40 = 241`
- Single Example Key + 2 Duplicates = `Single Example Key + Empty Key * 2 = 201 + 40 * 2 = 281`
- Empty KeySet [measured via `sizeof`]: `16`
- Empty KeySet (with data): `Empty KeySet + sizeof(KeySetData) = 16 + 48 = 64`
- Example KeySet: `Empty KeySet (with data) + 16 * pointer to keys = 64 + 16 * 8 = 192`
- Example KeySet + 1 Duplicate: `Example KeySet + Empty KeySet = 192 + 16 = 208`
- Example KeySet + 2 Duplicates: `Example KeySet + Empty KeySet * 2 = 192 + 16 * 2 = 224`

257.5.2 Allocations & Indirections comparison of COW approaches

For allocations want to measure the following properties:

- Empty key: how many objects to allocate for an empty key
- Empty Key (with name): how many objects to allocate for an empty key + name
- Empty Key (with name + data): how many objects to allocate for an empty key + name + data
- Duplication: how many objects to allocate for a duplication
- Key + 1 Duplication: how many objects to allocate for a full key + 1 duplication
- Key + 2 Duplications: how many objects to allocate for a full key + 2 duplications

Approach	Empty Key	Empty Key (with name)	Empty Key (with name + data)	Duplication	Key + 1 Duplication	Key + 2 Duplications
Current Implementation	1	1	1	1	2	3
mmapstorage-like COW implementation (without additional pointers)	1	1	1	1	2	3
mmapstorage-like COW implementation (with additional pointers)	1	1	1	1	2	3
Full-blown COW implementation	1	2	3	1	4	5

257.6 Decision

Implement the full-blown COW approach.

257.7 Rationale

- It is the most versatile option.
- No restrictions on the lifetime of `Key` and `KeySet` objects.
- Completely transparent to developers using Elektra's public API.

257.8 Implications

- The `mmapstorage` plugins needs to be updated

257.9 Related Decisions

- change tracking

- internal cache

257.10 Notes

Chapter 258

Cryptographic Key Handling

258.1 Problem

The crypto plugin applies cryptographic operations to Keys and KeySets. In order to do that it needs keys and initialization vectors (IV). The problem is how to retrieve or derivate those keys in a safe way and how to pass them on to the underlying crypto libraries (OpenSSL and libgcrypt at the time of writing).

258.2 Constraints

The solution must be reasonable to the user (not just experimental setups).
The key handling should follow best practises when dealing with cryptographic operations, thus:

- the combination of key and IV must not be used twice for symmetric encryption
- the key and IV material must be as random as possible (using an SNRG)
- the key is not exposed to other Elektra modules or the user

258.3 Assumptions

- The user knows or is willing to learn how to use GPG.
- The user is willing to configure `pinentry` on her system, as it does not always run out of the box (depending on the distribution)
- The user knows how to create and identify an asymmetric key pair using `gpg`

258.4 Considered Alternatives

We considered passing the key and IV in form of plugin configuration and metakeys, but this approach possibly exposes the key to other modules. Thus our constraints are violated.
Also we considered using the gpg-agent (or pcscd, which uses a similar protocol) for providing asymmetric cryptographic operations. The problem with gpg-agent is that it only provides operations which require the private part of the key-pair (i.e. signing and decrypting). We would still have to implement the counterpart operations (i.e. verifying and encrypting) on our own. Starting the gpg-agent and the whole interprocess communication is either tedious work (when implemented by oneself) or adds another dependency (when using libassuan). So we do not consider the gpg-agent to be a viable option.

258.5 Decision

258.5.1 General Approach

The introduction of a GPG interface enables the user to utilize her existing key-pairs for cryptographic operations in Elektra. The private key is used for encrypting a random sequence, which serves as seed for a key derivation

function (KDF). This way we can safely derivate cryptographic keys for symmetric value encryption. Both OpenSSL and libgcrypt have built-in support for the PBKDF2 (see RFC 2898).
The PBKDF2 needs an iteration number and a salt in order to work. Those values will be stored per Key as MetaKey.

258.5.2 Implementation Details

During the **mount phase** a random master password *r* is being generated. *r* is sent to the gpg binary for encryption. The resulting encrypted master password *m* is stored in the plugin configuration at `config/masterChallenge`.
During the **set phase** the master password *m* is sent to the gpg binary for decryption in order to retrieve *r*. The following steps will be repeated for every Key *k*, that is supposed to be encrypted. A random salt *s(k)* is generated. By applying the PBKDF2 (mentioned earlier) with *r* and *s(k)*, the cryptographic key *e(k)* and the initialization vector *i(k)* is being derived. The value of *k* will be encrypted using *e(k)* and *i(k)*. The seed *s(k)* will be encoded as prefix into the encrypted value.
During the **get phase** the master password *m* is sent to the gpg binary for decryption in order to retrieve *r*. The following steps will be repeated for every Key *k*, that is supposed to be decrypted. The salt *s(k)* is read from the encrypted message. By applying the PBKDF2 with *r* and *s(k)* the values of *e(k)* and *i(k)* are restored. Then the encrypted message can be decrypted.

258.6 Rationale

The solution is reasonable to all users who are in favor of GPG. Also the solution might lead to a decline in dependencies (i.e. all cryptographic operations could be handled by the gpg binary).
The constraints considering the key handling are met.

258.7 Implications

To manipulate the plugin configuration during the **mount phase** a hook is required within the `BackendBuilder`. See pull request #733 for full discussion.

258.8 Related Decisions

258.9 Notes

Chapter 259

Default Values

259.1 Problem

- KeySet might get modified on access (hash rebuilds)
- Expectation that already all keys are there after `kdbGet()`
- No default value calculation

259.2 Constraints

- Should work with dynamic search for keys

259.3 Assumptions

259.4 Considered Alternatives

259.5 Decision

- spec-plugin does a lookup for values (Maybe also resolving missing fallback/override links?)

259.6 Rationale

259.7 Implications

259.8 Related Decisions

259.9 Notes

- #533
- #972

Chapter 260

Deferred Plugin Calls

260.1 Issue

Calling exported functions is a primary way for coordinating plugins in Elektra. Since some plugins encapsulate other plugins, exported functions from encapsulated plugins become unavailable.
Encapsulating plugins cannot implement and forward calls for every function exported by encapsulated plugins. Instead we need a generic mechanism for function calls to these encapsulated plugins.
Since some plugins also lazy-load encapsulated plugins the call mechanism is required to be able to defer these calls until the plugins are loaded.
For example when setting I/O bindings with `elektraIoSetBinding()` the exported function `setIoBinding` is called for all globally mounted plugins. Since global mounting is implemented using the "list" plugin which uses lazy-loading for its plugins the exported functions from the plugins are unavailable.
Other examples is the "multifile" plugin which use multiple plugins to support different file formats. These plugins also "hide" functions exported by encapsulated plugins.

260.2 Constraints

1. Plugins encapsulating other plugins shall be able to lazy-load them.
2. Callers shall not be entangled in encapsulating plugin details.
3. The encapsulation of the plugins shall not be broken.
4. The mechanism shall be generic.

260.3 Assumptions

1. The called functions do not return a value (e.g. `set`, `open`, `close`, ...). Callbacks can be used as return channel (see "Implications")

260.4 Considered Alternatives

- Encapsulating plugins export a function called `getEncapsulatedPlugins`. This would break encapsulation and break lazy-loading since to return the encapsulated plugin handles they have to be loaded.
- Compatible plugins export a function called `getInterfaces`. It returns a KeySet with well-known interface names like "notification" or "ioBinding" and the required functions (e.g. `open` & `close` for "notification" or `set` for "ioBinding"). While normal plugins simply return their interfaces, encapsulating plugins collect interfaces from its plugins and combine them into a single KeySet.

  The example below shows a KeySet from a plugin encapsulating two plugins "A" and "B". Both plugins implement the "notification" interface, plugin "B" also implements the "ioBinding" interface.

  – `/notification/#0/open`: address of `openNotification` of plugin "A"

- `/notification/#0/close`: address of `closeNotification` of plugin "A"
- `/notification/#1/open`: address of `openNotification` of plugin "B"
- `/notification/#1/close`: address of `closeNotification` of plugin "B"
- `/ioBinding/#0/set`: address of `setIoBinding` of plugin "B"

The upside of this approach is that it makes encapsulating plugins transparent to the caller: it does not know whether the plugin encapsulates other plugins. This approach breaks lazy-loading since for combining all interfaces the plugins have to be loaded and their `getInterfaces` functions have to be called.

260.5 Decision

Encapsulating plugins export a function called `deferredCall` with the declaration `void elektra↩ DeferredCall (Plugin * plugin, char * name, KeySet * parameters).` Encapsulating plugins shall save multiple deferred calls and call the exported functions specified by `name` passing the `parameters` KeySet when a plugin is initialized in the same order as received.
Plugins that support deferred calls shall have the following declaration for their functions `void somePlugin↩ Function (Plugin * plugin, KeySet * parameters)`. The calling developer is responsible for ensuring that the called functions have a compatible declaration. Encapsulated plugins that do not export a specified function name are omitted.

260.6 Argument

The solution allows changing encapsulating plugin implementations without breaking callers.

260.7 Implications

The called function receive their parameters via a KeySet.
While called functions could return data using the `parameters` KeySet (or a separate KeySet) there is no defined moment when the data can be collected. Defining such a moment would break the lazy-loading constraint. It is recommended to use callbacks passed as `parameters`. Callback function declarations are not limited by this decision.

260.8 Related Decisions

- Elektra's invoke functionality will be extended to also allow us to use deferred calls with new functions:
- `int elektraInvokeFunctionDeferred (ElektraInvokeHandle * handle, const char * elektraPluginFunctionName, KeySet * ks)` which defers a call if the plugin exports `deferredCall`.
- `void elektraInvokeExecuteDeferredCalls (ElektraInvokeHandle * handle, ElektraDeferredCallList * list)` which executes deferred calls for an encapsulated plugin loaded with invoke.
- Functions supporting deferred calls should allow for multiple calls (i.e. they should be idempotent). This leaves state at affected plugins and does avoid duplicating state (e.g. "was this function called for this plugin before?") in encapsulating plugins.

260.9 Notes

Implemented in libelektra-invoke.

Chapter 261

Elektra Web structure

261.1 Problem

For Elektra Web, there needs to be a way to remotely manage instances and groups of instances (clusters). The remote configuration of a single instance is simple. However, to manage multiple instances, we need to store the information to access the daemons, as well as information about the grouping (clusters) of daemons.

261.2 Constraints

- We need to be able to manage single, as well as multiple instances.
- We need to be able to group instances and manage them together.

261.3 Assumptions

261.4 Considered Alternatives

- Accessing the elektrad daemons directly
- Using a separate daemon (clusterd) to manage multiple instances
  - Using one of these daemons for each cluster
  - Using one daemon for all clusters

261.5 Decision

Use one cluster daemon (clusterd) to manage all clusters and instances.

261.6 Rationale

Accessing the elektrad daemons directly would require us to store all the information in the client, which is not a good idea if we want to be able to switch machines and still be able to access all data. Thus, there should be a single point of entry that the client connects to.
Using a daemon for each cluster would make it harder to create new clusters, because the user would have to manually set up a new daemon for each cluster. Another problem with this approach is that we still don't have a single point of entry, so we would have to store information to connect to the daemons on the client.
Using one daemon to manage all instances and clusters is the easiest solution, that way the client simply connects to that daemon, which forwards requests to single instances or multiple instances at once. In this case, the cluster daemon can also serve the client, further simplifying the whole structure, because we don't need to host the client on a separate web server.
Another advantage of this solution is that both clusterd and elektrad are using nodejs, which means that a wrapper script can set up and start both daemons easily.

There are still some other issues, like possible conflicts when adding an instance multiple times, or assigning an instance to multiple clusters. The cluster daemon should prevent adding the same instance twice. It also won't be possible to add an instance to two clusters. If an instance is part of a cluster, single instances can still be configured, but all configuration options that are set in the cluster cannot be modified.

261.7 Implications

- There needs to be a single point of entry for the web client to connect to.

261.8 Related Decisions

- Elektra Web Recursive Structure decision

261.9 Notes

- `Discussion in the pull request`

Chapter 262

Elektra Web recursive structure

262.1 Problem

After deciding how to remotely manage instances and groups of instances (clusters) with Elektra Web, there is still the issue of recursively nested clusters (clusters of clusters).

262.2 Constraints

- We need to be able to manage single, as well as multiple instances.
- We need to be able to group instances (into clusters) and manage them together.
- We need to be able to group clusters and manage them together.

262.3 Assumptions

- Use one cluster daemon (clusterd) to manage all clusters and instances.

262.4 Considered Alternatives

- Allowing clusterd instances to add clusterd instances (instead of just elektrad instances)
- Managing the hierarchy in a single clusterd instance

262.5 Decision

Managing the hierarchy in a single clusterd instance.

262.6 Rationale

Accessing clusterd instances from other clusterd instances would mean that there also needs to be some authentication between those, complicating the set-up process. Furthermore, it would not be as easy to deal with conflicts, as the client might connect to a clusterd instance that belongs to another clusterd, in which case it would not be aware of the constraints of the parent clusterd instance.
There is also still the issue that using a daemon for each cluster would make it harder to create new clusters, because the user would have to manually set up a new daemon for each cluster.
Using one daemon to manage all instances and clusters, including sub-clusters is the easiest solution, that way the client simply connects to that daemon, which forwards requests to single instances or multiple instances at once. Furthermore, the cluster daemon could deal with conflicts easily as it is aware of the whole network. In this case, the cluster daemon can also serve the client, further simplifying the whole structure, because we don't need to host the client on a separate web server.

262.7 Implications

- There is a single point of entry for the web client to connect to.

262.8 Related Decisions

- Elektra Web Structure decision

262.9 Notes

Chapter 263

Empty Files

263.1 Problem

An empty KeySet is passed to kdbSet(). What is the correct persistent representation?

263.2 Constraints

263.3 Assumptions

- User does not want empty files lying around everywhere.
- User wants to come back to a clean situation using Elektra

263.4 Considered Alternatives

- no file, no empty directories
- keep directories, remove configuration file
- plugins write minimal, syntactical-valid configuration file
- plugins do whatever they think is correct
- remember initial situation at mounting time and restore it when empty key is passed (seems inefficient and complicated?)

263.5 Decision

Remove files on empty KeySet.

263.6 Rationale

- allows user to undo what a previous kdbSet() did
- easy to understand semantics
- makes storage plugins easier (do not need to remove files)

263.7 Implications

- less empty files are left
- no invalid empty files (yajl bugs)

263.8 Related Decisions

263.9 Notes

Chapter 264

Ensure

264.1 Problem

Applications want to ensure that some functionality (hooks) is present in Elektra.

264.2 Constraints

264.3 Assumptions

264.4 Considered Alternatives

- Keep `kdbEnsure` (Rejected because it is too flexible and can be called many times. Furthermore, `kdb↩ Open` would build up configurations that get removed afterwards.)
  - reduce for only global plugins, as the partial other functionality is confusing and not needed
  - find solution that list plugin is not needed
  - only as implementation detail below libraries (e.g. like done for notification)
- Specific APIs per plugin, Rejected because:
  - difficult for application developers
  - every plugin would need to design new APIs
- Have a new API: `KDB * kdbOpen (Key * parent); int kdbConfigure (KDB * handle, KeySet * contract, Key * parentKey); KDB * kdbOpenDefault (Key * parent);` The new kdbOpen only does the absolute minimum work, in particular it doesn't set up any global plugins. If you use kdbOpen you must call kdbConfigure otherwise kdbGet will fail. kdbConfigure configures global plugins (basically just a renamed kdbEnsure). Lastly, kdbOpenDefault does more or less what the old kdbOpen does. It sets up the default case and you can call kdbGet immediately. But you cannot call kdbConfigure after kdbOpenDefault. Rejected because of API bloat and introduction of further state in `kdb`.

264.5 Decision

Integrate `kdbEnsure` in `kdbOpen(Key *errorKey, KeySet *contract)` but only allow hooks.

264.6 Rationale

- can immediately build up correct plugin positioning
- does not allow starting applications if the contract cannot be fulfilled
- simplest and minimalistic solution

264.7 Implications

`elektraNotificationOpen` will be renamed and only return a contract KeySet:

```
KeySet * errorKey = keyNew ("/", KEY_END);
KeySet * contract = ksNew (0, KS_END);
elektraNotificationContract (contract, iobinding, errorKey);
```

The same for gopts:

```
elektraGOptsContract (contract, argc, argv, environ, errorKey);
```

Finally, we create `KDB` with the contracts we got before:

```
KDB * kdb = kdbOpen (contract, parentKey);
```

Opening `KDB` will fail if any of the contracts cannot be ensured.

As the `contract` gets copied, at any point after `kdbOpen` the contract can be safely deleted:

```
ksDel (contract);
```

The cleanup of the global plugins happens within:

```
kdbClose (kdb, errorKey);
```

It is safe to use the contract `KeySet` also for `kdbGet` and `kdbSet` invocations. Contract `KeySet`s only contain `Key`s below `system:/elektra/contract`. Therefore, normal `KeySet`s should not interfere.

264.8 Related Decisions

- Hooks
- Notifications

264.9 Notes

- Issue #2764
- Implemented in #3651

Chapter 265

Error Code Implementation

265.1 Problem

In the previous error concept it was very useful to generate macros as we often added new errors. The code generation itself, however, was not ideal for cross compilation.

265.2 Constraints

265.3 Assumptions

- Error codes do not change very often

265.4 Considered Alternatives

- Migrate to a more modern and easier way to generate code with our mustache system, and let this generate the (mapping) code for all compiled languages (C, C++, Java, Rust, Go). All we get out of this is the removal of `std::cout << ...` code from C++ but not much more.
- Migrate to CMake code that generates such macros/classes. Mustache templates have to be supplied with the input data somehow. Either we have to use a custom executable that is compiled at build time. In that case we would just get rid of the `std::cout << ...` in the C++ code, but not much else would change. The other option is to use the default mustache executable, which is a Ruby script and therefore requires Ruby to be installed. Also `kdb gen` cannot be reused, since that would require compiling kdb first, which needs kdberrors.h.

265.5 Decision

Write down the few macros manually, and also manually write down exceptions for the language bindings (and also the mappings from Elektra's internal errors to nice errors specific for the languages).
Since error codes and crucial parts Elektra's core implementation will not often change this is the best approach with minimal effort.
The existing code will be refactored so that error macros directly call macros.
When adding a new error code, language bindings have to be adapted accordingly such as the Rust or Java Binding.

265.6 Rationale

- Updates to error codes will imply syncing every binding. This though should happen in very rare circumstances.

265.7 Implications

- The current bindings stay untouched.
- Tutorial on how to write a binding should be written.

265.8 Related Decisions

265.9 Notes

- See also Issue #2814
- See also Issue #2871

Chapter 266

Error codes

266.1 Problem

The current error concept has disadvantages in following regards:

- A lot of redundant errors: At the moment, each new plugin introduces new error codes which led to about 210+ error codes. Many of those errors are duplicated because developers did not know or search for a similar error which is already present. This concept should group similar errors together so that there is one coherent and consistent state again.
- Hard to manage specification file: Since every developer adds its own error individually, a lot of merge conflicts happen which makes contributing to the codebase unpleasant. Additionally, if you want to reuse any error you have to scrape to the whole file with ∼1300+ lines. As there is no senseful ordering or scheme behind the errors (since they grew by time), it is a hassle to find the correct error code. The new concept should standardize errors, making it easy to categorize errors from new plugins and avoid merge conflicts.
- No sensible way for application developers to use error codes from Elektra: If developers of plugins/ external tools using Elektra want to react to errors, they have to be very specific. At the moment there is no possibility to catch all errors easily which force a certain behavior. Eg. if there happens a temporary recoverable error, developers have to catch for every specific error code rather than a general hierarchical error. The new concept should make it easy to react to errors as they are sensefully grouped together and are hierarchically structured.

266.2 Constraints

- Error codes/numbers must stay but can be changed to another format (eg. Strings)
- Supporting multiple programming languages
- Supporting Elektra's Plugin System

266.3 Assumptions

266.4 Considered Alternatives

- Removing the specification file without requiring error numbers
- Adding the key of the occurred error to the API which permits reading information from additional metadata such as an error message provided by a specification author. Reason against: The description of the key should already provide such information. Doing it in an extra key would imply redundant information.
- Removal of warnings with error codes from the specification file.

Various projects and standards:

- GStreamer: This project uses 4 domain type errors which are suited to their project: CORE, LIBRARY, RESOURCE or STREAM. Every domain type has further sub error codes which are numbered from 1-x where 1 is a general purpose error "FAILED" which should be used instead of inventing a new error code (additional enum). You can see an example of enum errors here
- Apache httpd: This project does not use any error codes at all. They solely rely on the printed message and pass various other information along like file, line, level, etc. The primary function they use can be seen here
- Jenkins: Since Jenkins is a java project they have inheritance of errors by nature. They mostly use reaction based Exception such as `MissingDependency`, `RestartRequired`, `FormFillValidation`, `BootFailure`, etc. Some exceptions even have more concrete exceptions such as a `NoTempDir` which inherits from `BootFailure`. A very similar approach will be implemented by Elektra, except that it is a C project and will use error codes.
- Postgresql: Postgres has one of the most advanced error concepts among all investigated projects. It also uses one bigger specification file which is parsed and generates multiple header files. Also noteworthy is that they once had multiple files containing error codes and merged them into a single one (commit #ddfe26f). Errors are a string made up of 5 chars, where the first two chars indicate a certain class. This follows the SQLSTATE conventions. Currently they have 43 classes which all come from SQLSTATE. Postgres also throws additional errors but have to subclass it to one of the current 43 classes and have a special naming convention which have to start with a `P` in the subclass.
- etcd: Etcd's approach for errors are tightly coupled to the programming language Go as well as the gRPC standard which currently has 16 codes defined. Some of these errors are similar or identical to those which will be used in Elektra. Every error of etcd is associated with one of these categories and gets its own error message which is specified in this file. This concept though does not allow easy subclassing which might be useful (eg. further split FailedPrecondition into more specific errors like semantic and syntactic errors)
- Windows Registry: The registry does not use any specific error concept but takes the standard Win32 Error Codes. These are neither hierarchical nor have any special ordering. Basically it is the same as Elektra has now except for no duplicated errors.
- macOS X plist: Just like Windows, plist uses standard macOS X errors which is a huge catalog of unordered return codes as integers.
- SNMP Standard: Being a standard network protocol, error codes are very specific to the domain itself. A list can be found here and would not meet the needs of Elektra at all.
- POSIX: Returning a non-zero value and retrieving the concrete information from `errno` would not suffice for Elektra as it is too simple. It would not solve any of our current problems like having excessive uncategorized codes for errors.

266.5 Decision

All "fatal" errors will be converted to "errors" as the distinction is not relevant.
Unused errors will be removed from the specification.
Errors will be categorized into logical groups with subgroups. Each error will be made up of 5 characters, where the first 2 character indicate the highest level and character 3 to 5 will be used for subgrouping. Errors are prepended with the letter `C` which is the abbreviation for "Code".

- Permanent errors C01000
  - Resource C01100
    - Out of Memory C01110
  - Installation C01200
  - Logical C01300
    - Internal C01310
    - Interface C01320

        * Plugin Misbehavior C01330
- Conflicting State C02000
- Validation C03000
  - Syntactic C03100
  - Semantic C03200

To see an explanation of the categories along with a guideline on how to categorize please see the Error Codes Guideline

266.6 Rationale

The grouping of errors will allow developers to filter for specific as well as more general errors to correctly react to them programmatically. The new concept will permit additional subgrouping of errors in case it might be needed in the future.
Splitting/merging/rearranging any category should only be done by a decision (such as this file here). Elektra developers should not be able to generate a new category as they wish because it would lead to the same proliferation of errors as we had before.
These categories are chosen because they can help developers to react programmatically and cover the majority of use cases to our present knowledge. If there is ever the need for another reaction based category, it can be extended very easily.

266.7 Implications

The specification file will stay but should be untouched in most of the cases in the future. Also the C++ code generation file which uses the specification will stay as it is easier to change categories.
Current errors will be migrated.

266.8 Related Decisions

- Error Message Format Shows the new format of the error message
- Error Codes Guideline Shows how to categorize errors

266.9 Notes

Chapter 267

Error message format

267.1 Problem

Too verbose error message. Currently for every error, 9 lines are shown in which most of them are not relevant to end users/administrators. One goal is to reduce the verbosity of such messages and let users/administrators see only information they need.

267.2 Constraints

- Supporting multiple programming languages
- Plugin System
- Error Code should be preserved

267.3 Assumptions

267.4 Considered Alternatives

Possible variations on what message should be displayed, e.g., to keep the mountpoint information or on how wordings should be (with or without "Sorry, ...", coloring of certain parts of a message, etc.)
Examples would be to

- Leave out the "Sorry" in the error message or leave the introduction sentence completely
- Drop `At`, `Mountpoint`, `Configfile`, `Module`. This information though yields useful information or was even added as a request
- Show mountpoint, configfile, module, etc in beneath the general introduction message. Eg. `The command kdb set failed while accessing the key database on mountpoint (...) with the info`
- Incorporating the description in another ways: `Reason: Validation of key "<key>" with string "<value>" failed. (validation failed)`
- Use one command line option to show all additional info which gets hidden per default from now on instead of two
- Color the main message differently compared to the general introduction message
- Do not color messages as it might confuse users with overwhelming many colors
- Do not print out the error code. It is useful though for googling

267.5 Decision

The error message has the current format:

```
The command kdb set failed while accessing the key database with the info:
Sorry, the error (#121) occurred ;(
Description: validation failed
Reason: Validation of key "<key>" with string "<value>" failed.
Ingroup: plugin
Module: enum
At: ....../src/plugins/enum/enum.c:218
Mountpoint: <parentKey>
Configfile: ...../<file>.25676:1549919217.284067.tmp
```

The new default message will look like this:

```
Sorry, module `MODULE` issued [error|warning] `NR`:
`ERROR_CODE_DESCRIPTION`: Validation of key "<key>" with string "<value>" failed.
```

The `NR` will be the color red in case of an error or yellow in case of a warning while `MODULE` will be the color blue. Optionally a third line indicating a solution can be added. Eg. for a permission related error there would be a third line:

```
Possible Solution: Retry the command as sudo (sudo !!)
```

To avoid losing information, the user can use the command line argument `-v` (verbose) to show `Mountpoint`, `Configfile` in addition to the current error message. Furthermore a developer can use the command line argument `-d` (debug) to show `At` for debugging purposes.

267.6 Rationale

The new error message is much more succinct which gives end users more relevant information. Furthermore the solution approach still holds all necessary information if requested by users.

267.7 Implications

`Description` will be incorporated into `Reason` whereas the `Module` will be incorporated into the general sentence starting the error message.

267.8 Related Decisions

- Error Codes Shows how the new error codes are meant to be

267.9 Notes

Chapter 268

Global KeySet

268.1 Problem

Some plugins need to communicate more data than is possible to do with metadata. This can limit the functionality of plugins, which need to exchange binary or very complex data.

268.2 Constraints

268.3 Assumptions

268.4 Considered Alternatives

268.5 Decision

To make the communication between plugins easier, plugins will additionally get a handle to a global keyset via `elektraPluginGetGlobalKeySet()`. The global keyset is tied to a KDB handle, initialized on `kdbOpen()` and deleted on `kdbClose()`.
The global keyset handle is initialized and accessible for all plugins except manually created plugins (by calling e.g. `elektraPluginOpen()`).
This decision removes the need to exchange information between plugins via the parentKey.

268.6 Rationale

The need for a global keyset arose when developing a global cache plugin. A global cache plugin needs to store internal and binary information for the KDB, which is simply not possible with metadata.

268.7 Implications

Plugins are responsible for cleaning up their part of the global keyset.

268.8 Related Decisions

268.9 Notes

Chapter 269

Replacing Maven as Java-related build system with Gradle

269.1 Problem

Originally, Maven was used as the build system for Elektra's Java-related modules. When it came time to modernize and improve Elekta's Java bindings, the question of which build system to use had to be answered.

269.2 Constraints

- Existing dependencies must be supported by the chosen build system.
- An established system with a supportive community would be of great advantage.
- Build performance should not be negatively impacted.
- Integration with existing build infrastructure is no more complicated than before.
- The benefits of the transition outweigh the potential costs, making it a worthwhile investment.
- Ease of use, community support, and flexibility were the most important factors to consider.
- The fact that Maven does not provide native support for Kotlin, which is becoming increasingly popular with developers, was also an important factor to consider.

269.3 Assumptions

The development team either has the necessary expertise and resources for a successful transition, including knowledge of the new build system and familiarity with its syntax and conventions, or the new build system should be easy and intuitive for the development team to use, especially if the team is not already familiar with the new system.

269.4 Considered Alternatives

- Ant is a popular build tool that uses XML files to define build processes. It is flexible and can be used for a variety of languages, but it can be more verbose and less user-friendly than other build tools.
- CMake is a build tool, which only has limited support for Java. It is popular for C/C++ projects and can be extensively customized, but it can be more difficult to use than other build tools and is not as flexible for projects that use multiple languages. Since Elektra is a C-based software project, this system would have had the advantage of already being in use.

269.5 Decision

We are switching from Maven to Gradle.

- Gradle is more flexible and customizable than Maven. It uses a Groovy-based DSL that allows developers to write custom plugins and scripts easily, making it easier to configure and automate complex build processes.
- Gradle is faster than Maven. It offers more efficient incremental builds, meaning it can quickly determine which parts of the codebase have changed and only rebuild those parts. This might not yet bring a great improvement but allows for future scalability.
- Gradle's dependency management system is more flexible and offers more advanced features than Maven, including the ability to manage transitive dependencies and exclude specific unwanted dependencies. It also supports the same dependency eco-system as Maven does.
- Gradle natively supports Kotlin.
- Gradle offers an easy migration path from Maven.

269.6 Rationale

Gradle is not only the modern alternative to Maven, using the same dependency management infrastructure, but also opens up new possibilities for the growth of Elektra's Java bindings and other Java-based language bindings.

269.7 Implications

- Changed the setup for building Java bindings to use Gradle instead of Maven.
- The build images needed to be modified to create the necessary dependencies for integrating the Java bindings using Gradle instead of Maven.
- Assets created will be transferred to Sonatype's repository so developers can use the dependency framework used by Gradle (and Maven).
- Developers need to switch to using Gradle.
- Leverage Gradle's caching capabilities to improve the performance of CI (continuous integration) builds. Gradle has several levels of cache. The global cache, which is shared by all projects built with Gradle on a given machine, stores artifacts such as JAR files and metadata downloaded from remote repositories and is used automatically. Using the local build cache is not beneficial for CI tasks. Depending on the actual build performance and whether it needs to be improved, configuring the remote build cache may be useful. It can be used to store the output of specific tasks or the entire build cache. Remote build caching can help speed up builds by reusing artifacts from previous builds, especially when building large projects with many dependencies - which is not currently the case with Elektra.

Chapter 270

High-level API

270.1 Problem

Projects usually do not want to use low-level APIs. `KDB` and `KeySet` is useful for plugins and to implement APIs but not to be directly used in applications.

270.2 Constraints

1. should be extremely easy to get started with
2. should be very hard to use it wrong
3. all high-level APIs should work together very nicely
    - same principles
    - same API style
    - same error handling
    - can be arbitrarily intermixed

270.3 Assumptions

- Thread-safety: a handle is the accepted better solution than having to care about whether it is reentrant, thread-safe, ...
- assumes that spec is available (either by compiled-in `KeySet` or exit after elektraOpen)
- many projects do not care about some limitations (no binary, no metadata) but prefer a straightforward way to get/set config
- when people hit limitations they fall back to direct use of `KeySet`, `Key`

270.4 Considered Alternatives

- storing errors in the handle:
    - Maintenance problem if error handling is added later
- only provide `KDB`, applications need to implement their own APIs:
    - reduces consistency of how the API is used
- only provide generated API
- assume type as `string` if not given
- force `default` to be part of parameters

270.5 Decision

We provide two high-level C APIs:

1. libelektra-highlevel (generic key-value getter/setter)
2. code generator (specified key-value getter/setter with function names, KeySets, or strings from specifications)

Furthermore, we will:

- have as goal that no errors in specified keys with default can occur
- if you use `elektraGetType` before getting a value, no error can occur when getting it later
- enforce that every key has a type
- use `elektraError` as extra parameter (for prototyping and examples you can pass 0)

270.6 Rationale

1. Very easy to get started with, to get a key needs 3 lines of codes (without error handling):

```
Elektra *handle = elektraOpen ("/sw/elektra/kdb/#0/current", 0);
printf ("number /mykey is " ELEKTRA_LONG_F "\n", elektraGetLong (handle, "/mykey"));
elektraClose (handle);
```

1. It is also easier to get started with writing new bindings.
2. User can combine the different APIs.

270.7 Implications

270.8 Related Decisions

270.9 Notes

- Currently it is not possible to combine low-level and high-level API.
- Hierarchical version not implemented and probably not needed.
- #1359

Chapter 271

High-level API Help Message

This decision *does not* assume code-generation is used. For the case of code-generation see the `Notes` section.

271.1 Problem

We want to allow to print the help message no matter what errors happened in `kdbOpen` or `kdbGet`.

271.2 Constraints

- `elektraOpen` should not return a broken `Elektra` instance.
- The help message can only be printed, if `elektraOpen` returns an `Elektra` instance and no `Elektra↩ Error`.

271.3 Assumptions

- We assume that the application in question was correctly installed.
- We assume `gopts` was mounted. This is not the default right now, but the code-generator template `highlevel` contains code that will mount `gopts`, if it is missing.
- We assume the application was called in *help mode*, i.e. with `--help`. Otherwise printing the help message is not possible, anyway.

271.4 Considered Alternatives

- Ignore all errors (in help mode): Not a feasible solution, because there may have been problems when reading the storage file and therefore, the help message may be broken or incomplete.
- Ignore all errors (in help mode), which occurred after the `gopts` plugin ran: Complicated to implement (we need to know about plugin order, etc.). Not actually necessary (see `Rationale`).

271.5 Decision

Ignore missing `required` keys (in help mode), but fail for every other error.

271.6 Rationale

Required keys **must** be provided by the user/admin and cannot come from another source (Elektra, app developer, etc.). Therefore they will be missing until the user makes changes to the KDB. Before that, no other error should occur (we assumed a correct installation). If a user runs `app` for the first time and receives an error about a missing required key, they will:

1. know what to do and add the key, thereby fixing the problem.
2. try `app -h` and see that it doesn't show a help message. They will probably continue with 3.
3. try `app --help` to find out more. The help message may or may not contain useful information. If not they may try 4.
4. read some other documentation to find out more. Ideally this leads them to 1.

In any case after this the user definitely know how to interact with the KDB. Since we assumed that there won't be any errors before the KDB was changed, we can assume that the user caused other errors by changing the KDB.

271.7 Notes

If code-generation is used, the situation is a little different. If the parameter `embedHelpFallback` is set to `1`, a fallback help message will be created from the specification originally passed to the code-generator and embedded into the application. The parameter also changes, how help mode is detected and ultimately allows the help message function (`printHelpMessage` by default) to always print a help message. Although it may not reflect changes, the user made to the specification.

Chapter 272

Holes and Non-leaf values in KeySets

A hole is the absence of a key, which has keys below it, e.g. if `some/key` is missing in a property file:

```
some = value
some/key/below = value
```

`some` has a non-leaf value. Another example of a non-leaf value in XML (`abc`): `<abc>value<def>value2</def></abc>` interpreted by xerces plugin:

```
abc/def = value2
abc = value
```

272.1 Problem

Config files ideally do not copy any structure if they only want to set a single key.

272.2 Constraints

- strongly hierarchically structured data must still be supported

272.3 Assumptions

272.4 Considered Alternatives

- data structure must always be complete
- prohibit non-leaves values

272.5 Decision

Support holes and values for non-leaves in a KeySet if the underlying format allows it.
If the underlying format does not support it and there is also not an obvious way how to circumvent it – e.g., JSON which does not have comments – holes and values in non-leaves can be supported with key names starting with ®elektra.

272.6 Rationale

- It fits very good to the idea of key-value.
- Some formats support it (e.g. XML supports non-leaves values; property-files support holes).
- It can be useful for migration purposes, e.g. there is `/some/key`, and later `/some/key/enable` gets added. Then it is beneficial if `/some/key` still can hold a value.

272.7 Implications

272.8 Related Decisions

272.9 Notes

Chapter 273

Logging

273.1 Problem

Both code comments and assertions are unfortunately not very popular. A quite efficient way to still get some documentation about the code are logging statements. In Elektra they are currently inconsistent and unusable. Thus there is an urge for this decision.

273.2 Constraints

- Logging must be disabled by default.
- This decision is irrelevant for plugins and bindings that are not written in C/C++.
- Should completely compile away with the cmake variable `ENABLE_LOGGER=OFF`
- Should support minimalistic, compile-time filtering (per modules and verbosity level) and some sinks (stderr, syslog or files)

273.3 Assumptions

- run-time problems are checked via assertions, not logged
- opinions about if logging should be to stderr or files differ
- filtering with grep is not enough
- per default there should be no output
- with ENABLE_LOGGER=ON only warnings and errors should be shown on stderr
- other sinks like syslog and file may log more (they are not immediately visible and distracting)
- performance is not so important (because logging is usually turned off anyway)

273.4 Considered Alternatives

- log similar to the warnings/error system work, discarded because of the run-time overhead and no use case why end users should see log statements.
- C++ logging library (boost, apache,..), discarded because C++ should not be in core
- libraries needed static initializing: problematic, logging should just work, even if application does not initialize anything
- using syslog: no info from which source file the logging statement comes from
- using journald: adds deps problematic for non-linux
- zlog: incompatible licence (LGPL)

273.5 Decision

Provide a Macro
```
ELEKTRA_LOG (int module, const char *msg, ...);
```
that calls
```
elektraLog ([as above], const char * function, const char * file,
            const int line, ...);
```
and adds current function, file and line to `elektraLog`'s arguments.
`elektraLog` is implemented in a separate `log.c` file. If someone needs filtering, logging to different sources or similar, he/she simply modifies `log.c`.

273.5.1 Severity

The severity passed to `ELEKTRA_LOG_` should be as in syslog's priority, except the error conditions which are not needed (asserts should be used in these situations).
So we have:

- `ELEKTRA_LOG_WARNING`: warning conditions
- `ELEKTRA_LOG_NOTICE`: normal, but significant, condition
- `ELEKTRA_LOG_INFO`: informational message
- `ELEKTRA_LOG_DEBUG`: debug-level message

273.5.2 Modules

To add a new module, one simply adds his/her module to `elektramodules.h` via `#define`:
```
#define ELEKTRA_MODULE_<NAME> <SEQNUMBER>
```
The module name <NAME> shall be consistent with module names used in `module:` of `src/error/specification`.

273.6 Rationale

A more complex system is overkill. Thus libraries should not have any effects other than what is described by their API, logging should nearly always be disabled.
A more "hackable" logger seems to be more suitable for individual needs. Having a separate `log.c` means that the logger can be changed without the need to recompile anything but a single file. It also removes the dependency of `stdio.h` from every individual file to a single place.

- Logging is very easy to use (only include `elektralog.h` and use `ELEKTRA_LOG`)
- Logging is still flexible enough (with modules, severity, file, line and function information)

273.7 Implications

The current VERBOSE would be turned off forever and the code within VERBOSE needs to be migrated to `ELEKTRA_LOG`.

273.8 Related Decisions

273.9 Notes

See CODING.md for recommendations how to use the logger.

Chapter 274

Lookup Every Key

274.1 Problem

On structures like maps or arrays there are different possibilities which keys are looked up in the KeySet and which are simply iterated.
Without any guidelines, applications would provide arbitrary inconsistent behavior.

274.2 Constraints

274.3 Assumptions

- Applications that have good reasons to ignore the guidelines (e.g. they only read from one namespace), are allowed to do so.

274.4 Considered Alternatives

- only lookup the roots and then iterate over the next keys
- let the applications do what they want without any guideline

274.5 Decision

Every key that an application wants to use, must be looked up with `ksLookup` using a cascading lookup key.

274.6 Rationale

- very simple rule, easy to understand, easy to follow
- provides consistent behavior (`spec` is always honored)
- `ksLookup` is quite cheap as it has only a few simple loops, only one allocation and less than 10% of CPU time in profiling, even in very simple applications with many lookups.

274.7 Implications

Needs some helper functions or support in bindings as it is a bit tricky to implement e.g. for arrays.

274.8 Related Decisions

- Arrays

274.9 Notes

Chapter 275

Memory Layout

275.1 Problem

`mmapstorage` leaks internals of the `struct _Key`.

275.2 Constraints

275.3 Assumptions

275.4 Considered Alternatives

- don't make `Key` opaque
- don't provide `mmapstorage`

275.5 Decision

Have versioning and memory layout checks[1] of cache files written by `mmapstorage`. The cache files get discarded if these checks fail.

275.6 Rationale

- will also discard cache files from different architectures
- representation of `struct _Key` can be changed as wanted

275.7 Implications

See Rationale.

275.8 Related Decisions

275.9 Notes

[1] A `KeySet` with known content gets written and mmapstorage checks if this `KeySet` was restored correctly.

Chapter 276

Multiple File Backends

A file backend refers to exactly one file per namespace. This file can be returned using `kdb file`.

276.1 Problem

In some situations a single mountpoint refers to more than one file per namespace:

- For XDG in the `system` namespace may contain several files (XDG_CONFIG_DIRS).
- A fallback file if some data cannot be stored in some format (Idea from @kodebach: writing the same content to several files, merging when reading)

276.2 Constraints

276.3 Assumptions

276.4 Considered Alternatives

276.5 Decision

Multiple File Backends are not supported in the case of writing files.
Multiple sources in one namespace only work, if the fallback KeySet is part of the mountpoint config. That way any change to the fallback KeySet would essentially make the whole thing a different mountpoint and thereby invalidate all guarantees.

276.6 Rationale

Writeable multiple file backends would:

- make it impossible for admins to modify content of all files using Elektra
- do not work atomically (a journal would be needed)
- do not work together with mmap (as it only checks one file for cache misses)

276.7 Implications

See Rationale.

276.8 Related Decisions

276.9 Notes

Chapter 277

Null

277.1 Problem

Null keys do not have a semantic and make API more difficult.

277.2 Constraints

277.3 Assumptions

277.4 Considered Alternatives

- remove null keys, i.e. keyValue always returns a string (e.g. empty string on new keys)

277.5 Decision

Null keys get clear semantics: they are for representing keys that do not have a value, i.e., only represent structure in configuration files.
For example, an INI `section` is a null key:

```
[section]
key = value
```

For example, in other config formats there might be keys without any value. Here `key1` would be a null-key and thus is different to the value of `key2`, i.e., an empty string:

```
key1
key2 = ""
```

277.6 Rationale

277.7 Implications

277.8 Related Decisions

- Key Name

277.9 Notes

Chapter 278

Relative

278.1 Problem

There is a different behavior of various plugins whether their name is absolute or relative, including:

1. mounting the same file somewhere else does not work
2. importing somewhere else (other than from where it was exported) does not work (See here)

278.2 Constraints

- at least the dump plugin must be able to handle its old files

278.3 Assumptions

- it still will be easy to support a workflow that exports/imports everything
- mounting across namespaces (user/system) does not make sense

278.4 Considered Alternatives

- allow relative/absolute plugins and mark them what they are, tools (e.g. import/export) use this knowledge and react accordingly. This would still not solve problem 1.)

278.5 Decision

Key names shall be relative to parent key name.

278.6 Rationale

Provides a better import/export/remount and also a more uniform experience between different plugins.

278.7 Implications

All plugins that had absolute paths were adapted, see #51.

278.8 Related Decisions

None

278.9 Notes

Chapter 279

REST API Documentation

279.1 Problem

A standard way of describing REST APIs offered by tools and plugins for Elektra is required to ease development for and usage of these. Because many good standards for describing APIs are out there already, an existing one shall be used.

279.2 Constraints

- The chosen standard should support Markdown syntax
- API descriptions created within the standard should be human-readable
- Only free software should be used
- To enhance reusability, there should be a separation of data modeling and API description

279.2.1 Soft-Constraints

- (Nice looking) documentation should be generateable from the plain documentation
- Created documentation should be testable by tools (automatic tests)

279.3 Assumptions

- There is a well-suited standard with enough (free) tools available to satisfy all or most constraints.
- Scenarios created by Elektra REST APIs are simple enough to be allegeable by the chosen standard, also in the future.

279.4 Considered Alternatives

- `Swagger`
- `apiary` with `API blueprints`
- `RAML`

279.5 Decision

The decision is to use `API blueprints` together with additional tools from its ecosystem.

279.6 Rationale

API Blueprints together with some (free) tools for it support all given constraints and also all soft-constraints. It also fits the current documentation style of the Elektra Initiative the most.
Additionally to modeling data apart from the API, **API Blueprints** also supports schemata modeling, which is more precise than giving examples for requests and responses.

279.7 Implication

- API descriptions should also follow other conventions like the usage of similar error codes.

279.8 Notes

There are many tools available to use with **API Blueprints**, for example the `CLI tool` of apiary. Other tools can be found on the `official website` of the standard.
Decision discussions have taken place in #917. An API proposal for cluster configurations was made in #912, whereas initial discussion started in #829.

Chapter 280

Script Testing

280.1 Problem

Writing portable shell code for testing command-line tools is difficult.

280.2 Constraints

- Should be able to record input/output/exit codes of command-line tools
- Should be aware of configuration settings (KDB), for example, restore it on changes

280.3 Assumptions

None.

280.4 Considered Alternatives

- pythonpaste
  - Pros:
    - easy to work with
  - Cons:
    - can only trace a single directory (would not work with /etc + ~)
    - extra dependency not in any distro
- robotframework
  - Cons:
    - additional (fat) dependency
    - integration with cmake?
    - does not allow one to capture stdout, stderr + return code
- expect
  - Pros:
    - interactive testing (e.g. for kdb mount)
  - Cons:
    - quite long for simple things (e.g. check /bin/true needs 4 lines)
    - new syntax for Elektra (TCL)
    - additional dependency

280.5 Decision

Develop shell recorder and tutorial wrapper.

280.6 Rationale

280.7 Implications

280.8 Related Decisions

280.9 Notes

- 12.11.2017: pythonpaste not maintained anymore, site is offline

Chapter 281

Semantics in Key Names

281.1 Problem

It can get quite cumbersome to find out about key interrelations like arrays.

281.2 Constraints

281.3 Assumptions

281.4 Considered Alternatives

The alternative would be to have semantics in key names, with following advantages:

- maybe less metadata to save memory (only array)

281.5 Decision

Do not encode any semantics into the key names. All semantics must be in metadata.
Nevertheless, there are guidelines (without any checks in `keySetBaseName`):

- `#` is used to indicate that array numbers follow.
- `®` is used to indicate that some information was encoded in the key name. This is usually only needed internally in storage plugins.
- The UTF-8 sequence `®elektra` (i.e. the 9-byte sequence `C2 AE 65 6C 65 6B 74 72 61`) is reserved, see key name documentation.

There are, however, rules and conventions which syntax to use for specific semantics. The `spec` plugin guards these rules.

281.6 Rationale

- for consistency, whenever possible, metadata should be preferred
- no escaping of key base names necessary
- it is very unlikely that `®elektra` collides with a real key base name a user wanted to have
- `®elektra` makes very clear that there is a special reserved meaning
- `®elektra` UTF-8 encoding decodes to "some character" + ® in many 8-bit encodings (including ISO 8859-1 aka Latin1 and Windows (Codepage) 1252, in the encoding `C`, however, you get `'"$'\302\256"elektra`')

281.7 Implications

281.8 Related Decisions

- Arrays
- Base Names

281.9 Notes

Chapter 282

C++ Unit Testing Framework

282.1 Problem

The previous unit testing framework started as hack to have a bit more than simple asserts. It is not easy to use (needs explicit enumeration of all test cases) and lacks really important features (e.g. output of the assertion that failed).

282.2 Constraints

- Must be BSD licenced
- Must be easy to use
- should be portable
- should support mocking

282.3 Assumptions

282.4 Considered Alternatives

- Continue with current framework
- Boost Unit Testing framework

282.5 Decision

- Keep C framework for C tests and ABI tests
- Google Unit testing framework `gtest` with code downloaded by CMake for systems where no source is packaged (Debian Wheezy, Arch Linux, Fedora,...) for C++ tests

282.6 Rationale

- Having the output of current values when an assertion fails in any case
- No listing of all test cases in main (but instead having test discovery)
- No more commenting out if you only want to run parts of the test-suite
- No more typos in test-suite namings
- xUnit output for jenkins

- value and type-parameterized tests
- Mock-Support (not available in gtest?)
- setup/teardown global+per test
- supports death tests
- writing many parts of it on our own adds to the total amount of code to write and maintain.
- integrations into IDEs

282.7 Implications

- It is not ideal to have different frameworks intermixed (C vs. C++ frameworks, but most code is C).
- In the end we have to write a lot of functionality ourselves anyway (e.g. comparing Keys and KeySets).
- Testsuite execution are already handled by cmake and kdb run-all.
- The selection of tests within a test suite does not play well with ctest.
- Rewriting all current tests to have unified behavior is a lot of work.
- Might not work for ABI compatibility tests.
- Mock only by extra framework.

282.8 Related Decisions

- Script Testing

282.9 Notes

- We had discussions on Mailinglists
- We had discussions in #26

Chapter 283

EXPLANATIONS

We base our decision process and template on:

- ''using patterns to capture architectural decisions'',
- arc42 decisions,
- ADR, and
- RFCs in rust-lang.

This document describes every section of our TEMPLATE.md.

283.1 Problem

Clearly define:

- the context in which the problem exists
- your observation of the problem
- ideally an example for that problem

283.2 Constraints

List all constraints given by:

- use cases
- requirements
- Elektra's goals
- guidelines, e.g. documentation guidelines
- standards, e.g. C99
- other decisions
- the scope (i.e. describe what shouldn't change)

  Note: The decision (but not necessarily the solutions) must fulfill all constraints.

283.3 Assumptions

Assumptions are often overlooked, so this section needs special care and honesty. Assumptions are what we believe to be true but do not or cannot really know, e.g.:

- what users will accept
- perceived usability
- if the implementation will be faster/slower
- estimations of costs
- problems/risks that might turn up

  Note: The decision (but not necessarily the solutions) must not break any assumptions.

283.4 Solutions

This is a list of all solutions and a rationale why not-chosen solutions were not taken, e.g. because:

- the solution does not solve the whole problem
- some constraints or assumptions are violated
- another solution:
  - solves the problem better
  - is more in line with Elektra's architecture
  - better supports Elektra's goals
  - better fulfills non-functional requirements

283.5 Decision

Here should be a detailed description of the best solution, i.e., the decision. It should make clear how the implementation should be done.

Referring back to the solutions written above is allowed.

283.6 Rationale

Give all details why the solution:

- solves the problem best
- is best in line with our goals
- fulfills all constraints and assumptions

Also describe all drawbacks the solution has.

283.7 Implications

Here is a full description of everything that the decision will change or whatever needs to be changed because of the decision. This can be:

- effect on other decisions, goals, etc.
- non-obvious implementation tasks, e.g. changing in different needs to be implemented

- which issues get solved
- which documentation needs to be updated
- which concepts change
- which guarantees are added/removed

283.8 Related Decisions

This section has links to other decisions with description what the relation is. One-side relations are allowed, not every decision must link back. Decisions that give constraints must be listed in "Constraints" above.

> Guideline: Links to decisions should be in the form `../step of decision/name of decision.md`. In particular, they should always contain the step of the decision, even if they are in the same directory. This makes renaming issues easier.
>
> Note: Sometimes the best solution is only understood if the relation between decisions becomes clear. Make sure that everything that requires updates to a decision, is listed as "Constraints" or "Assumptions".

283.9 Notes

Here is a full list of off-line discussions, issue trackers, PRs etc. related to this decision. Preferable it is linked, but if it is not possible, it can also be in full-text here. If particular information is important and not present in any sections above, please quote it here.
Any incomplete and unexplored idea/opinion, which is not complete enough to be in "Solutions", can be written here. For example, if it is obvious that the idea does not even solve the problem. Unlike the main decision and solutions, text in the notes does not need rationale.
Furthermore, the author, acknowledgments, dates etc. can be written here.

Chapter 284

Decisions

284.1 Introduction

Before you write your first decision, read about the decision process and steps.

284.2 Meta-Information

Even though they use the decision template, following decisions are not decisions:

- Template
- Explanations

Chapter 285

Steps

This document describes all steps a decision can run through.

```
flowchart LR
  s((Start)) --> Drafts --> prob(Problem Clear) --> alt(Solutions Clear) --> review(In Review) -- merge -->
       Decided
    -- merge --> part(Partially Implemented) --> Implemented
  %% Shortcuts:
  s --> Decided
  Drafts -- merge --> Decided
  prob -- merge --> Decided
  alt -- merge --> Decided
  Decided -- merge --> Implemented
```

Additionally, decision that are not yet "Decided" may be become "Drafts", "Rejected" or "Postponed" at any point, e.g., if the decision author stops working on the decision.
The label `merge` on the edges means that a decision PR must be merged before it can target the next step.

> The first PR for a decision usually creates the decision in the "Drafts" state. If during the reviews of this PR it becomes clear that the decision is further along, can be moved to "Problem Clear", "Solutions Clear" or even "In Review" before the merge.

Short summary:

1. The first PR for a decision is created as "Drafts" (recommended!), "Problem Clear", "Solutions Clear" or "Decided".
2. A decision PR that wants to merge a decision as "Decided" must:
   (a) Only contain changes to one decision.
   (b) Already start with the decision as "Decided".
   (c) Already have a clearly stated problem and all the solutions fully explored.
   (d) Only contain discussions/reviews about deciding between one of the solutions or details thereof.
   (e) If there is any change in the direction of the decision then the PR cannot be merged as "Decided". The decision must at least be moved back to "In Review".
3. After a decision is merged as "Decided", it can be moved to "Partially Implemented" or "Implemented" by any PR.

The rest of the document describes the steps in full details.

285.1 Drafts

> This step is highly recommended and it is even required if the problem is not yet clear to all the core developers.

The first step is to create a PR with:

- **one** decision, where at least the "Problem" is filled out and "Decision", "Rationale" and "Implications" are **not** yet filled out.

- optional backlinks from related decisions.

This step is brainstorming for completely new ideas.

> No agreement about anything is needed to merge decisions in this stage.

285.2 Problem Clear

This step is very important:

- it clarifies the importance of the problem, answering: Why should we put our time and energy in this problem and not in another problem.
- it clarifies the scope of the decision.
- it clarifies relation to other problems.

Decisions will have much smoother further steps if this step is done carefully without prejudice. It is especially important that one shouldn't have a fixed mind-set about a preferred solution from the beginning.

> Everyone must agree that the problem exists and is worth solving so that a decision PR in "Problem Clear" step can be merged. A problem is clear if everyone would be able to describe an experiment or test case that shows if a solution fixes the problem.

285.3 Solutions Clear

> This step is recommended if it is not yet clear to the core developers which solution is the best.

Here you must ensure:

- problem, constraint and assumptions are well-explained and sound
- links from/to related decisions are created
- there are several solutions described, each with rationale and implication
- "Decision", "Rationale" and "Implications" are **not** yet filled out

Here the decision should not only have one decision but should describe several solutions. For each solution a proposal, rationale and optionally implications should be given.

> The solutions are clear if all reviewers understand the given solutions and no reviewer can come up with better solutions. The solution space is clear. I.e. the trade-offs, combinations and pros/cons of the solutions are explored. Decision author and reviewers are satisfied that every useful solution is present in the decision.

285.4 In Review

Now it is allowed to have the decision from the previous round in the "Decision" section. In this step, the last details of the chosen decision get polished:

- consistency with other decisions
- links from/to related decisions have been checked
- "Decision", "Rationale" and "Implications" are fully filled out

> Without merges in between, this is the last step reachable for a decision PR that started in "Drafts".

285.5 Decided

This step is mandatory. I.e., there must be a dedicated decision PR that puts the decision into "↩ Decided".

- "Decision", "Rationale" and "Implications" are now filled out and fixed according to the reviews
- decisions of this step usually already have an implementation PR

Decisions that need an update, e.g. because assumptions changed, sometimes directly start with the step "↩ Decided".

In this step, decision PRs only modify a *single* decision. Only exceptions like backlinks from other decisions are allowed.

285.6 Partially Implemented

We want to avoid this step, ideally PRs fully implement decisions.
Nevertheless, this can be useful for decisions that need to be done for every module like plugin or library. It is for decisions where only a few not-so-important modules are missing and/or issues exist for the remaining pieces.
The "Implication" must clearly say how much of the decision is already implemented.

285.7 Implemented

This step is mandatory. I.e., there must be a decision PR that puts the decision into "Implemented".

- Here the details of the decisions are stripped from the decision and moved to the documentation.
- The documentation links to the decision.
- The decision links to the new documentation.

In this step, decision PRs only modify a *single* decision. Here more exceptions are allowed, in particular documentation updates are okay.

285.8 Postponed

This step is:

- for decisions that would be useful contributions but there is currently nobody with the time to focus on the problem.
- a graveyard for neglected decisions, e.g., where the decision authors focus on something more important instead.

The purpose of this step is to have a clean "Drafts" folder.

285.9 Rejected

Alternatively, decisions might be rejected (e.g. status quo wins). This step is for decisions for which a consensus exists that the decision should not be taken right now. If circumstances change, the decision may be revisited.

The purpose of this final step is to have a clean "Drafts" folder.

Chapter 286

TEMPLATE

286.1 Problem

286.2 Constraints

1. 2. 3.

286.3 Assumptions

1. 2. 3.

286.4 Solutions

286.4.1 Alternative A

286.4.2 Alternative B

286.4.3 Alternative C

286.5 Decision

286.6 Rationale

286.7 Implications

-
-
-

286.8 Related Decisions

- [](doc_decisions__README_md)
- [](doc_decisions__README_md)
- [](doc_decisions__README_md)

286.9 Notes

Chapter 287

DESIGN

This document describes the design of Elektra's C-API and provides hints for binding writers. It is not aimed at plugin writers, since it does not talk about the implementation details of Elektra.
Elektra aims to fulfill the following design principles:

1. To make the API future-proof so that it can remain compatible and stable over a long period of time,
2. to make it hard to use the API the wrong way by making it simple & robust, and
3. to make the API easy to use for programmers reading and writing configuration.

The C-API is suitable to be reimplemented, also in non-C-languages, like Rust. Elektra provides a full-blown architecture to support configuring systems, and the C-API is the core of this endeavour.

287.1 Data Structures

The `Key`, `KeySet` and `KDB` data structures are defined in `kdbprivate.h` to allow ABI compatibility. This means, it is not possible to put one of Elektra's data structures on the stack. You must use the memory management facilities mentioned in the next section.

287.2 Memory Management

Elektra provides functions that create and free data. For example after you call:

```
KDB * kdbOpen();
```

you need to use:

```
int kdbClose(KDB *handle);
```

to get rid of the resources again. The second function may also shut down connections. Therefore, it must be called before the end of a program.

```
Key *keyNew(const char *keyName, ...);
int keyDel(Key *key);
KeySet *ksNew(int alloc, ...);
int ksDel(KeySet *ks);
```

In the above pairs, the first function reserves the necessary amount of memory. The second function frees the allocated data segment. There are more allocations happening, but they are invisible to the user of the API and happen implicitly within any of these 3 classes: `KDB`, `Key` and `KeySet`.
Key names and values cannot be handled as easy without helper libraries, because Elektra does not provide a string library. The function

```
const void *keyValue(const Key *key);
```

returns a value. You are not allowed to change the returned value. The life time is bound to the `Key`. The function

```
ssize_t keyValueSize(const Key *key);
```

gives the length of the value in bytes.

287.3 Variable Arguments

The constructors for `Key` and `KeySet` take a variable sized list of arguments. They can be used as an alternatives to the various `keySet*` methods and `ksAppendKey`. With them you are able to generate any `Key` or `KeySet` with a single C-statement. This can be done programmatically by the plugin `c`.

To just retrieve a key, use
```
Key *k = keyNew("/", KEY_END);
```
To obtain a `keyset`, use
```
KeySet *k = ksNew(0, KS_END);
```
Alternatively pass a list as described in the documentation. The idea of these variable arguments is, that one function call can create any `KeySet`. For binding writers `keyVNew` might be useful.

287.4 Off-by-one

We avoid off-by-one errors by starting all indices with 0, as usual in C. The size returned by the `*GetSize` functions (`keyGetValueSize`, `keyGetCommentSize` and `keyGetOwnerSize`) is exactly the size you need to allocate. So if you add 1 to it, too much space is allocated, but no error will occur.
The same is true for `elektraStrLen` which also already has the null byte included.

287.5 Minimal Set

`kdb.h` contains a minimal set of functions to fully work with a key database. The functions are implemented in `src/libs/elektra` in ANSI C.
Useful extensions are available in further libraries.

287.6 Return Values

Elektra's function share common error codes. Every function must return `-1` on error, if its return type is integer (like `int`, `ssize_t`). If the function returns a pointer, `0` (`NULL`) will indicate an error.
Elektra uses integers for the length of C strings, reference counting, `KeySet` length and internal `KeySet` allocations.
The interface always accepts `ssize_t` and internally uses `size_t`, which is able to store larger numbers than `ssize_t`.
The real size of C strings and buffers is limited to `SSIZE_MAX`. When a string exceeds that limit `-1` or a `NULL` pointer (see above) must be returned.
The following functions return an internal string:
```
const char *keyName(const Key *key);
const char *keyBaseName(const Key *key);
```
and in the case that `keyIsBinary(key)==0`:
```
const void *keyValue(const Key *key);
```
does so, too. If in any of the functions above `key` is a `NULL` pointer, then they also return `NULL`.
If there is no string you will get back `""`, that is a pointer to the value "\0'. `The function to determine the size will return1` in that case. That means that an empty string – nothing except the NULL terminator – has size `1`.
This is not true for `keyValue` in the case of binary data, because the value "\0'`in the first byte is perfectly legal binary data.` keyGetValueSize`may also return0` for that reason.

287.7 Error Handling

For `KDB` functions the user does not only get the return value but also a more elaborate error information, including an error message, in the metadata of the `parentKey` or `errorKey`. Furthermore, it is also possible to get warnings, even if the calls succeeded.
Using different error categories, the user of the API can have suitable reactions on specific error situations. Additional information about error handling is available here.
Elektra does not set `errno`. If a function you call sets `errno`, make sure to set it back to the old value again.

287.8 Naming

All function names begin with their class name, e.g. `kdb`, `ks` or `key`. We use capital letters to separate single words (CamelCase). This leads to short names, but might be not as readable as separating names by other means.

Get and *Set* are used for getters/and setters. We use *Is* to ask about a flag or state and *Needs* to ask about state related to databases. For allocation/deallocation we use C++ styled names (e.g `*New`, `*Del`).
Macros and Enums are written in capital letters. Flags start with `KDB_`, namespaces with `KEY_NS_` and macros with `ELEKTRA_`.
Data structures start with a capital letter for every part of the word:

- `KDB` ... Key Data Base Handle
- `KeySet` ... Key Set
- `Key` ... Key

We use singular for all names.
Function names not belonging to one of the three classes use the prefix `elektra*`.

287.9 const

Wherever possible functions should use the keyword `const` for parameters. The API uses this keyword for parameters, to show that a function does not modify a `Key` or a `KeySet`, e.g.:

```
const char *keyName(const Key *key);
const char *keyBaseName(const Key *key);
const void *keyValue(const Key *key);
const char *keyString(const Key *key);
const Key  *keyGetMeta(const Key *key, const char* metaName);
```

The reason behind this is, that the above functions – as their name suggest – only retrieve values. The returned value must not be modified directly.

287.10 Design Guidelines Checklist

On potential changes of the API/ABI as detected by the `build server`, please make sure the API has been reviewed according to the following 2 checklists:

287.11 Checklist for overall API

287.11.1 Consistency

- [ ] Consistent naming schemes for enums, macros, typedefs and functions
- [ ] Same things are named the same and included in Glossary
- [ ] Different things are named differently
- [ ] The order of arguments should be consistent across similar functions

287.11.2 Structural Clarity

- [ ] Functions with similar functionality have the same prefix

287.11.3 Compatibility

- [ ] All bindings have been updated to reflect the new API and work properly

287.11.4 Extensibility

- [ ] New API is easily extensible with additional functionality
- [ ] Components only depend on each other if needed

287.12 Checklist for each function

There are several checklists for functions, depending on the language in which the function is written:

- C
- Rust
- Java

Chapter 288

Algorithm

You might want to read about architecture and data structures first.

288.1 Outdated

> **Warning** Many of the things described below (especially about `KDB` and the `kdb*` functions) are outdated. See `kdb-operations.md` and `kdb-contracts.md` for up-to-date information.

288.2 Introduction

In this section, we will explain the heart of Elektra. `kdbOpen()` is responsible for the setup and the construction of the data structures needed later. `kdbGet()` does, together with the plugins, all actions necessary to read in the configuration. `kdbSet()` orchestrates the plugins to write out the configuration correctly. `kdbClose()` finally frees all previously allocated data structures.

288.2.1 kdbOpen

`kdbOpen()` retrieves the *mount point configuration* with `kdbGet()` using the *default backend*. During this process, the function sets up the data structures which are needed for later invocations of `kdbGet()` or `kdbSet()`. All backends are opened and mounted in the appropriate parts of the key hierarchy. The resulting backends are added both to the `Split` and the `Trie` object. `kdbOpen()` finally returns a `KDB` object that contains all this information.
The reading of the mount point configuration and the consequential self configuring of the system is called *bootstrapping*. Elektra builds itself up with a default backend (consisting of `libelektra-plugin-resolver` and `libelektra-plugin-storage`). Read more about bootstrapping here
`kdbOpen()` creates a `Split` object. It adds all backend handles and `parentKeys` during bootstrapping. So the buildup of the `Split` object takes place once. The resulting object is then used for both `kdbGet()` and `kdbSet()`. This approach is much better testable because the `Split` object is first initialised using the mount point configuration – separated from the filtering of the backends for every specific `kdbGet()` and `kdbSet()` request.
Afterwards the key hierarchy is static. Every application using Elektra will build up the same key database. Application-specific mount points are prohibited because changes of mount points would destroy the global key database. Elektra could not guarantee that every application retrieves the same configuration with the same key names any longer.
In `kdbOpen()`, nearly no checks are done regarding the expected behavior of the backend. The contract checker guarantees that only appropriate mount points are written into the mount point configuration. `kdbOpen()` checks only if the opening of plugin was successful. If not, the backend enclosing the plugin is not mounted at all.

288.2.2 Removing Keys

In Elektra version 0.6, removing keys was an explicit request. Only a single `Key` object could be removed from the database. For configuration files this method is inapplicable. For `filesys`, however, it was easy to implement.

In Elektra version 0.7, the behavior changed. Removing keys was integrated into `kdbSet()`. The user tagged keys that should be removed. After the next `kdbSet()`, these keys were removed from the key database. On the one hand, backends writing configuration files simply ignored the keys marked for removal. On the other hand, `filesys` needed that information to remove the files. To make this approach work for `filesys`, the marked keys were located at the very end of the `KeySet` and sorted in reverse. With this trick, recursive removing worked well. But this approach had major defects in the usage of `KeySet`. Because marking a key to be removed changed the sort order of the key set `ksLookupByName()` did not find this key anymore.
So in the present version removing keys is consistent again. A `KeySet` describes the current configuration. The user can reduce the `KeySet` object by *popping* keys out. The `kdbSet()` function applies exactly this configuration as specified by the key set to the key database. Contrary to the previous versions, the popped keys of the key set will be permanently removed.
The new circumstance yields **idempotent** properties for `kdbSet()`. The same `KeySet` can be applied multiple times, but after the first time, the key database will not be changed anymore. Note that `kdbSet()` actually detects that there are no changes and will do nothing. To actually show the idempotent behavior the KeySet has to be regenerated or the key database needs to be reopened.
It is, however, not known if keys should be removed permanently only by investigating the `KeySet`. But only if this knowledge is present, the core can decide if the key set needs to be written out or if the configuration is unchanged. So we decided to track how many keys are delivered in `kdbGet()`. If the size of the `KeySet` is lower than this number determined at the previous `kdbGet()`, Elektra's core knows that some keys were popped. Hence, the next `kdbSet()` invocation needs to change the concerned key database.
The situation is now much clearer. The semantics of popping a key will result in removing the key from the key database. And the intuitive idea that a `KeySet` will be applied to the key database is correct again.

288.3 kdbGet

It is critical for application startup-time to retrieve the configuration as fast as possible. Hence, the design goal of the `kdbGet()` algorithm is to be efficient while still enabling plugins to have relaxed postconditions. To achieve this, the sequence of *syscalls* must be optimal. On the other hand, it is not tolerable to waste time or memory inside Elektra's core, especially during an initial request or when no update is available.
The synopsis of the function is:

```
int kdbGet(KDB *handle, KeySet *returned, Key * parentKey);
```

The user passes a key set, called `returned`. If the user invokes `kdbGet()` the first time, he or she will usually pass an empty key set. If the user wants to update the application's settings, `returned` will typically contain the configuration of the previous `kdbGet()` request. The `parentKey` holds the information below which key the configuration should be retrieved. The `handle` contains the data structures needed for the algorithm, like the `Split` and the `Trie` objects.
`kdbGet()` does a rather easy job, because `kdbSet()` already guarantees that only well formatted, non-corrupted and well-typed configuration is written out in the key database. The task is to query all backends in question for their configuration and then merge everything.

288.3.1 Responsibility

A backend may yield keys that it is not responsible for. It is not possible for a backend to know that another backend has been mounted below and the other backend is now responsible for some of the keys that are still in the storage. Additionally, plugins are not able to determine if they are responsible for a key or not. Consequently, it can happen that more than one backend delivers a key with the same name.
`kdbGet()` ensures that a key is uniquely identified by its name. Elektra's core will pop keys that are outside of the backend's responsibility. Hence, these keys will not be passed to the user and we get the desired behavior: The nearest mounted backend to the key is responsible.
For example, a generator plugin in the backend (A) always emits following keys. (A) and (B) indicate from which backend the key comes from.

```
user:/sw/generator/akey (A)
user:/sw/generator/dir (A)
user:/sw/generator/dir/outside1 (A)
user:/sw/generator/dir/outside2 (A)
```

It will still return these keys even if the plugin is not responsible for some of them anymore. This can happen if another backend B is mounted to `user:/sw/generator/dir`. In the example it yields the following keys:

```
user:/sw/generator/dir (B)
user:/sw/generator/dir/new (B)
user:/sw/generator/dir/outside1 (B)
user:/sw/generator/outside (B)
```

In this situation kdbGet() is responsible to pop all three keys at, and below, user:/sw/generator/dir of backend (A) and the key user:/sw/generator/outside of backend (B). The user will get the resulting key set:

```
user:/sw/generator/akey (A)
user:/sw/generator/dir (B)
user:/sw/generator/dir/new (B)
user:/sw/generator/dir/outside1 (B)
```

Note that the key exactly at the mount point comes from the backend mounted at user:/sw/generator/dir.

288.3.2 Sequence

kdbOpen() already creates a Split object for the whole configuration tree. In this object, kdbOpen() will append a list of all backends available. A specific kdbGet() request usually includes only a part of the configuration. For example, the user is only interested in keys below user:/sw/apps/userapp. All backends that cannot contribute to configuration below user:/sw/apps/userapp will be omitted for that request. To achieve this, parts of the Split object are filtered out. After this step we know the list of backends involved. The Split object allocates a key set for each of these backends.
Afterwards the first plugin of each backend is called to determine if an update is needed. If no update is needed, the algorithm has finished and returns zero.
Now we know which backends do not need an update. For these backends, the previous configuration from returned is appointed from to the key sets of the Split object. The algorithm will not set the *syncbits* of the Split object for these backends because the storage of the backends already contains up-to-date configuration.
The other backends will be requested to *retrieve* their configuration. The initial empty KeySet from the Split object and the relevant file name in the key value of parentKey are passed to each remaining plugin. The plugins extend, validate and process the key set. When an error has occurred, the algorithm can stop immediately because the user's KeySet returned is not changed at this point. When this part finishes, the Split object contains the whole requested configuration separated in various key sets.
Subsequently the freshly received keys need some *post-processing*:

- Newly allocated keys in Elektra always have the *sync flag* set. Because the plugins allocate and modify keys with the same functions as the user, the returned keys will also have their sync flag set. But from the user's point of view the configuration is unmodified. So some code needs to remove this sync flag. To relax the post conditions of the plugins, kdbGet() removes it.
- To detect removed keys in subsequent kdbSet() calls, kdbGet() needs to store the number of received keys of each backend.
- Additionally, for every key it is checked if it belongs to this backend. This makes sure that every key comes from a single source only as designated by the Trie. In this process, Elektra pops all duplicated and overlapping keys in favor of the responsible backend.

The last step is to *merge* all these key sets together. This step changes the configuration visible to the user. After some cleanup the algorithm finally finishes.

288.3.3 Updating Configuration

The user can call kdbGet() often even if the configuration or parts of it are already up-to-date. This can happen when applications reread configuration in some events. Examples are signals (SIGHUP is the signal used for that on Unix systems. It is sent when the program's controlling terminal is closed. Daemons do not have a terminal so the signal is reused for reloading configuration.), notifications, user requests and in the worst case periodical attempts to reread configuration.
The given goal is to keep the sequence of needed syscalls low. If no update is needed, it is sufficient to request the timestamp (On POSIX systems using stat()) of every file. No other syscall is needed. Elektra's core alone cannot check that because getting a timestamp is not defined within the standard C99. So instead the resolver plugin handles this problem. The resolver plugin returns 0 if nothing has changed.
This decision yields some advantages. Both the storage plugins and Elektra's core can conform to C99. Because the resolver plugin is the very first in the chain of plugins, it is guaranteed that no useless work is done.

288.3.4 Initial kdbGet Problem

Because Elektra provides self-contained configuration, kdbOpen() has to retrieve settings in the *bootstrapping* process below system:/elektra as explained in bootstrapping. Because of the new way to keep track

of removed keys, the internally executed kdbGet() creates a problem. Without countermeasures even the first kdbGet() of a user requesting the configuration below system:/elektra fails, because the resolver finds out that the configuration is already up-to-date. The configuration delivered by the user is empty at this point. As a result, the empty configuration will be appointed and returned to the user.
A simple way to resolve this issue is to reload the default backend after the internal configuration was fetched. Reloading resets the timestamps and kdbGet() works as expected.

288.4 kdbSet

Not performance, but robust and reliable behavior is the most important issue for kdbSet(). The design was chosen so that some additional in-memory comparisons are preferred to a suboptimal sequence of syscalls. The algorithm makes sure that keys are written out only if it is necessary, because applications can call kdbSet() with an unchanged KeySet. For the code to decide this, performance is important.

288.4.1 Properties

kdbSet() guarantees the following properties:

- Modifications to permanent storage are only made when the configuration was changed.
- When errors occur, every plugin gets a chance to rollback its changes as described in **exception safety**.
- If every plugin does this correctly, the whole KeySet is propagated to permanent storage. Otherwise nothing is changed in the key database. Plugins delivered with Elektra meet this requirement.

The synopsis of the function is:
```
int kdbSet(KDB *handle, KeySet *returned, Key * parentKey);
```
The user passes the configuration using the KeySet returned. The key set will not be changed by kdbSet(). The parentKey provides a way to limit which part of the configuration is written out. For example, the parentKey user:/sw/org/app/#0/current will induce kdbSet() to only modify the key databases below user:/sw/org/app even if the KeySet returned also contains more configuration. Note that all backends with no keys in returned but that are below parentKey will completely wipe out their key database. The KDB handle contains the necessary data structures.

288.4.2 Search for Changes

As a first step, kdbSet() *divides* the configuration passed in by the user to the key sets in the Split object. kdbSet() searches for every key if the *sync flag* is checked. Then kdbSet() decides if a key was removed from a backend by comparing the actual size of the key set with the size stored from the last kdbGet() call. We see that it is necessary to call kdbGet() first before invocations of kdbSet() are allowed.
We know that data of a backend has to be written out if at least one key was changed or removed. If no backend has any changes, the algorithm will terminate at this point. The careful reader notices that the process involves no file operations.

288.4.3 Duplicated Key Sets

If some backends need synchronization, the algorithm continues by filtering out all backends in the Split object that do not have changes. At this point, the Split object has a list of backends with their respective key sets.
Plugins in kdbSet() can change values. Other than in kdbGet(), the user is not interested in these changes. Instead, the values are transformed to be suitable for the storage. To make sure that the changed values are not passed to the user, the algorithm continues with a *deep duplication* of all key sets in the Split object.

288.4.4 Resolver

All plugins of each included backend are executed one by one up to the resolver plugin. If this succeeds, the resolver plugin is responsible for committing these changes. After the successful commit, *error codes* of plugins are ignored. Only logging and notification plugins are affected.

288.4.5 Atomic Replacement

Up to now only file-based storages with atomic properties were developed. The replacement of a file with another file that has not yet been written is not trivial. The straightforward way is to lock a file and start writing to it. But this approach can result in broken or partially finished files in events like “out of disc space”, signals or other asynchronous aborts of the program.
A temporary file solves most of this problem, because in problematic events the original file stays untouched. When the temporary file is written out properly, it is renamed and the original configuration file is overwritten. But another concurrent invocation of `kdbSet()` can try to do the same with the result that one of the newly written files is lost. To avoid this problem, locks are needed and protect cooperating processes (such as other processes using Elektra). Additionally modification time is used to detect if a file was modified. Unfortunately the modification time on some file systems has a resolution of one second. So any changes within that time slot might not be recognized.

288.4.6 Errors

The plugins within `kdbSet()` can fail for a variety of reasons. Conflicts occur most frequently. A conflict means that during executions of `kdbGet()` and `kdbSet()` another program has changed the key database. In order not to lose any data, `kdbSet()` fails without doing anything. In conflict situations Elektra leaves the programmer no choice. The programmer has to retrieve the configuration using `kdbGet()` again to be up-to-date with the key database. Afterwards it is up to the application to decide which configuration to use. In this situation it is the best to ask the user, by showing him the description and reason of the error, how to continue:

1. Save the configuration again. The changes of the other program will be lost in this case.
2. The key database can also be left unchanged as the other program wrote it. After using `kdbGet()` the application is already up-to-date with the new configuration. All configuration changes the user made before will be lost.
3. The application can try to merge the key sets to get the best result. If no key is changed on both sides the result is clear, otherwise the application has to decide if the own or the other configuration should be favored. The result of the merged key sets has to be written out with `kdbSet()`.
4. Merging the key sets can be done with `ksAppend()`. The source parameter is the preferred configuration. Note that the downside of the third option is that the merged configuration might not be valid.

Sometimes a concrete key causes the problem that the whole key set cannot be stored. That can happen on validation or because of type errors. Such errors are usually caused by a mistake made by the user. So the user is responsible for changing the settings to make it valid again. In such situations, the *internal cursor* of the `KeySet returned` will point to the problematic key.
A completely different approach is to export the configuration when `kdbSet()` returned an error code. The user can then edit, change or merge this configuration with more powerful tools. Finally, the user can import the configuration into the global key database. The export and import mechanism is called "streaming" and will be explained in *streaming*.

Chapter 289

Architecture

In this document we start to explain the implementation of Elektra. There are several follow-up documents which explain all details of:

- error handling,
- data structures, and
- finally the core algorithm.

We discuss problems and the solution space so that the reader can understand the rationale of how problems were solved.
To help readers to understand the algorithm that glues together the plugins, we first describe some details of the data structures. Full knowledge of the algorithm is not presumed to be able to develop most plugins.
Further important concepts are explained in:

- bootstrapping
- granularity

289.1 Outdated

> **Warning** Many of the things described below (especially in relation to backends and mountpoints) are outdated. See `kdb-operations.md`, `backend-plugins.md` and `mountpoints.md` for more up-to-date information.

289.2 API

The aim of the Elektra Initiative is to design and implement a powerful API for configuration. When the project started, we assumed that this goal was easy to achieve, but dealing with the semantics turned out to be a difficult problem. For the implementation, an ambitious solution is required because of the necessary modularity to implement flexible backends as introduced in Elektra. But also the design of a good API has proved to be much more difficult than expected.

289.2.1 Changes in the APIs

From Elektra 0.7 to Elektra 0.8, we changed the API of Elektra as little as possible. It should be mentioned that `KeySet` is now always sorted by name. The function `ksSort()` is now depreciated and was removed. The handling of removed keys was modified. Additionally, the API for metadata has fundamentally changed, but the old interface still works. These changes will be described in implementation of metadata. However, the implementation of Elektra changed radically as discussed in algorithm.

289.2.2 API Design

API Design presents a critical craft every programmer should be aware of. We will shortly present some of the main design issues that matter and show how Elektra has solved them.
A design goal is to detect errors early. As easy as it sounds, as difficult it is to actually achieve this goal. Elektra tries to avoid the problem by checking data being inserted into `Key` and `KeySet`. Elektra catches many errors like invalid key names soon. Elektra allows plugins to check the configuration before it is written into the key database so that problematic values are never stored.
"Hard to use it wrong" tends to be a more important design objective than "Easy to use it right". Searching for a stupid bug costs more time than falling into some standard traps which are explained in documentation. In Elektra, the data structures are robust and some efforts were taken to make misuse unlikely.
Another fundamental principle is that the API must hide implementation details and should not be optimized towards speed. In Elektra, the actual process of making configuration permanent is completely hidden.
"Off-by-one confusion" is a topic of its own. The best is to stick to the conventions the programming language gives. For returning sizes of strings, it must be clear whether a terminating ‘'\0'` is included or not. All such decisions must be consistent. In Elektra the terminating null is always included in the size.
The interface must be as small as possible to tackle problems addressed by the library. Internal and external APIs must be separated. Internal APIs in libraries shall be declared as `static` to prevent its export. In Elektra, internal names start with `elektra` opposed to the external names starting with `key`, `ks` or `kdb`.
Elektra always passes user context pointers, but never passes or receives a full data structure by value. It is impossible to be ABI compatible otherwise. Elektra is restrictive in what it returns (strong postconditions), but as liberal as possible for what comes in (preconditions are avoided where possible). In Elektra even null pointers are accepted for any argument.
"Free everything you allocate" is a difficult topic in some cases. If Elektra cannot free space or other resources after every call, it provides a `close()` function. Everything will be freed. The tool **Valgrind** with **Memcheck** helps us locate problems. The whole test suite runs without any memory problems. The user is responsible for deleting all created `Key` and `KeySet` objects and closing the `KDB` handle.
As a final statement, we note that the Unix philosophy should always be considered: "Do only one thing, but do it in the best way. Write it that way that programs work together well."

289.3 Modules

Elektra's core can be compiled with a C compiler conforming to the `ISO/IEC 9899:1999 standard`, called C99 henceforth. Functions not conforming to C99 are considered to be not portable enough for Elektra and are separated into plugins. But there is one notable exception: it must be `libelektra-kdb`'s task to load plugins. Unfortunately, C99 does not know anything about modules. **POSIX** (Portable Operating System Interface) provides `dlopen()`, but other operating systems have dissimilar APIs for that purpose. They sometimes behave differently, use other names for the libraries and have incompatible error reporting systems. Because of these requirements Elektra provides a small internal API to load such modules independently from the operating system. This API also hides the fact that modules must be loaded dynamically if they are not available statically.
Plugins are usually realized with modules. Modules and libraries are technically the same in most systems. (One exception is macOS.) After the module is loaded, the special function plugin factory is searched for. This function returns a new plugin. With the plugin factory the actual plugins are created.

289.3.1 Static Loading

For the static loading of modules, the modules must be built-in. With `dlopen(const char* file)` POSIX provides a solution to look up such symbols by passing a null pointer for the parameter `file`. Non-POSIX operating systems may not support this kind of static loading. Therefore, Elektra provides a C99 conforming solution for that problem: a data structure stores the pointers to the plugin factory of every plugin. The build system generates the source file of this data structure because it depends on built-in plugins.
Elektra distinguishes internally between modules and plugins. Several plugins can be created out of a single module. During the creation process of the plugin, dynamic information - like the configuration or the data handle - is added.

289.3.2 API

The API of **libloader** consists of the following functions:
Interface of Module System:

```
int elektraModulesInit (KeySet *modules, Key *error);
elektraPluginFactory elektraModulesLoad (KeySet *modules,
                const char *name, Key *error);
int elektraModulesClose (KeySet *modules, Key *error);
```

`elektraModulesInit()` initializes the module cache and calls necessary operating system facilities if needed. `elektraModulesLoad()` does the main work by either returning a pointer to the plugin factory from cache or loading it from the operating system. The plugin factory creates plugins that do not have references to the module anymore. `elektraModulesClose()` cleans up the cache and finalises all connections with the operating system.
Not every plugin is loaded by `libloader`. For example, the *version plugin*, which exports version information, is implemented internally.

289.4 Mount Point Configuration

`kdb mount` creates a **mount point configuration** as shown in the example below. `/hosts` is a unique name within the mount point configuration provided by the administrator. To escape the `/` character, the name is changed to `\/hosts` in the configuration.
Example for a mount point configuration:

```
system:/elektra/mountpoints/\/hosts
system:/elektra/mountpoints/\/hosts/config
system:/elektra/mountpoints/\/hosts/config/glob/set/#0
system:/elektra/mountpoints/\/hosts/config/glob/set/#1
system:/elektra/mountpoints/\/hosts/config/glob/set/#2
system:/elektra/mountpoints/\/hosts/config/glob/set/#3
system:/elektra/mountpoints/\/hosts/config/glob/set/#4
system:/elektra/mountpoints/\/hosts/config/glob/set/#4/flags
system:/elektra/mountpoints/\/hosts/config/mountpoint
system:/elektra/mountpoints/\/hosts/config/path
system:/elektra/mountpoints/\/hosts/error
system:/elektra/mountpoints/\/hosts/error/rollback
system:/elektra/mountpoints/\/hosts/error/rollback/#0
system:/elektra/mountpoints/\/hosts/error/rollback/#0/label (="resolver")
system:/elektra/mountpoints/\/hosts/error/rollback/#0/name (="resolver_fm_hpu_b")
system:/elektra/mountpoints/\/hosts/get
system:/elektra/mountpoints/\/hosts/get/postgetstorage
system:/elektra/mountpoints/\/hosts/get/postgetstorage/#0
system:/elektra/mountpoints/\/hosts/get/postgetstorage/#0/label (="glob")
system:/elektra/mountpoints/\/hosts/get/postgetstorage/#0/name (="glob")
system:/elektra/mountpoints/\/hosts/get/getresolver
system:/elektra/mountpoints/\/hosts/get/getresolver/#0
system:/elektra/mountpoints/\/hosts/get/getresolver/#0/reference (="resolver")
system:/elektra/mountpoints/\/hosts/get/getstorage
system:/elektra/mountpoints/\/hosts/get/getstorage/#0
system:/elektra/mountpoints/\/hosts/get/getstorage/#0/label (="hosts")
system:/elektra/mountpoints/\/hosts/get/getstorage/#0/name (="hosts")
system:/elektra/mountpoints/\/hosts/set
system:/elektra/mountpoints/\/hosts/set/commit
system:/elektra/mountpoints/\/hosts/set/commit/#0
system:/elektra/mountpoints/\/hosts/set/commit/#0/reference (="resolver")
system:/elektra/mountpoints/\/hosts/set/precommit
system:/elektra/mountpoints/\/hosts/set/precommit/#0
system:/elektra/mountpoints/\/hosts/set/precommit/#0/label (="sync")
system:/elektra/mountpoints/\/hosts/set/precommit/#0/name (="sync")
system:/elektra/mountpoints/\/hosts/set/presetstorage
system:/elektra/mountpoints/\/hosts/set/presetstorage/#0
system:/elektra/mountpoints/\/hosts/set/presetstorage/#0/reference (="glob")
system:/elektra/mountpoints/\/hosts/set/presetstorage/#1
system:/elektra/mountpoints/\/hosts/set/presetstorage/#1/label (="error")
system:/elektra/mountpoints/\/hosts/set/presetstorage/#1/name (="error")
system:/elektra/mountpoints/\/hosts/set/presetstorage/#2
system:/elektra/mountpoints/\/hosts/set/presetstorage/#2/label (="network")
system:/elektra/mountpoints/\/hosts/set/presetstorage/#2/name (="network")
system:/elektra/mountpoints/\/hosts/set/setresolver
system:/elektra/mountpoints/\/hosts/set/setresolver/#0
system:/elektra/mountpoints/\/hosts/set/setresolver/#0/reference (="resolver")
system:/elektra/mountpoints/\/hosts/set/setstorage
system:/elektra/mountpoints/\/hosts/set/setstorage/#0
system:/elektra/mountpoints/\/hosts/set/setstorage/#0/reference (="hosts")
```

Let us look at the subkeys below the key `system:/elektra/mountpoints/\/hosts`:

- **config**: Everything below `config` is the system's configuration of the backend. Every plugin within the backend will find this configuration directly below `system` in its **plugin configuration**. For example,

```
system:/elektra/mountpoints/\/hosts/config/glob/set/#4/flags
```

will be translated to

```
system:/glob/set/#4/flags
```

and inserted into the plugin configuration for all plugins in the `/hosts` backend.

It is the place where configuration can be provided for every plugin of a backend. The contract checker deduces this configuration to satisfy the contract for a plugin.
For example, `/hosts` recommends the use of the `glob` plugin. In this case, it is included. To work properly, the `glob` plugin needs to receive certain configuration, which is provided by `/hosts`:

```
system:/elektra/mountpoints/\/hosts/config/glob/set/#0
system:/elektra/mountpoints/\/hosts/config/glob/set/#1
system:/elektra/mountpoints/\/hosts/config/glob/set/#2
system:/elektra/mountpoints/\/hosts/config/glob/set/#3
system:/elektra/mountpoints/\/hosts/config/glob/set/#4
system:/elektra/mountpoints/\/hosts/config/glob/set/#4/flags
```

The **contract checker** writes out the configuration looking like the one in this example.

- **config/path**: is a common setting needed by the resolver plugin. It is the relative path to a filename that is used by this backend. On Unix systems, the resolver would determine the name `/etc/hosts` for system configuration.
- **config/mountpoint**: is a key that represents the mount point. Its value is the location where the backend is mounted. If a mount point has an entry for both the user and the system hierarchy, it is called **cascading mount point**. A cascading mount point differs from two separate mount points because internally only one backend is created. In the example, the mount point `/hosts` means that the backend handles both `user:/hosts` and `system:/hosts`. If the mount point is `/`, the backend will be mounted to `dir:/`, `user:/` and `system:/`.
- **error**: presents a list of all plugins to be executed in the error case of `kdbSet()` which will be explained in **error situation**.
- **get**: is a list of all plugins used when reading the configuration from the key database. They are executed in `kdbGet()`.
- **set**: contains a list of all plugins used when storing configuration. They are executed in `kdbSet()`.

Each of the plugins inside the three lists may itself have the subkey `config`. The configuration below this subkey provides plugin specific configuration. This configuration appears in the user's configuration of the plugin. Configuration is renamed properly. For a fictional backend named `backendname` and a plugin named `pluginname` containing configuration named `property`, the key

```
system:/elektra/mountpoints/backendname/set/storage/config/pluginname/property
```

is transformed to

```
user:/pluginname/property
```

and appears in the plugin configuration of the `pluginname` plugin inside the `backendname` backend.

289.4.1 Roles and Placements

Expressions after `get`, `set` or `error` such as `poststorage`, `storage` or `resolver` describe the role that the plugin fulfills. For example, the key

```
system:/elektra/mountpoints/\/hosts/set/precommit/#0/name
```

belongs to a plugin fulfilling the `precommit` role.
The cypher `#0` describes the placement of the plugin in relation to potential other plugins fulfilling the same role. More on roles and placements here.

289.4.2 Referencing

The same plugin often must occur in more than one place within a backend. The most common use case is a plugin that has to be executed for both `kdbGet()` and `kdbSet()`. It must be the same plugin if it preserves state between the executions.
Other plugins additionally have to handle error or success situations. One example of exceptional intensive use is the resolver plugin. It is executed twice in `kdbSet()`. In `kdbGet()` it is also used as shown above. Plugins can be defined in the following ways:

- **Single use**:

  Assuming that the following key contains the value `"pluginname"`,

```
system:/elektra/mountpoints/backendname/get/poststorage/#0/name
```

introduces a new plugin from the module `pluginname` which cannot be referenced later.

- **Labelling**:

  Assuming that the key ending with `label` contains the value `"pluginlabel"` and the key ending with `name` stays the same,

```
system:/elektra/mountpoints/backendname/get/poststorage/#0/label
system:/elektra/mountpoints/backendname/get/poststorage/#0/name
```

also introduces a new plugin from the module `pluginname` and gives it the name `pluginlabel`. By using the introduced label, the plugin can be used later.

- **Referencing**:

  Assuming that the following key contains the value `"pluginlabel"`,

```
system:/elektra/mountpoints/backendname/get/poststorage/#0/reference
```

references back to the label which was introduced before. That way, the same plugin which was created previously is used.
`kdb mount` implements the generation of these names as described above.

289.4.3 Changing Mount Point Configuration

When the user changes the mount point configuration, without countermeasures, applications already started will continue to run with the old configuration. This could lead to a problem if backends in use are changed or removed. It is necessary to restart all such programs. Notification is the best way to deal with the situation. Changes of the mount point configuration, however, do not occur often. For some systems, the manual restart may also be appropriate.
In this situation, applications can receive warning or error information if the configuration files are moved or removed. The most adverse situation occurs if the sequence of locking multiple files produces a *dead lock*. Under normal circumstances, the sequence of locking the files is deterministic, so either all locks can be requested or another program will be served first. But several programs with different mountpoint configurations running at the same time can cause a disaster. The problem gets even worse, because `kdb mount` is unable to detect such situations. Every specific mount point configuration for itself is trouble-free.
But still a dead lock can arise when multiple programs run with different mountpoint configurations. Suppose we have a program `A` which uses the backends `B1` and `B2` that requests locks for the files `F1` and `F2`. Then the mount point configuration is changed. The user removes `B1` and introduces `B3`. `B3` is in a different path mounted after `B2`, but also accesses the same file `F1`. The program `B` starts after the mount point configuration is changed. So it uses the backends `B2` and `B3`. If the scheduler decides that first `A` and then `B` both successfully lock the files `F1` and `F2`, a dead lock situation happens because in the afterwards the applications `A` and `B` try to lock `F2` and `F1`.
A manual solution for this problem is to enable `kdb` to output a list of processes that still use old mount point configuration. The administrator can restart these processes. The preferred solution is to use notification for mount point configuration changes or simply to use a lock-free resolver.
Continue reading with the data structures.

Chapter 290

Backend Plugins

290.1 Backend Contract

There exists a *backend contract* between `libelektra-kdb` and any plugin acting as a backend plugin. This contract describes:

- the order of the phases (see kdb-operations.md)
- the interactions between a backend plugin and `libelektra-kdb`.

```
sequenceDiagram
    actor user
    participant kdb as libelektra-kdb
    participant backend as backend
    participant other as other-backend
    participant storage
    participant validation
    participant sa as storage-unit-a (e.g. file)
    participant sb as storage-unit-b (e.g. database)
    user-»+kdb: kdbGet
    kdb-»backend: init
    kdb-»other: init
    kdb-»+backend: resolver
    backend-»-kdb: "storage-unit-a", update needed
    kdb-»+other: resolver
    other-»-kdb: "storage-unit-b", no update needed
    kdb-»backend: prestorage
    kdb-»+backend: storage
    backend-»+storage: #
    storage-»+sa: #
    sa--»-storage: #
    storage--»-backend: #
    backend--»-kdb: #
    kdb-»+backend: poststorage
    backend-»+validation: #
    validation--»-backend: #
    backend--»-kdb: #
    kdb-»kdb: merge backends
    kdb--»-user: #
sequenceDiagram
    actor user
    participant kdb as libelektra-kdb
    participant backend as backend
    participant other as other-backend
    participant storage
    participant validation
    participant sa as storage-unit-a (e.g. file)
    user-»+kdb: kdbSet
    kdb-»kdb: check backends initialized
    kdb-»+backend: resolver
    backend--»-kdb: "storage-unit-a"
    critical try to store
      kdb-»+backend: prestorage
      backend-»+validation: #
      validation--»-backend: #
      backend--»-kdb: #
      kdb-»+backend: storage
      backend-»+storage: #
      storage-»+sa: write to temp
      sa--»-storage: #
      storage--»-backend: #
      backend--»-kdb: #
      kdb-»backend: poststorage
```

```
    kdb-»backend: precommit
    backend-»sa: make changes permanent
    kdb-»+backend: commit
    backend--›-kdb: #
    kdb-»backend: postcommit
  option on failure
    kdb-»backend: prerollback
    kdb-»+backend: rollback
    backend-»sa: revert changes
    backend--»-kdb: #
    kdb-»backend: postrollback
  end
  kdb--»-user: #
```

The diagrams above show some typical sequences of phases during a `get` and a `set` operation. For each of the phases of a `get` operation `libelektra-kdb` calls the backend plugin's `elektra<Plugin>Get` function once. Similarly, for the phases of a `set` operation `elektra<Plugin>Set` is called once. The backend plugin can also (optionally) delegate to other plugins.
The current phase is communicated to the backend plugin (and any other plugin) via the global keyset. It can be retrieved via the `elektraPluginGetPhase` function.

290.1.1 <tt>parentKey</tt>

The key `parentKey` that is given to the backend plugin as an input at various points, must be treated carefully. *All* modifications to this key will be propagated to the `parentKey` that was used to call `kdbGet`.
The name of the `parentKey` is marked read-only and therefore cannot be changed. The value and metadata can, and in some cases must be, changed. Importantly however, there are no guarantees that the metadata of `parentKey` can be changed arbitrarily.

290.1.2 Operation <tt>get</tt>

The `get` operation is mandatory and all backend plugins must implement it.

290.1.2.1 Initialization Phase

During the `init` phase the backend plugin is called with:

- A key `parentKey` whose name is the root of the mountpoint configuration (e.g. `system:/elektra/mountpoints/system:\/hosts`) and whose value is an empty string. The key name and value of this key are read-only. The name of `parentKey` is chosen to make it easier for the plugin to produce good error messages.
- A keyset `definition` containing the mountpoint definition. To make things easier for the plugin, keys in `definition` are renamed to be below `system:/`. For example, if the key `system:/elektra/mountpoints/system:\/hosts/path` is set in the KDB, then `definition` will contain a key `system:/path`.
- Additionally, the plugins for the current mountpoint are opened by `libelektra-kdb` and provided to the backend plugin via the global keyset. They can be accessed via the `elektraPluginFromMountpoint` function.

The backend plugin then:

- **MUST** parse the mountpoint definition and store the necessary information for later phases internally.
- **SHOULD** validate that the mountpoint definition *can be* valid.
- **SHOULD NOT** do other validation. For example a file-based backend, *should not* check whether the file(s) or path(s) referenced in the mountpoint definition exist. Such a check should be done in the `resolver` phase.
- **MAY** decide that the mountpoint should be read-only. If so, this must be indicated to `libelektra-kdb` via the return value.

This phase exists purely for the backend plugin to initialize and configure itself for the mountpoint.

> **Note**: This phase is only executed *once per instance of* `KDB`. Only the first `kdbGet()` call will result in `libelektra-kdb` executing this phase, all future calls to `kdbGet()` (and `kdbSet()`) start with the `resolver` phase. The backend plugin must store the necessary information contained in the mountpoint definition internally to accommodate this.

290.1.2.2 Resolver Phase

During the `resolver` phase the backend plugin is called with:

- A key `parentKey` whose name is the parent key of the mountpoint and whose value is an empty string. The key name of this key is read-only.
- An empty keyset `ks`.

The backend plugin then:

- **MUST** set the value of the `parentKey` to a value identifying the storage unit (the *storage identifier*) that contains the data of the mountpoint. For file-based backend plugins, this means setting the value of `parentKey` to an absolute filename.
- **MAY** set additional metadata on `parentKey`, if it is not suitable to encode the information required for the following phases as a single string.

  Note: The backend plugin may also modify the keyset `ks`, but `libelektra-kdb` will discard this keyset after this phase, so these modifications won't have any effects.

During a `set` operation, the backend plugin must ensure

- that errors in the storage phases can be safely rolled back and
- that the `set` operation does not affect concurrent operations. For file-based backends, this means creating a temporary storage file and returning its absolute filename instead of the name of the actual storage file. In other words, in a `set` operation the `resolver` phase is also about preparing a transaction in addition to resolving the storage unit.

290.1.2.3 Cache-Check Phase

Implementing this phase is optional. If a backend plugin does not support caching, it should immediately return a value indicating that the cache is invalid. If there is no cache entry for this backend, `libelektra-kdb` skips this phase.
During the `cachecheck` phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `resolver` phase of this `get` operation. The key name and value of this key are read-only. Additionally, the metakey `internal/kdb/cachehandle` is set to a value indicating the cache handle (usually modification time) of the cache entry.
- An empty keyset `ks`.

The backend plugin then:

- **MUST** indicate via the return value, whether the cache entry for this backend is still valid.

  Note: The backend plugin may also modify the keyset `ks`, but `libelektra-kdb` will discard this keyset after this phase, so these modifications won't have any effects.

290.1.2.4 Storage Phases

These phases are responsible for reading and validating the actual data stored in the KDB.
In the `prestorage` phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `resolver` phase of this `get` operation. The key name and value of this key are read-only.
- An empty keyset `ks`.

There are no restrictions on what the backend plugin may do in this phase, but just like in the `resolver` phase, change to `ks` will be discarded. This phase is useful for file-level manipulations, like file-based encryption, line ending conversion or verifying file signatures. In this sense, it is the counter-part of the `precommit` phase of the `set` operation.
The `storage` phase is for reading the actual data. In this phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `prestorage` phase of this `get` operation. The key name and value of this key are read-only.
- An empty keyset `ks`.

The backend plugin then:

- **MUST** read the data for the mountpoint from the storage unit that was selected in the `resolver` phase.
- **MUST** parse that data and insert it below `parentKey` into `ks`.

The last of the storage phases is the `poststorage` phase. In this phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `storage` phase of this `get` operation. The key name and value of this key are read-only.
- The exact `ks` that was returned by the `storage` phase of this `get` operation.

Again there are no restrictions on what the backend plugin may do in this phase. However, unlike the `prestorage` phase, this phase is a very important one. It is where validation, generation of implicit values and similar tasks happen.
Finally, `libelektra-kdb` merges the keyset returned by the `poststorage` phase with the ones returned by other backend plugins for different mountpoints and then returns it to the user.

290.1.3 Operation <tt>set</tt>

The `set` operation is optional. A mountpoint is automatically read-only and doesn't support the `set` operation, if the backend plugin does not define a `elektra<Plugin>Set` function.
Alternatively, the read-only nature of the mountpoint may also be indicated by the backend plugin during the `init` phase of the `get` operation.

290.1.3.1 Resolver Phase

During the `resolver` phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `resolver` phase of the last `get` or `set` operation. The key name of this key is read-only.
- An empty keyset `ks`.

The backend plugin then:

- **MUST** set the value of the `parentKey` to a value identifying the storage unit that contains the data of the mountpoint. For file-based backend plugins, this means setting the value of `parentKey` to an absolute filename.
- **MAY** set metadata on `parentKey`, if it is not suitable to encode the information required for the following phases as a single string.
- **MUST** check whether the data was changed since the last `get` operation. The result of this check is given to `libelektra-kdb` via the return value of the `get` function.
- **MUST** ensure that errors in the storage phases can be safely rolled back and that the `set` operation does not affect concurrent operations. For file-based backends, this means creating a temporary storage file and returning its absolute filename instead of the name of the actual storage file.

  Note: The backend plugin may also modify the keyset `ks`, but `libelektra-kdb` will discard this keyset after this phase, so these modifications won't have any effects.

290.1.3.2 Storage Phases

These phases are responsible for validating and writing data to the KDB.
In the `prestorage` phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `resolver` phase of this `set` operation. The key name and value of this key are read-only.
- The exact subset below `parentKey` of the keyset `ks` that was provided by the user.

There are no restrictions on what the backend plugin may do in this phase. This phase can be used for validation to avoid storing invalid configuration. However, it should not be used for generating keys or values implicitly defined by other keys. Such keys should be generated during the `poststorage` phase of a `get` operation and should actually be *removed* again in this phase. That way there cannot be conflicts, if a key that implies another keys value changes.

> **Note**: Just in case there is actually a use case, where keys have to be generated, removed or modified during this phase, we do *not* discard changes to `ks` (like we would do in a `get` operation).

The `storage` phase is for writing the actual data. In this phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `prestorage` phase of this `set` operation. The key name and value of this key are read-only.
- The exact keyset `ks` that was returned by the `prestorage` phase of this `set` operation. All keys in this keyset and the keyset itself are fully read-only.

The backend plugin then:

- **MUST** serialize the data at and below `parentKey` in `ks`.
- **MUST** write the data for the mountpoint into the storage unit that was selected in the `resolver` phase.
- **MUST** ensure that the data is written in such a way that
  - it can be read, if the storage unit is mounted at another mountpoint
  - that reading such a mountpoint will result in the same data just below a different parent key

  An important implication here is that all names inside storage units should be relative to the parent key.

The last of the storage phases is the `poststorage` phase. In this phase the backend plugin is called with:

- The exact `parentKey` that was returned by the `storage` phase of this `set` operation. The key name and value of this key are read-only.
- The exact keyset `ks` that was returned by the `storage` phase of this `set` operation. All keys in this keyset and the keyset itself are fully read-only.

There are no formal restrictions, other than those enforced by `parentKey` and `ks` being (partially) read-only. But the `poststorage` phase should not be used as a counter-part to the `prestorage` phase in the `get` operation. Use the `precommit` phase instead. Therefore, the `poststorage` phase has very little use cases other than logging and exists mostly because of symmetry.

290.1.3.3 Commit Phases (<tt>set</tt> only)

If all storage phases completed successfully, `libelektra-kdb` will continue with calling the `commit` phases. Even though the `commit` phases are part of the `set` operation, `libelektra-kdb` calls `elektra<Plugin>Commit` and not `elektra<Plugin>Set` for these phases.
All the `commit` phases (`precommit`, `commit`, `postcommit`) are called with:

- The exact `parentKey` that was returned by the previous phase (`poststorage`, `precommit` or `commit`) of this `set` operation. The key name and value of this key are read-only.

- The exact keyset ks that was returned by the poststorage phase of this set operation (which is the same one that was returned by the prestorage phase). All keys in this keyset and the keyset itself are fully read-only.

There are no restrictions on the precommit phase, other than those enforced by parentKey and ks being (partially) read-only. This phase can be used for file-level manipulations, like file-based encryption, line ending conversion or adding file signatures. In this sense, it is the counter-part of the prestorage phase of the get operation.
In the commit phase the backend plugin:

- **MUST** make the changes done during the storage phase of this set operation permanent in such a way that a following get operation will be able to read them (assuming there is no other set operation in between).

There are no restrictions on what the backend plugin may do in the postcommit phase. However, it is important to keep in mind that an error in the postcommit phase will **not** make the set operation fail. Once the commit phase completes successfully, the set operation is also deemed successful, since the changes were made permanent. If an error does occur in the postcommit phase, it is reported as warning. This makes the postcommit phase mostly useful for logging.
Finally, libelektra-kdb merges the keyset returned by the postcommit phase (still the same one that was returned by the prestorage phase) with the ones returned by other backend plugins for different mountpoints and then returns it to the user.

290.1.3.4 Rollback Phases (<tt>set</tt> only)

If any of the phases prestorage, storage, poststorage, precommit or commit fail, libelektra-kdb will continue with the rollback phases. Even though the rollback phases are part of the set operation, libelektra-kdb calls elektra<Plugin>Error and not elektra<Plugin>Set for these phases.
Similar to the commit phases, the rollback phases (prerollback, rollback and postrollback) are called with:

- The exact parentKey that was returned by the phase of this set operation that reported an error. The key name and value of this key are read-only.
- The exact keyset ks that was returned by the phase of this set operation that reported an error. All keys in this keyset and the keyset itself are fully read-only.

Additionally, the phase that reported an error is communicated to the backend plugin via the global keyset (together with the current phase). The value of the key system:/elektra/kdb/backend/failedphase is set to the failed phase.
The prerollback and postrollback phases are mostly useful for logging. There are no restrictions on these phases, other than those enforced by parentKey and ks being (partially) read-only. However, they are similar to the postcommit phase, in that any errors they report will be ignored and reported as warnings. In particular, even if the prerollback phase fails, libelektra-kdb will continue with the rollback phase as if prerollback succeeded.
In the rollback phase the backend plugin:

- **MUST** revert all changes made to the state of the storage unit chosen in the resolver phase of this set operation. For file-based backends, this means removing the temporary file.
- **MUST** ensure that a following get or set operation will act as if the failed set operation never happened.
- **MAY** act differently depending on which phase failed.

Finally, libelektra-kdb will restore ks to the state in which the user provided it and return.

Chapter 291

Classes

This overview complements the introduction in the API documentation.

291.1 Key

A Key consists of a name, a value and metadata. It is the atomic unit in the key database. Its main purpose is that it can be serialized to be written out to permanent storage. It can be added to several aggregates using reference counting. Putting Key objects into other data structures of supported programming languages presents no problem.

291.2 KeySet

The central data structure in Elektra is a KeySet. It aggregates Key objects in order to describe configuration in an easy but complete way. As the name "set" already implies every Key in a KeySet has a unique name. A user can iterate over the Key objects of a KeySet. KeySet sorts the keys by their names. This yields a deterministic order advantage. So, independent of the appending sequence and, in particular, the number of fetches and updates, KeySet guarantees the same order of the Key objects. Some configuration storage systems need this property, because they cannot remember a specific order. On the other hand, any particular order can easily be introduced (See order).
On the one side backends generate or store a KeySet object and, on the other side, elektrified applications receive and send a KeySet object. Both sides, as well as the core in between, have the possibility to iterate, update, modify, extend and reduce the key set. Appending of new or existing Key objects extends the key set. Otherwise it can be reduced if keys are popped out. The Key object becomes independent of the KeySet afterwards. The user can still change such a key or append it into another key set. The affiliation to a key set is not exclusive.
Every key in a KeySet object has a unique name. Appending Key objects with the same name will override the already existing Key object.

291.3 KDB

While objects of Key and KeySet only reside in memory, Elektra's third class KDB actually provides access to the global key database. KDB, an abbreviation of key database, is responsible for actually storing and receiving configuration. KeySet represents the configuration when communicating with KDB. The typical elektrified application collects its configuration by one or many calls of kdbGet(). As soon as the program finishes its work with the KeySet, kdbSet() is in charge of writing all changes back to the key database.
This technique has some advantages. First, applications have full control over modifying Key and KeySet objects without touching the key database. Second, the decision how many KeySet objects the application administrates is left to the application. It can choose how to split up the KeySet objects. The main reason for this technique is that for backend development the same data structure is used, and as we will see, the borderline between application and backend development becomes blurred.
The application adapts the configuration between kdbGet() and kdbSet() in memory. The modifications are not only faster, they also allow large atomic configuration upgrades, robust merging of settings and handling of complicated inter-relationships between keys without problematic intermediate steps. Elektrified applications, however, should be aware of conflicts. It can happen that the key database is changed while working with a Key↩

`Set`. Then, attempts to use `kdbSet()` lead to a conflict. `KDB` detects such situations gracefully and lets the application decide which configuration should be used.
For details and background read more about elektra data structures. For further information see `the API documentation`.

Chapter 292

Data Structures

For an introduction, please read first about elektra classes. You might want to read about architecture first.

292.1 Introduction

Data structures define the common layer in Elektra. They are used for transferring configuration between Elektra and applications, but also between plugins.

292.1.1 ADT

Both the `KeySet` and the interface to metadata within a `Key` are actually **ADT** (abstract data types). The API is designed so that different implementations of the data structures can be used internally.
A sorted **array** provides very fast iteration O(1) and has nearly no size-overhead.
A **hash map** data structure presents the best candidate for lookups O(1).
`KeySet` combines the best of both worlds: `KeySet` is implemented as a sorted array and uses an `order-preserving minimal perfect hash map (OPMPHM)` for lookups.

292.1.2 ABI compatibility

Application binary interface, or ABI, is the interface to all data structures of an application or library directly allocated or accessed by the user.
Special care has been taken in Elektra to support all changes within the data structures without any ABI changes. ABI changes would entail the recompilation of applications and plugins using Elektra. The functions `keyNew()`, `ksNew()` and `kdbOpen()` allocate the data structures for the applications. The user only gets pointers to them. It is not possible for the user to allocate or access these data structures directly when only using the public header file <`kdb.h`>. The functions `keyDel()`, `ksDel()` and `kdbClose()` free the resources after use. Using the C++ binding deallocation is done automatically.

292.1.3 Copy-on-Write

The two basic Elektra datastructures `Key` and `KeySet` implement full copy-on-write (COW) functionality. If a key or a keyset gets copied, only a shallow copy with references to the original data (name, value, contained keys, etc.) is created. When this shared data is modified, new memory is allocated to keep the shared version in tact. As a consequence, duplicated keys or keysets only require a fraction of the memory compared to their source counterparts.

292.2 Metadata

Read here.

292.3 KeySet

This subsection describes what has changed between 0.7 and 0.8 and deals with some general implementation issues.

292.3.1 Operations

`KeySet` resembles the classical mathematical set. Operations like union, intersection or difference are well-defined. In mathematics typically every operation yields a new set. Instead, we try to reuse sets in the following ways:

- A completely new and independent `KeySet` as return value would resemble the mathematical ideal closely. This operation would be expensive. Every `Key` needs to be duplicated and inserted into a new `KeySet`.

  Such a **deep duplication** was only needed in `kdbSet()`.

- The resulting `KeySet` is created during the operation, but only a flat copy is made. This means that the keys in it are actually not duplicated, but only their reference counter is increased. This method is similar to the mathematical model. Compared with a deep copy it can achieve good performance. But all changes to the values of keys in the resulting `KeySet` affect the original `KeySet`, too.

  `ksDup(const KeySet *source)` produces a new `KeySet`. That way the `source` is not changed as shown by the `const` modifier.

- The result of the operation is applied to the `KeySet` passed as argument directly. This is actually quite common, but for this situation other names of the operations are more suitable.

  For example, a union which changes the `KeySet` is called `ksAppend()`.

- A new `KeySet` is created, but the `KeySet` passed as parameter is reduced by the keys needed for the new `KeySet`. This is useful in situations where many operations have to be applied in a sequence reducing the given `KeySet` until no more keys are left. None of the reference pointers change in this situation.

  `ksCut(KeySet *ks, const Key *cutpoint)` works that way. All keys below the `cutpoint` are moved from `ks` to the returned key set.

292.3.2 Consistency

There are several ways to define consistency relations on key sets. For **strict consistency** every parent key must exist before the user can append a key to a key set. For example, the key set with the keys

```
system:/
system:/elektra
system:/elektra/mountpoints
```

would allow the key `system:/elektra/mountpoints/tcl` to be added, but not the key `system:/apps/abc` because `system:/apps` is missing. File systems enforce this kind of consistency.

These semantics are however not useful for configurations. Especially for user configurations often only some keys need to be overwritten. It is not a good idea to copy all parent keys to the users' configuration. For this reason we use a less strict definition of consistency supporting such holes.

We also evaluated a form of **weak consistency**. It avoids adding some unnecessary keys. A constraint is that a key can be added only if it has a parent key. But the constraint does not apply if no other key exists above the key about to be inserted. From that moment it will serve as parent key for other keys. With the current implementation of `KeySet`, however, it is not possible to decide this constraint in constant time. Instead its worst-case complexity would be $log(n) * x$ where n is the number of keys currently in the key set and x is the *depth* of the key. The depth is the number of `/` in the key name. The worst-case of the complexity applies when the inserting works without a parent key. For example, with the keys

```
user:/sw/apps/abc/current/bindings
user:/sw/apps/abc/current/bindings/key1
user:/sw/apps/abc/current/bindings/key2
```

the weak consistency would allow inserting `user:/sw/apps/abc/current/bindings/key3` because it is directly below an existing key. It would also allow adding `user:/sw/apps/xyz/current` because it does not have any parent key. But it would not allow `user:/sw/apps/abc/current/bindings/dir/key1` to add. The worst-case complexity was found to be too expensive, and hence `KeySet` has **no consistency** check at all.

This means any key with a valid key name can be inserted into `KeySet`. The `KeySet` is changed so that it is now impossible to append keys without a name. `ksAppendKey(ks, Key *toAppend)` takes ownership of the

key `toAppend` and will delete the key in that case. The caller does not have to free `toAppend`: either it is in the key set or it is deleted.
Binary search determines the position where to insert a key. The C version of binary search `bsearch()` cannot tell where to insert a key when it is not found. So the algorithm has to be reimplemented. Java's binary search `binarySearch()` uses a trick to both indicate where a key is found and where to insert it with the same return code by returning the negative value `((-insertion point) - 1)` indicating where the new value should be inserted when the key is not found. Elektra now also uses this trick internally.

292.3.3 Iteration

Iterating over a `KeySet` is similar to iterating over arrays. With `ssize_t ksGetSize (const KeySet *ks)`, the total number of `Key`s in a `KeySet` can be retrieved and with the function `Key *ksAtCursor (const KeySet *ks, elektraCursor cursor)` the `Key *` at the specified position `cursor` in `ks` can be retrieved. The first element (`Key`) has the index 0, so the last `Key` in the `KeySet ks` can be accessed with `Key * k = ksAtCursor (ks, ksGetSize (ks) - 1)`. Please be aware the elements in a `KeySet` can move and therefore change their index, e.g. when deleting or adding elements or using `ksCut ()`.

```
for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
{
        Key * cur = ksAtCursor (ks, it);
        // ...
}
```

292.4 Order Preserving Minimal Perfect Hash Map (aka OPMPHM)

The OPMPHM is a non-dynamic randomized hash map of the Las Vegas type, that creates an index over the elements, to gain O(1) access.
The elements must be arranged in an array and each element must have a unique name, to identify the elements.
The source can be found in `kdbopmphm.h` and `opmphm.c` and also works outside of Elektra.
The OPMPHM does not store any buckets, your array of elements are the buckets and the OPMPHM represent an arbitrary index over those. The desired index of an element, also known as the order, is set in `OpmphmGraph->edges[i].order`, where `i` is the i-th element in your array. When the orders should represent the array indices, the default order can be applied during the assignment step. When the orders are not the default order, `OpmphmGraph->edges[i].order` should be set before the assignment step.
Because the OPMPHM is non-dynamic, there are no insert and delete operations. The OPMPHM gets build for a static set of elements, once the OPMPHM is build, every:

- change of at least one indexed element name
- addition of a new element
- deletion of an indexed element

leads to an invalid OPMPHM and forces a rebuild. A build consists of two steps the mapping step and the assignment step.
During the mapping step the OPMPHM maps each element to an edge in a random acyclic r-uniform r-partite hypergraph. In a r-uniform r-partite hypergraph each edge connects `r` vertices, each vertex in a different component. The probability of being acyclic and the number of mapping step invocations depends on the following variables:

- `r`: The `r` variable defines the number of components in the random r-uniform r-partite hypergraph. Use the `opmphmOptR (n)` function to get an optimal value for your number of elements (`n`).
- `c`: The `c` variable defines the number of vertices in each component of the random r-uniform r-partite hypergraph. The number of vertices in one component is defined as `(c * n / r) + 1`, where `n` is the number of elements and `r` is the variable from above. The `c` variable must have a minimal value to ensure a success probability, use the `opmphmMinC (r)` function, with your `r` from above. To ensure an optimal time until success increment the `c` variable with the value from the `opmphmOptC (n)` function, where `n` is the number of elements.
- `initSeed`: The initial seed set in `OpmphmInit->initSeed`.

`opmphmOptR (n)` and `opmphmOptC (n)` are heuristic functions constructed through benchmarks. Optimal is only one mapping step invocation in 99.5% of the observed cases. The benchmarks took arbitrary uniform distributed initial seeds and the heuristic functions are made to work with almost every seed.

292.4.1 The Build

292.4.1.1 Initialization

Use `opmphmNew ()` and `opmphmGraphNew (...)` to instantiate the needed structures. The function `opmphmGraphNew (...)` takes `r` and `c` as parameter. Use the `opmphmOptR (...)` function to get your `r` value, use this `r` also to get your `c` value the following way:
`c = opmphmMinC (r) + opmphmOptC (n)`
To initialize the OPMPHM build the `OpmphmInit` must be set with information about your data. Set your data array `OpmphmInit->data` and the element name extraction function `OpmphmInit->getName`, which should extract the string from a single data array entry. Provide a good seed in `OpmphmInit->initSeed`, needed in the next step.

292.4.1.2 Mapping

The function `opmphmMapping` uses your seed (the `OpmphmInit->seed` will be changed) and tries to construct the random acyclic r-uniform r-partite hypergraph, this might not succeed, on cycles just call it again.

292.4.1.3 Assignment

The `opmphmAssignment ()` function assigns either your order (set at `OpmphmGraph->edges[i].order`) or a default order. The `defaultOrder` parameter indicates the behavior.
After the build the OpmphmInit and OpmphmGraph should be freed. The OPMPHM is now ready for constant lookups with the `opmphmLookup ()`.

292.4.2 The Rebuild

Once build, follow the steps from the build, just omit the `opmphmNew ()` invocation.

Chapter 293

Developer's glossary of Elektra

Chapter 294

Error Categorization

This document explains how to categorize errors and should act as a form of guideline. The categorization aims to reduce duplicated errors and the maintenance effort for errors.

294.1 Categorization Mindset

Errors along with their unique code only primarily exist because they can be **reacted differently on programmatically**. To get an idea of programmatic reactions take a method call which returns a `Timeout` error. Naturally a senseful reaction would be to retry at a later point in time. So the reaction here would be to retry with a time based approach. On the other hand it does not make sense to differentiate between `No Write Permission` and `No Read Permission` as the application just knows that it simply cannot access the desired resource and tells the user to grant it.
Categories are hierarchically structured. In some categories you cannot put an error such as `Permanent errors` (see below) because they are too general and developers should choose a more specific category. Even though most categories which allow putting errors into are at the end of the hierarchy (leafs), it may also be the case that a more general category also allows errors to be put in (e.g., `Resource Errors`). The categories below are marked as `concrete` or `abstract`, meaning that you either can or cannot create errors for the category. Please choose the most specific category as possible when trying to assign an error to a category.
If you feel for a new category, please forge a design decision document and make a PR to Elektra's repo.

294.2 Error categorization Guideline

Now we will investigate each category in more detail and when to put an error/warning in there.
For a complete structural overview please visit the corresponding design decision document

294.2.1 Permanent errors ("C01000", abstract)

The branch category `Permanent Errors` refer to such errors which cannot be fixed by retry at all. `Permanent Errors` are subdivided into

- Resource
- Installation
- Logical

294.2.1.1 Resource ("C01100", concrete)

`Resource Errors` are all kind of permission, existence and resource errors which are essential for Elektra to operate. Resource errors is a branch category which also allows for errors to be put in. Compared to validation errors this category has its focus the underlying system resources whereas validation errors for usages with specifications. Examples are missing files/ directories or insufficient permission to execute certain commands (e.g. you would require sudo permissions). Reactions are fixing the permissions, remount, creating the file/directory and retry the operation. Compared to validation errors, administrators would change the specification or retry with a different value.

294.2.1.1.1 Out of Memory ("C01110", concrete) `Out of Memory Errors` are special resource errors which come from failed `elektraMalloc` calls primarily as no more memory could be allocated for the application. Errors with not enough hard disc space do not belong here but into `Resource`. Such errors will gain special handling in future releases and users cannot deal with such errors as of now.

294.2.1.2 Installation ("C01200", concrete)

`Installation Errors` are errors that are related to a wrong installation such as wrong plugin names, missing backends, initialization errors, misconfiguration of Elektra etc. Installation errors might also be non-Elektra specific but also from dependent library/applications such as gpg. Also, plugin configuration errors belong to `Installation Errors` as this happens during mounting. Users will have to reconfigure, reinstall, recompile (with other settings) Elektra in order to get rid of this error or fix the installation of the corresponding library/application.

294.2.1.3 Logical ("C01300", abstract)

`Logical Errors` is a branch category in which you indicate a logical flaw in the code such as internal errors, not implemented features, passing illegal parameters to functions, plugins which do not behave accordingly (wrong return codes, uncaught exceptions) or errors which should not happen such as going into a `default` branch when you are assured that all cases are covered. Usually such errors come with a message to report such failures to Elektra's bugtracker. Applications cannot handle such errors themselves.

294.2.1.3.1 Internal ("C01310", concrete) `Internal Errors` are such errors which indicate a flaw or bug in the internal logic of Elektra. Examples are going into a `default` branch which you do never expect to happen. Another use case is if you use an external library and have to catch a generic exception. If you can however, catch the most specific exceptions and convert them into the appropriate category. If you have to use this error please add a message indicating that this bug should be reported.

294.2.1.3.2 Interface ("C01320", concrete) `Interface Errors` indicate a wrong usage of Elektra's API. Compared to `internal` errors this category does not indicate a bug but a misuse. An example would be to pass a NULL pointer to `kdbGet`. Also, violations of the backend belong into this category. Compared to semantic validation errors, this category has its focus on detecting wrong usages of the API instead of a "retry with a different value" approach. Also, validation errors focus on specifications of configurations whereas this category tries to handle specifications for APIs. Users should investigate the concrete reason and might use a different, more appropriate method or change their passed values before retrying.

294.2.1.3.3 Plugin Misbehavior ("C01330", concrete) `Plugin Misbehavior Errors` indicate that a plugin does not behave in an intended way. Unrecognized commands, unknown return codes, plugin creation errors, etc. belong to this category. Also, uncaught exceptions belong here because Elektra expects all exceptions to be caught. For wrong plugin versions please use `Installation` errors. Users can try to remount, recompile the plugin under different options or use a different plugin (e.g., switching to `mini` instead of `ini`).

294.2.2 Conflicting State ("C02000", concrete)

`Conflicting State Errors` are errors where the current state is incompatible with the attempted operation. These kind of errors are usually in resolver plugins when the state of the file has changed without the system knowing. An example would be to try to push your changes into a Git repository where the remote branch has already changed. Try to synchronize your internal state and retry to get rid of this error. Examples are the need for calling `kdbGet` before `kdbSet`.

294.2.3 Validation ("C03000", abstract)

`Validation Errors` are heavily used for Elektra's `configuration specification` feature and should tell users that their given input does not match a certain pattern/type/expected semantic etc.
Validation errors can either be syntactic or semantic.

294.2.3.1 Syntactic ("C03100", concrete)

`Syntactic Errors` are errors which tell users or applications that the current format is not valid. Examples are wrong date formats or missing closing brackets `]` inside of a regular expression when it is checked. Also, path related errors like missing slashes come into this category. Parsing errors are also associated with syntactic errors such as unexpected encounters like missing line endings, illegal characters or wrong encodings. Also, transformation and conversion errors are to be categorizes here because the format of the given input does not allow such actions. Since syntactic errors demand a specific format and structure, also structural validation errors belong here. Users should try a different value/format and retry setting it with Elektra.

294.2.3.2 Semantic ("C03200", concrete)

`Semantic Errors` are errors which indicate a misunderstanding of the intended meaning between a user's/developer's/administrator's way of seeing a setting and the application's one. The main focus of this category is to enforce specifications compared to other categories. So if users provide input which do not match prespecified criteria even though syntactically everything is valid, this category should be used. Examples are references to non-existent keys (`reference` plugin), setting values to keys which require to be empty (`required` plugin), wrong provided values in a specification if you restricted the values (`enum` plugin), etc.
Users should try a different value or fix the underlying specification and retry again.

294.3 Underlying design decision document

- Decision Document The underlying design decision document.

Chapter 295

Error handling

You might want to read about data structures first.

295.1 Terminology

It is sometimes unavoidable that errors occur that ultimately have an impact for the user. One example for such an error is that hard disc space is exhausted. For a library it is necessary to pass information about the facts and circumstances to the user, because the user wants to be informed why a requested action failed. So Elektra gathers all information in these situations. We call this resulting information **error information** or **warning information** depending on the severity.
If the error is critical and ultimately causes a situation in which the post conditions cannot be fulfilled we say that Elektra is in a **faulty state**. Such a faulty state will change the control flow inside Elektra completely. Elektra is unable to resolve the problem without assistance. After cleaning up resources, a faulty state leads to immediate return from the function with an **error code**. As a user expects from a library, Elektra never calls `exit()` or something similar, regardless of how fatal the error is. In this situation error information should be set.
On the other hand, for many problems the work can go on with reasonable defaults. Nevertheless, the user will be warned that a problem occurred using the **warning information**. These situations do not influence the control flow. But applications can choose to react differently in the presence of warning information. They may not be interested in any warning information at all. It is no problem if warning information is ignored because they are stored and remain accessible in a circular buffer. The implementation prevents an overflow of the buffer. Instead, the oldest warnings are overwritten.
When error or warning information is presented to the user, it is called *error message* or *warning message*. The user may reply to this message in which way to continue.

295.1.1 Error vs. Warning Information

When an error in a faulty state occurs, the error information must still hold the original error information. So even if a new problem would cause a faulty state otherwise, the error information must be omitted or transformed to warning information. In some places only the addition of warning information is possible:

- The main purpose of `kdbClose()` is to free the handle. This operation is always successful and is carried out even if some resources cannot be freed. Therefore, in `kdbClose()`, setting error information is prohibited. Warning information is, however, very useful to tell the user the circumstance that some actions during cleanup failed.
- Also in `kdbOpen()`, only adding warning information is allowed. If `kdbOpen()` is not able to open a plugin, the affected backend will be dropped. The user is certainly interested why that happened. But we decided not to make this a faulty state, because the application might not even access the faulty part of the key hierarchy. An exception to this rule is if `kdbOpen()` fails to open the default backend. This situation will induce a faulty state.
- In `kdbSet()`, the clean-up of resources involves calling plugins. But during this process Elektra is in a faulty state, so only adding of warning information is allowed. This ensures that the original error information is passed unchanged to the user.

On the other hand, any access to the key database can produce warning information. Plugins are allowed to yield warning information at any place and for any reason. For example, the storage plugin reading a configuration file with an old syntax, is free to report the circumstance as warning information. Warning information is also useful when more than one fact needs to be reported.

295.2 Error Information

295.2.1 Reporting Errors

Reporting errors is a critical task. Users expect different aspects:

- The **user of the application** does not want to see any error message at all. If it is inevitable, he or she wants little, but very concrete information, about what he or she needs to do. The message should be short and concise. Some error information may already be captured by the application, but others like “no more free disk space” have to be displayed. Conflicts should also be presented to the user. It is a good idea to ask how to proceed if a diversity of possible reactions exists. In case of conflicts, the user may have additional knowledge about the other program which has caused the problem. The user is more likely to decide correctly by which strategy the configuration shall be restored.
- The **user of the library** wants more detailed information. Categories of how severe the error is can help to decide how to proceed. Even more important is the information if it makes sense to try the same action again. If, for example, an unreliable network connection or file system is used, the same action can work in a second try.
- A **developer of the library** (Library refers to both Elektra’s core and plugins.) wants full information about anything needed to be able to reproduce and locate potential bugs. Ideally the error information should even mention the file and line where the error occurred. This can help developers to decide if there is a bug inside Elektra or if the problem lies somewhere else.
- Vast information is needed to support correct error handling in *other programming languages*. In languages supporting exceptions, class name, inheritance or interface information may be necessary. Language specific extensions are, however, not limited to exceptions. Other ways of handling errors are continuations or `longjmp` in C. A plugin is free to add, for example, `jmp_buf` information to the error information.

It is certainly not a good idea to put all this previously mentioned information into a single string. Elektra chooses another way described in the next chapter.

295.2.2 Metadata

As stated above, a library always informs the user about what has happened, but does not print or log anything itself. One way to return error information is to add a parameter containing the error information. In the case of Elektra, all `KDB` methods have a key as parameter. This key is additionally passed to every plugin. The idea is to add the error and warning information as metadata to this key. This approach does not limit flexibility, because a key can hold a potentially unlimited number of metakeys.
The error information is categorized in metadata as follows:

- [error] indicates that a faulty state is present. The value of the metakey contains the name of all the subkeys that are used for error indication. Metakeys do not guarantee any particular order on iteration. Instead, the user can find out the information by looking at this metavalue.

Additional metakeys yield all the details.

- [error/number] yields a unique number for every error.
- [error/description] is a description for the error to be displayed to the user. For example, the metavalue can hold the text “could not write to file”.
- [error/reason] specifies the reason of the error. The human readable message is in the metavalue of `error/reason`. It states why the error occurred. One example for it is "no disc space available".
- [error/module] indicates the name of the specific module or plugin.

- [error/file] yields the source file from where the error information comes.
- [error/line] represents the exact line of that source file.

Beside errors, Elektra can also emit warnings metadata. While only a single error can be set on a specific error key, warnings can be up to 100 entries (#0 - #_99):

- [warnings] indicate that at least one warning is present. The value of the metakey contains the number of warnings which can be accessed.

Additional metakeys yield all the details. The warnings are stored in a special array format which range from 0 to _99. E.g., the first warning number can be accessed by getting the key `warnings/#0/number`. The following metadata is available and have the same semantics as the error metadata: `[warnings/<number>/number]`, `[warnings/<number>/description]`, `[warnings/<number>/reason]`, `[warnings/<number>/module]`, `[warnings/<number>/file]`, `[warnings/<number>/line]`
If there are more than 100 warnings, the information will be overwritten from the start again.
As we see, the system is powerful because any other text or information can be added in a flexible manner by using additional metakeys.

295.2.3 Error Specification

The error specification in Elektra is written in simple colon-separated text files. Each entry has a unique identifier and all the information we already discussed above. No part of Elektra ever reads this file directly. Instead, it is used to generate source code which contains everything needed to add a particular error or warning information. With that file we achieved a central place for error-related information. All other locations are automatically generated instead of having error-prone duplicated code. This principle is called “Don't repeat yourself”.
In Elektra’s core and plugins, C macros are responsible for setting the error information and adding warning information. In C only a macro is able to add the file name and line number of the source code. In language bindings other code may be generated.

295.2.4 Sources of Errors

`Key` and `KeySet` functions cannot expose more error information than the error code they return. But, of course, errors are also possible in these functions. Typically, errors concern invalid key names or null pointers. These problems are mostly programming errors with only local effects.
The most interesting error situations, however, all occur in `KDB`. The error system described here is dedicated to the four main `KDB` functions: `kdbOpen()`, `kdbGet()`, `kdbSet()` and `kdbClose()`. The place where the configuration is checked and made persistent is the source of most error information. At this specific place a large variety of errors can happen ranging from opening, locking up and saving the file. Sometimes in plugins, nearly every line needs to deal with an error situation.

295.3 Exceptions

Exceptions are a mechanism of the language and not just an implementation detail. Exceptions are not intended to force the user to do something, but to enrich the possibilities. In this section, we discuss two issues related to exceptions. On the one hand, we will see how Elektra supports programming languages that provide exceptions. On the other hand, we will see how the research in exceptions helps Elektra to provide more robustness.

295.3.1 Language Bindings

C does not know anything about exceptions. But Elektra was designed so that both plugins and applications can be written in languages that provide exceptions. One design goal of Elektra’s error system is to transport exception-related information in a language neutral way from the plugins to the applications. To do so, a language binding of the plugin needs to catch every exception and transform it into error information and return an error code.
Elektra recognizes the error code, stops the processing of plugins, switches to a faulty state and gives all the plugins a chance to do the necessary cleanups. The error information is passed to the application as usual. If the application is written in C or does not want to deal with exceptions for another reason, we are finished because the application gets the error information inside metadata as expected. But, if the application is written in another language, the

binding translates the error code to an exception and throws it. It is worth noting that the source and target language do not need to be the same.
Such a system needs a central specification of error information. We already introduced such a specification file in error specification. The exception classes and converters can be generated from it. An exception converter is either a long sequence of try-catch statements that transforms every known exception into an appropriate metakeys. Each exception thrown by the plugin has to be caught. Alternatively, a converter can be a long switch statement for every error number. In every case the appropriate exception is thrown.
The motivation for using exceptions is that in C every return value has to be checked. On the other hand, the C++ exception mechanism allows the programmer to throw an exception in any place and catch it where it is needed. So in C++ the code is not cluttered with parts responsible for error handling. Instead, in a single place all exceptions of a plugin can be transformed to Elektra's error or warning information. The code for this place can be generated automatically using an exception converter.
Applications not written in C can also benefit from an exception converter. Instead of using the metadata mechanism, the error information can be converted to the exception mechanism already used for that application. We see that Elektra is minimally invasive in this regard.

295.3.2 Exception Safety

We can learn from the way languages define the semantics for **exception safety**. Exception safety is a concept which ensures that all resources are freed regardless of the chosen return path. **Basic guarantees** make sure that some invariants remain on an object even in exceptional cases. On the other hand, **strong guarantees** assure that the investigated operation is successful or has no effect at all. Methods are said to be exception safe if the object remains in a valid state. The idea of exception safety is to ensure that no resource leaking is possible. `kdbSet()` written in C++ would look like:

```
 {c++}
try {
        plugin[1].set(); // may throw
        plugin[2].set(); // may throw
        plugin[3].set(); // may throw
        ...
        plugin[PLUGIN_COMMIT].set(); // now all changes are committed
} catch (...) {
        // error situation, roll back the changes
        plugin[1].error(); // does not throw
        plugin[2].error(); // does not throw
        plugin[3].error(); // does not throw
        ...
        // now all changes are rolled back
        return -1;
} // now do all actions on success after commit
plugin[POSTCOMMIT].set(); // does not throw
...
return 1; // commit successful
```

This pseudocode is much clearer than the corresponding C code. Let us explain the guarantee Elektra provides using this example. One by one plugin gets its chance to process the configuration. If any plugin fails, the semantics resemble that of a thrown exception. All other plugins will not be executed. Instead, the plugins get a chance to recover from the faulty state. In this catch part, the plugins are not allowed to yield any error information, but they are allowed to add warnings.
Continue reading with the algorithm.

Chapter 296

Error Message

Elektra has a lot of developers which leads to error messages written in various ways. This guideline aims to unify the format of messages so that users see consistent error messages.

- All error messages start with a capital letter. Reword sentences that theoretically require a lowercase first letter such as regex patterns, method names or variable strings ("%s"-strings) to have a normal word at the start of the sentence. Also surround method names with single quotes. Example: `kdbSet() not supported` should be changed to 'Method 'kdbSet()' is not supported`
- Always report all information which could be useful for the user. For some error types, templates are provided to not forget something. (eg. keys, files, errno, etc.)
- If the error reason is in beneath a variable string (such as `errno` or a caught exception), write `"Reason: %s", variable_as_string` at the end of the error message. Also end the preceding sentence with a dot. Example: `"XY failed. Reason: %s", exception.what()`.
- The last sentence (or single sentence of your message if you just provide one) must not end with a dot. This should encourage users to continue reading if other messages appear.
- If your message has multiple sentences, separate them with dots and start with a capital letter like you would do in the normal English language.
- Short but many sentences are preferable over long ones.
- Use abbreviations and acronyms with care. Not everybody will know them. You can write them in brackets though. (eg. `No Byte Order Mark (BOM) found` instead of `No BOM found`. But `XML does not support feature xy` is fine).
- Never use exclamation marks at the end of sentences.
- If it is unclear where embedded strings in error messages start and end ("%s"-strings), surround them by single quotes '. Sentences containing a variable string and end with `Reason: s` do not need extra quotes because it is clear where they start and end.

296.1 Examples

Actual reason might be in errno or an exception

- `"The configuration file %s contains invalid syntax at line %s. Reason: %s", file.path(), line, e.what()` Result: `The configuration file /etc/conf/file.csv contains invalid syntax at line 6. Reason: Missing column`
- `"Could not rename file %s. Reason: %s" file.path(), e.what()` Result: `Could not rename file /etc/conf/file.yml. Reason: File is not existent`
- '"The key %s contained the value '%s', but only 'unmounted' is supported for non-global clauses at the moment", keyName(key), keyString(key)`Result:` The key user:/my/key contained the value 'mounted', but only 'unmounted' is supported for non-global clauses at the moment`

Chapter 297

History

297.1 Elektrify

The development of Elektra started in 2004. Initially the initiative only aimed at the straightforward idea to unify a configuration access API and a configuration format. Elektra started by introducing an API with a few language bindings.
The idea of introducing a configuration access API was not new: Most proprietary software systems already had similar APIs for a long time. Nevertheless, it was clear that there was no portable configuration library available to be used for typical FLOSS applications. For example, the configuration libraries X/Q/GSettings, KConfig, dconf, plist, and Windows Registry are tied to their respective platforms.
The first idea to improve adoption of Elektra was that the Elektra Initiative itself patches applications to use Elektra. This was done, e.g., for the X Server. The maintainers of the X Server, however, had other plans and later improved their software to auto-detect hardware. This was admittedly the right choice for them: Elektra would not have been able to help them at that time.

297.2 Mounting

One obvious limitation of Elektra was that no gradual migration is possible and previous configuration files would need to be rewritten. In a first step, we introduced backends, chosen by users at run-time.
An implementation providing only a single backend for the whole system proved to be too limited: Individual applications cannot customize their backends for their own needs. We implemented a layer similar to a virtual file system, which enables applications and system administrators to mount configuration files.

297.3 Dependencies

Furthermore, dependences in the core made Elektra unattractive. To solve this problem, we modularized Elektra so that users can select exactly the features they need.
Such changes made part of the software more complicated. At first, we provided too little documentation for newcomers to grasp the abstraction mechanisms. Then we started to put efforts into rebuilding the community by overhauling documentation, introducing more regression tests, writing tutorials, and designing a new website.

297.4 Elektrify: Back Again

In the beginning of Elektra, there was no vision how configuration management and Elektra would work together. After many years of research, we now have a clear vision how a world with Elektra would look like.
To actually improve the adoption of Elektra, however, configuration management cannot help directly. Instead we need applications that depend on Elektra, so that Elektra gets packaged for all distributions. We now elektrify KDE, GNOME and XFCE to achieve this goal. Furthermore, we are now in the process of doing the last cleanups before version 1.0.

Chapter 298

Hooks

Hooks are central points in the KDB lifecycle, where specialized plugins are called.

298.1 Selecting which Plugin will be Used for a Specific Hook

The names of the plugins are hard coded. This decision was made, because these plugins are meant to fulfil very specific purposes. A symlink replacing the shared library file of the plugin could be used to change the implementation.

298.2 Interface of the hooks

If a plugin should be able to act upon a hook, it must export all the functions that the hook requires. These exports are of the form `system:/elektra/modules/<plugin name>/exports/hook/<hook name>/<hook function>`.
For example, the `gopts` hook only requires the `get` function. A plugin that wants to act upon the `gopts` hook therefore has to export `system:/elektra/modules/<plugin name>/exports/hook/gopts/get`. Other hooks (e.g. `spec`) require multiple exported functions.

298.2.1 <tt>gopts</tt> hook

Hard coded to search for a plugin named `gopts`.
The following function **must** be exported:

- `get`
  - Signature: `(Plugin * handle, KeySet * returned, Key * parentKey)`
  - Called in `kdbGet` after the storage phase, after `notification/send` hook but before the `spec` hook.
  - TODO: Describe what the function should do

298.2.2 <tt>spec</tt> hook

Hard coded to search for a plugin named `spec`.
The following functions **must** be exported:

- `copy`
  - Signature: `(Plugin * handle, KeySet * returned, Key * parentKey, bool isKdbGet)`
  - Called in:
    - `kdbGet`: after the storage phase, after `notification/send` and `gopts` hook.
    - `kdbSet`: right after the backends are initialized
  - Should copy all the spec meta keys into the keyset

- remove
  - Signature: (Plugin * handle, KeySet * returned, Key * parentKey)
  - Called in kdbSet right after the prestorage phase
  - Should remove all the spec meta keys from the keyset

298.2.3 <tt>notification/send</tt> hook

We look within the array system:/elektra/hook/notification/send/plugins for the plugins that shall be loaded. The name of the plugin **must** be the value of the keys directly below this, e.g. system:/elektra/hook/notification/send/plugins/#0 (= dbus).
The following functions **may** be exported (optional):

- get:
  - Signature: (Plugin * handle, KeySet * returned, Key * parentKey)
  - Called in kdbGet after the storage phase.
- set:
  - Signature: (Plugin * handle, KeySet * returned, Key * parentKey)
  - Called in kdbSet after the storage phase.

298.2.4 <tt>record</tt> hook

Used for the session recording plugin. Hard coded to search for a plugin named recorder.
The following function must be exported:

- record:
  - Signature: (Plugin * handle, KeySet * returned, Key * parentKey)
  - Called in kdbSet after the storage phase.
  - Must not modify the returned keyset.
  - The parentKey must not be modified, except for adding errors and warnings.
  - Calculates the changes and stores them.
- lock:
  - Signature: int (Plugin * handle, Key * parentKey)
  - Called in kdbSet before the storage phase.
  - The parentKey must not be modified, except for adding errors and warnings.
  - Must ensure that this is only process that can record changes until unlock is called.
    - If successful, must return ELEKTRA_PLUGIN_STATUS_SUCCESS.
    - If not successful, must return ELEKTRA_PLUGIN_STATUS_ERROR
- unlock:
  - Signature: int (Plugin * handle, Key * parentKey)
  - Called in kdbSet before returning, after lock has been called.
  - The parentKey must not be modified, except for adding errors and warnings.
  - Must remove any acquired locks and mutexes, so that other processes can record changes again.

298.3 Lifecycle

1. Hooks are initialized within kdbOpen after the contract has been processed. This includes loading the plugins.
2. The appropriate hooks are called within each kdbGet and kdbSet call.
3. Hooks are deinitialized within kdbClose. This includes unloading the plugins.

298.4 Additional information

To specify the place of a hook plugin add `infos/placements = hook` to your plugins `README.md`.

Chapter 299

Migrating from internal to external KeySet iterators

The deprecated internal iterator are removed and replaced with external iteration.

299.1 Interactions with users

Developers of plugins should already use the external iterators instead of the internal ones. However, following functions will be removed:

```
int ksRewind (KeySet *ks);
Key *ksNext (KeySet *ks);
Key *ksCurrent (const KeySet *ks);
elektraCursor ksGetCursor (const KeySet *ks);
int ksSetCursor (KeySet *ks, elektraCursor cursor);
int keyRewindMeta (Key *key);
const Key *keyNextMeta (Key *key);
const Key *keyCurrentMeta (const Key *key);
```

299.2 The Architecture

Every occurrence of using the internal iterator has to be replaced with the external iterator (see #4281). The functions of the internal iterator are listed in #3171. There are also some uses of the internal iterator found in the SWIG bindings (Lua, Ruby, Python) that also need removal (see #4279).

299.2.1 Internal vs. external Iterators

When the client (i.e. the programmer) controls the iteration, the iterator is called external iterator. Otherwise, when the iterator controls the iteration, it is called internal iterator. Due to the fact that external iterators are more flexible the internal iterators are the way to go.

299.3 Examples

An example for using an external iterator is:

```
for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
{
    Key * cur = ksAtCursor (ks, it);
    // [loop body]
}
```

You can obtain the `Key` at a specific position in the `KeySet` and the overall size of the `KeySet`.
If you want to delete `Key`s during the iteration of a `KeySet`, be aware that all keys after the deleted `Key` are moved one slot forward, so maybe you have to change to value of `it` after deleting a `Key`:

```
for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
{
    Key * cur = ksAtCursor (ks, it);
    if ( shouldKeyGetDeleted (cur))
    {
            keyDel (cur);
            --it; //next key is now at the position of the current key which was deleted
    }
}
```

For such scenarios, it is also important that you recalculate the size with `ksGetSize ()` within the loop-header or explicitly after changing the `KeySet`, e.g. by deleting a `Key`.

That should be all you need for iterating over keys. For future releases, the function `ksAtCursor` will be renamed to `ksAt`. (see issue #3976)
You can iterate over metakeys in a similar fashion as long as you iterate over a `KeySet`, e.g., `keyMeta("my_↩ key")` returns a `KeySet` of metakeys.
The following is a comprehension of how to use iterators in various languages.

299.3.0.1 C

```
for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
{
    Key * cur = ksAtCursor (ks, it);
    // ...
}
```

299.3.0.2 C++

C-style fashioned loop:

```
for (elektraCursor it = 0; it < ks.size (); ++it)
{
        Key key = ks.at(it);
        // ...
}
```

Real iterators the C++ interface supports:

```
for (KeySet::iterator i = ks3.begin(); i != ks3.end(); ++i)
{
        Key key(*i);
        // ...
}
```

299.3.0.3 Python

```
size = ksSize(keySet)
for cursor in range(size):
        key = ksAt(keySet, cursor)
        # ...
```

299.3.0.4 Lua

```
size = kdb.ksSize(keySet);
if size > 0 then
        for i = 1, size do
                key = kdb.ksAt(keySet, i - 1)
                -- ...
        end
end
```

299.3.0.5 Ruby

```
keySet.each do |key|
        # ...
end
```

Chapter 300

KDB Contracts

The `kdbOpen()` function accepts a `KeySet * contract` parameter. This parameter allows you to extend and configure the KDB within your application without permanently changing files on disk.

300.1 Contract Structure

The contract consists of Keys below `system:/elektra/contract/<type>`, where `<type>` is one of a set of predefined contract types. Currently, the types `globalkeyset` and `mountglobal` are supported.

300.1.1 Global KeySet Contracts

To insert data into the global KeySet during `kdbOpen`, you can add keys below `system:/elektra/contract/globalkeyse`
All these keys will be renamed and copied into the global KeySet that is passed to all plugins.
Specifically, the subset below `system:/elektra/contract/globalkeyset` is moved to `system:/elektra` and then inserted into the global KeySet.

300.1.2 Mounting Global Plugins

You can also mount a global plugin with a contract. To do this, add a key `system:/elektra/contract/mountglobal/<plu`
where `<plugin>` is the name of the plugin you want to mount. The keys below `system:/elektra/contract/mountgloba`
will be moved to `user:/` and used as the config for `<plugin>`.

300.2 Pre-defined Contracts

There are a few pre-defined contracts that can be accessed via helper functions.

300.2.1 GOpts

To mount and configure the `gopts` plugin you can use `elektraGOptsContract`. There is also `elektraGOptsContractFromStrings`, but its use is discouraged unless you really need it. It mainly exists for use from various bindings.

300.2.2 I/O binding

To set the I/O binding to be used in a KDB instance, use `elektraIoContract`.

300.2.3 Notification

To set up notifications use the `elektraNotificationContract` function. It automatically sets up the `internalnotification` plugin.
If you also need to set up notification transport plugins, you should manually add the relevant `system:/elektra/contract/mountglobal/<plugin>` keys and the required config below.

Chapter 301

KDB Operations

There are four main *operations* in `libelektra-kdb`: `open`, `get`, `set` and `close`. For each of these there is a `kdb*` function the user calls to trigger the operation and plugins export a function for each of the operations they support (at least `get`).
Additionally, plugins may implement `commit` and `error`. These are part of the `set` operation and there is no corresponding `kdbCommit` or `kdbError` function available in `libelektra-kdb`.
The operations `get` and `set` also have different *phases*:

- The `get` operation has: `init`, `resolver`, `cachecheck`, `prestorage`, `storage` and `poststorage`.
- The `set` operation has: `resolver`, `prestorage`, `storage` and `poststorage` followed by `precommit`, `commit` and `postcommit` if the previous phases where successful or by `prerollback`, `rollback` and `postrollback` if the previous phases failed.

These phases are implemented by a backend plugin. Read the Documentation on Backend Plugins for more information on what backend plugins do.

> **Note** The steps of the operations described below, are referenced in the source code with `// Step X` comments.

301.1 <tt>open</tt> Operation

The `open` operation implemented in `kdbOpen` is the first thing that happens to all `KDB` instances.
The basic flow of this operation is:

1. Create empty `KDB` instance
2. Configure `KDB` instance for bootstrap
3. Run bootstrap `get` operation: This loads the contents of `system:/elektra/mountpoints` so that the mountpoints can be configured.
4. Process contract and set up plugins for hooks (see Hooks)
5. Parse mountpoints: This transforms the contents of `system:/elektra/mountpoints` into the internal state stored in a `KDB` instance.
6. Reconfigure `KDB` with real mountpoints: This switches the `KDB` instance from bootstrap mode to use the real mountpoint state created above.
7. Add hard coded mountpoints to `KDB` instance: There are a few hard coded mountpoints (root mountpoints, `system:/elektra/modules`, `system:/elektra/version`, etc.) that are always present. They are added in this step.

Namespaces in mountpoint configs:

- `dir:/`, `user:/` and `system:/` mountpoints can be created without restrictions, except for the reserved sections listed below.
- `spec:/` mountpoints can be created with the same restrictions, but they are also treated specially during `get` and `set`.
- `proc:/` mountpoints are always read-only and receive special treatment during `get`
- `default:/` mountpoints are read-only and receive special treatment during `get`, specifically they only go through the `poststorage` phase
- mountpoints in all other namespaces are entirely illegal

  Note The special treatments of the various namespaces are explained below in the sections for the `get` and `set` operation.

Reserved sections:

- Creating a mountpoint for `/elektra` or below in *any namespace* is forbidden. This section of the KDB is reserved for Elektra's own config.
- `system:/elektra/mountpoints`, `user:/elektra/mountpoints` and `dir:/elektra/mountpoints` are all required for the bootstrap process and use a hard coded backend. The backends are implemented by a standard file-based backend plugin that is defined at compile-time of `libelektra-kdb`.
- `system:/elektra/version` and `system:/elektra/modules` will always use hard coded read-only backends containing information about this Elektra installation. The backends are implemented by special purpose backend plugins.

301.2 <tt>get</tt> Operation

The purpose of the `get` operation is to read data stored in backends into a `KDB` instance.

Note: Some details of a `get` operation are defined in the contract with backend plugins.

Properties of `kdbGet()`:

- After calling `kdbGet (kdb, ks, parentKey)`, the KeySet `ks` will contain *all keys* (including their values) that are stored in *any backend* with a mountpoint that is *below* *`parentKey`*.
- After calling `kdbGet (kdb, ks, parentKey)`, below `parentKey` the KeySet `ks` will *mostly* contain keys that are stored in a backend. The exception here are `proc:/` and `spec:/` keys. For other namespaces, all keys below `parentKey` will be removed from `ks`. For `proc:/` and `spec:/` only keys that overlap with a backend that was loaded will be removed from `ks`.
- The KeySet `ks` *may* contain other keys not below `parentKey`:
  1. Keys that are not below `parentKey`, but are stored in a backend that contains other keys which are below `parentKey`. These keys are returned, because backends are treated as one atomic unit. Either all keys within a backend are read, or none of them are.
  2. Keys that were already present in `ks` when `kdbGet()` was called and do not conflict with the goal of representing the current state of the KDB below `parentKey`.
- After calling `kdbGet (kdb, ks, parentKey)`, the Key `parentKey` will only have the `meta:/error/*` or `meta:/warnings/#/*` metakeys, if the errors/warnings originate from this `kdbGet()` call. In other words, `kdbGet ()` first clears any existing errors/warnings and only then starts doing the actual work.
- It is an error to use a `parentKey` with a namespace other than: `default:/`, `proc:/`, `spec:`, `dir:/`, `user:/`, `system:/` or cascading

To the caller it looks as if kdbGet() had removed all keys below parentKey, as well as some others, from ks and then loaded the data from the backends. Which backends are actually read is an implementation detail. Which keys are removed from ks depends on the backends that are read.
kdbGet() will always try to be efficient in achieving its goal of reading the keys below parentKey. It is only guaranteed that below parentKey the KeySet ks correctly represents the state of the KDB. For the rest of ks there are no such guarantees.

> **Note:** In the list below "phase" always refers to a phase of the get operation as described in the backend plugin contract.

The flow of this operation is:

1. Determine the backends needed to read all keys below parentKey
2. Run the open operation for all required backends that haven't been opened
3. Run the init phase on all the backends that haven't been initialized
4. Run the resolver phase on all backends
5. From now on ignore all backends, which indicated that there is no update.
6. If all backends are now ignored, **return**.
7. If a global cache plugin is enabled: Ask the global cache plugin for the cache handles (normally modification times) for all backends.
8. If all backends have an existing cache entry: Run the cachecheck phase on all backends
9. If all backends indicated the cache is still valid: Ask the global cache plugin for the cached data and **return**.
10. Run the prestorage and storage phase on all backends.
11. Run the poststorage phase of all spec:/ backends.
12. Merge the data from all backends
13. If enabled, run the gopts/get hook.
14. Run the spec/copy hook.
15. Split data back into individual backends.
16. Run the poststorage phase for all non-spec:/ backends.
17. Remove all keys which are below the parent key of any backend that has been read from ks.
18. Merge the data from all backends into ks.
19. If a global cache plugin is enabled, update cache.
20. Run the notification/send hook. Then **return**.

> **Note:** In case of error, we abort immediately, restore ks to its original state and return.

Influence of namespaces:

- spec:/ backends go through init, resolver, cache, presetstorage and storage phases as normal, but their poststorage phase is called earlier. This is required, because any validation and post-processing of spec:/ keys needs to happen, before they are used as the specification for other keys in the actual poststorage phase.
- dir:/, user:/ and system:/ go through all phases as described above.
- proc:/ mountpoints go through all the phases as described above, but they are not stored in the cache.
- default:/ backends only go through the poststorage phase. This is because default:/ keys are generated from the specification (stored as spec:/ keys). Therefore, no default:/ keys can exist before the specification is processed by the spec/copy hook.
- keys with other namespaces are always illegal in ks

301.3 <tt>set</tt> Operation

The purpose of the `set` operation is to write data from a `KDB` instance into backends.

> **Note:** Some details of a `set` operation are defined in the contract with backend plugins.

Properties of `kdbSet()`:

- When calling `kdbSet (kdb, ks, parentKey)` the contents (key names, values and metadata) of `ks` will *mostly* not be modified. The only modifications that are made to `ks` are those that originate from the `spec/copy` hook.
- *All keys* in `ks` that are below `parentKey` will be persisted in the KDB, when a `kdbSet (kdb, ks, parentKey)` call returns successfully. Additionally, any key in `ks` that shares a backend with another key which is below `parentKey` will also be persisted.
- Calling `kdbSet` may result in an error, if `kdbGet` wasn't called on this `KDB` instance with the same `parentKey` at least once.
- After calling `kdbSet (kdb, ks, parentKey)`, the Key `parentKey` will only have the `meta:/error/*` or `meta:/warnings/#/*` metakeys, if the errors/warnings originate from this `kdbSet()` call. In other words, `kdbSet ()` first clears any existing errors/warnings and only then starts doing the actual work.
- It is an error to use a `parentKey` with a namespace other than: `default:/`, `proc:/`, `spec:`, `dir:/`, `user:/`, `system:/` or cascading

The flow of this operation is:

1. Determine the backends needed to write all keys below `parentKey`.
2. Check that all backends are opened and initialized (i.e. `kdbGet()` was called).
3. Run the `spec/copy` hook on `ks` (to add metakeys to newly created keys).
4. Deep-Copy `ks` (below `parentKey`) into a new KeySet `set_ks`
5. Split `set_ks` into individual backends
6. Determine which backends contain changed data. Any backend that contains a key that needs sync (via `KEY_FLAG_SYNC`) could contain changed data. From now on ignore all backends that have not changed. From now on also ignore all backends that were initialized as read-only. Issue a warning, if a change was detected (via `KEY_FLAG_SYNC`) in a read-only backend.

   > **Note**: Steps 4-6 might be combined into a single procedure that deep-copies only keys from changed backends into separate KeySets per backend

   7. Run the `resolver` and `prestorage` on all backends (abort immediately on error and go to e).
7. Merge the results into a new version of `set_ks`.
8. Run the `spec/remove` hook on `set_ks` (to remove copied metakeys).
9. Split `set_ks` into individual backends again.
10. Run the `storage` and `poststorage` phases on all backends (abort immediately on error and go to e).
11. If everything was successful: Run the `precommit` and `commit` phases on all backends (abort immediately on error and go to e), then run the `postcommit` phase on all backends (record all errors as warnings and ignore them) and **return**.

1. If there was an error: Run the `prerollback`, `rollback` and `postrollback` phases on all backends and **return**.

Influence of namespaces:

- `default:/` and `proc:/` keys are completely ignored by `kdbSet()`
- `spec:/`, `dir:/`, `user:/` and `system:/` go through all phases as described above.
- keys with other namespaces are always illegal in `ks`

301.4 <tt>close</tt> Operation

The `close` operation is very simple. It simply frees up all resources used by a `KDB` instance.

Chapter 302

Metadata

Make sure to first read elektra-metadata(7) to familiarize yourself with the *metadata* concept.
In Elektra, *metakeys* represent metadata. They can be stored in a key database, often within the representation of the `Key`. Metakey is a `Key` that is part of the data structure `Key`. It can be looked up using a **metaname**. Metanames are unique within a `Key`. Metakeys can also carry a value, called **metavalue**. It is possible to iterate over all metakeys of a `Key`, but it is impossible for a metakey to hold other metakeys recursively. The purpose of metakeys next to keys is to distinguish between configuration and information about settings.

302.1 Implementation

In this document, we discuss the implementation of metadata. Metakey is implemented directly in a `Key`: Every metakey belongs to a key **inseparable**. Unlike normal key names there is no absolute path for it in the hierarchy, but a relative one only valid within the key.
The advantage of embedding metadata into a key is that functions can operate on a key's metadata if a key is passed as a parameter. Because of this, `keyNew()` directly supports adding metadata. A key with metadata is self-contained. When the key is passed to a function, the metadata is always passed with it. Because of the tight integration into a `Key`, the metadata does not disturb the user.
A disadvantage of this approach is that storage plugins are more likely to ignore metadata because metakeys are distinct from keys and have to be handled separately. It is not possible to iterate over all keys and their metadata in a single loop. Instead only a nested loop provides full iteration over all keys and metakeys.
The metakey itself is also represented by a `Key`. So the data structure `Key` is nested directly into a `Key`. The reason for this is to make the concept easier for the user who already knows how to work with a `Key`. But even new users need to learn only one interface. During iteration the metakeys, represented through a `Key` object, contain both the metaname and the metavalue. The metaname is shorter than a key name because the name is unique only in the `Key` and not for the whole global configuration.
The implementation adds no significant memory overhead per `Key` if no metadata is used. For embedded systems it is useful to have keys without metadata. Special plugins can help for systems that have very limited memory capacity. Also for systems with enough memory we should consider that adding the first metadata to a key has some additional overhead. In the current implementation a new `KeySet` is allocated in this situation.

302.1.1 Interface

The interface to access metadata consists of the following functions:
Interface of metadata:

```
const Key * keyGetMeta (const Key * key, const char * metaName);
ssize_t keySetMeta (Key * key, const char * metaName, const char *newMetaString);
KeySet * keyMeta (Key * key);
```

Inside a `Key`, metadata with a given metaname and a metavalue can be set using `keySetMeta()` and retrieved using `keyGetMeta()`.
With `keyMeta()` you can get a `KeySet` with all metakeys of a given `Key`. It's possible to iterate the returned metadata `KeySet` like any other `KeySet`.
Example for iterating metadata:

```
void printMetaData (Key * key)
{
    Key * curMeta;
    KeySet metaKeys = keyMeta (key);
```

```
    ssize_t ksSize = ksGetSize (metaKeys);
    for (elektraCursor it = 0; it < ksSize; ++it)
    {
        curMeta = ksAtCursor (metaKeys, it);
        printf ("metaname: %s, metavalue: %s\n", keyName (curMeta), keyString (curMeta));
    }
}
```

Developers used to Elektra will immediately be familiar with the interface. Tiny wrapper functions still support the old metadata interface.

302.1.2 Sharing of Metakey

Usually substantial amounts of metadata are shared between keys. For example, many keys have the type `int`. To avoid the problem that every key with this metadata occupies additional space, `keyCopyMeta()` was invented. It copies metadata from one key to another. Only one metakey resides in memory as long as the metadata is not changed with `keySetMeta()`. To copy metadata, the following functions can be used:

```
int keyCopyMeta(Key *dest, const Key *source, const char *metaName);
int keyCopyAllMeta(Key *dest, const Key *source);
```

The name `copy` is used because the information is copied from one key to another. It has the same meaning as in `ksCopy()`. In both cases it is a flat copy. `keyCopyAllMeta()` copies all metadata from one key to another. It is more efficient than a loop with the same effect.

`keyDup()` copies all metadata as expected. Sharing metadata makes no difference from the user's point of view. Whenever a metavalue is changed a new metakey is generated. It does not matter if the old metakey was shared or not. This is the reason why a const pointer is always passed back. The metakey must not be changed because it can be used within another key.

302.2 See also

- `all available metadata` in Elektra (defines SpecElektra)
- glossary

Chapter 303

New mountpoint config structure

A mountpoint is defined by adding keys below `system:/elekta/mountpoints/<mountpoint>`, where `<mountpoint>` is the key name for the parent key of the mountpoint with slashes properly escaped. For example for the mountpoint `user:/mymountpoint` you have to add keys below `system:/elekta/mountpoints/user:\/mymountpoint`.
Every mountpoint consists of four parts:

1. A set of plugins required for the mountpoint.
2. A single *backend plugin* within the set of plugins.
3. The *mountpoint definition* specific to the backend plugin.
4. The *mountpoint config* available to all plugins of this mountpoint.

The set of plugins is defined below `system:/elektra/mountpoints/<mountpoint>/plugins`. The keys below `system:/elektra/mountpoints/<mountpoint>/plugins/<ref>` define a single instance of a plugin. The part `<ref>` will be used as name by which this plugin instance will be referenced later. The name may be arbitrary, but it must not be `backend`. The name `backend` is reserved for the backend plugin (see below).
To define a plugin instance, `system:/elektra/mountpoints/<mountpoint>/plugins/<ref>/name` must be set to the name of the plugin, i.e., it must be possible to open the plugin with `elektraPluginOpen` via this value. Optionally, the configuration keyset can be defined by the keys below `system:/elektra/mountpoints/<mountpoint>/plugins/<ref>/config`.
The single *backend plugin* is the only one that the `libelektra-kdb` library communicates with. It is responsible for calling other plugins when needed. A *backend contract* exists between this plugin and `libelektra-kdb`. The backend plugin alone is responsible for upholding this contract and enforcing it upon other plugins it calls.
The backend plugin is defined (like the other plugins) below `system:/elektra/mountpoints/<mountpoint>/plugins`.
Specifically, it uses `backend` as `<ref>` and is therefore defined by the keys below `system:/elektra/mountpoints/<mour`
The *mountpoint definition* for the backend plugin is defined by the keys below `system:/elektra/mountpoints/<mountpoir`
The *mountpoint config* is defined by the keys below `system:/elektra/mountpoints/<mountpoint>/config`.
It is merged into the configuration keyset of every plugin of this mountpoint (including the backend plugin). Specifically, the mountpoint config is put into the `user:/` namespace, while the config from `system:/elektra/mountpoints/<mountpoint>/plugins/<ref>/config` is put into the `dir:/` namespace of the keyset passed to a plugin.
Additionally, the `config/needs` config from plugins' contract is moved to `system:/`. This config is not stored in `system:/elektra/mountpoints` and instead loaded at runtime by `libelektra-kdb` during the `open` operation.
This leaves the `default:/` namespace for plugins to add their own defaults before calling `ksLookup` to access the config keyset.

> **Note**: This *mountpoint definition* is separate from the normal configuration keyset passed to a plugin.
> The backend plugin may still have such a keyset below its `system:/elektra/mountpoints/<mountpoint>/plugi` key. The difference between these two keyset is that the configuration keyset should be used to define the general operation of the plugin, while the mountpoint definition should specify a mountpoint.

For example, a backend plugin could support a logging mode, in which it produces a log entry every time it calls another plugin. This should be configured in the configuration keyset not in the mountpoint definition, since it doesn't change how the mountpoint works.

Additionally, the config of a plugin is *always* processed for *all* mountpoints, no matter if the mountpoint is actually accessed or the plugin actually used. This is because config is processed during the `open` operation, which doesn't know what parts of the whole KDB are needed for the current application. The mountpoint definition meanwhile is only processed for the mountpoints that are actually accessed. In this case the processing happens during the first `get` operation (within an application) that uses this mountpoint.

All other keys below `system:/elektra/mountpoints/<mountpoint>` are ignored.
To illustrate this structure, here is an example configuration:

```
# List of plugins with their config
system:/elektra/mountpoints/\/hosts/plugins/resolver/name (="resolver_fm_hpu_b")
system:/elektra/mountpoints/\/hosts/plugins/myglob/name (="glob")
system:/elektra/mountpoints/\/hosts/plugins/myglob/config/set/#0
system:/elektra/mountpoints/\/hosts/plugins/myglob/config/set/#1
system:/elektra/mountpoints/\/hosts/plugins/myglob/config/set/#2
system:/elektra/mountpoints/\/hosts/plugins/myglob/config/set/#3
system:/elektra/mountpoints/\/hosts/plugins/myglob/config/set/#4/flags
system:/elektra/mountpoints/\/hosts/plugins/store/name (="hosts")
system:/elektra/mountpoints/\/hosts/plugins/sync/name (="sync")
system:/elektra/mountpoints/\/hosts/plugins/error/name (="error")
system:/elektra/mountpoints/\/hosts/plugins/network/name (="network")
## part of the list of plugins, but also defines backend plugin
system:/elektra/mountpoints/\/hosts/plugins/backend/name (="backend")
# Configuration for backend plugin
system:/elektra/mountpoints/\/hosts/definition/path (="myhosts")
system:/elektra/mountpoints/\/hosts/definition/positions/get/resolver (="resolver")
system:/elektra/mountpoints/\/hosts/definition/positions/get/storage (="store")
system:/elektra/mountpoints/\/hosts/definition/positions/get/poststorage/#0 (="myglob")
system:/elektra/mountpoints/\/hosts/definition/positions/set/resolver (="resolver")
system:/elektra/mountpoints/\/hosts/definition/positions/set/prestorage/#0 (="myglob")
system:/elektra/mountpoints/\/hosts/definition/positions/set/prestorage/#1 (="error")
system:/elektra/mountpoints/\/hosts/definition/positions/set/prestorage/#2 (="network")
system:/elektra/mountpoints/\/hosts/definition/positions/set/storage (="store")
system:/elektra/mountpoints/\/hosts/definition/positions/set/precommit/#0 (="sync")
system:/elektra/mountpoints/\/hosts/definition/positions/set/commit (="resolver")
system:/elektra/mountpoints/\/hosts/definition/positions/set/rollback (="resolver")
# Everything else below system:/elektra/mountpoints/\/hosts is ignored
```

Explanation:

- This configures a mountpoint `/hosts` that loads the plugins `resolver_fm_hpu_b`, `glob`, `hosts`, `sync`, `error`, `network` and `backend`. The plugin `glob` also has some configuration keys defined.
- The `backend` plugin is set as the backend plugin.
- The rest is the mountpoint definition specific to the `backend` plugin.

As you can see from this example, the "positions" are part of the plugin-specific mountpoint definition. This allows different backend plugins to use different "positions".
For example, there is the `version` backend plugin that always just provides a few keys with version information for Elektra. It doesn't need to call any other plugins and requires no configuration at all. A mountpoint with this plugin could look like this:

```
system:/elektra/mountpoints/\/version/plugins/backend/name (="version")
```

Note: The above example, is also the minimal setup for a mountpoint. The key `system:/elektra/mountpoints/<mountpoint>/plugins/backend/name` is the only one that must be defined.

303.1 Operations and Phases

See KDB Operations Documentation for a description of operations and phases.
In each of the phases of a `get` or `set` operation, the corresponding function of the backend plugin is called. For a description of how this works exactly read the Backend Plugins Documentation.
In the first example above, the phases were mapped one-to-one to what the plugin `backend` called "positions". The different terms are very much intentional, since this not a requirement.

303.2 Further Examples

We already had two examples above. Here we will look at a few more.

```
# List of plugins with their config
system:/elektra/mountpoints/\/hosts/plugins/resolver/name (="resolver_fm_hpu_b")
system:/elektra/mountpoints/\/hosts/plugins/glob/name (="glob")
system:/elektra/mountpoints/\/hosts/plugins/glob/config/set/#0
system:/elektra/mountpoints/\/hosts/plugins/glob/config/set/#1
system:/elektra/mountpoints/\/hosts/plugins/glob/config/set/#2
system:/elektra/mountpoints/\/hosts/plugins/glob/config/set/#3
system:/elektra/mountpoints/\/hosts/plugins/glob/config/set/#4/flags
system:/elektra/mountpoints/\/hosts/plugins/hosts/name (="hosts")
system:/elektra/mountpoints/\/hosts/plugins/sync/name (="sync")
system:/elektra/mountpoints/\/hosts/plugins/network/name (="network")
# Define backend plugin
system:/elektra/mountpoints/\/hosts/backend/name (="other_backend")
# Configuration for backend plugin
system:/elektra/mountpoints/\/hosts/definition/path (="myhosts")
system:/elektra/mountpoints/\/hosts/definition/positions/get/resolver = (="resolver")
system:/elektra/mountpoints/\/hosts/definition/positions/get/storage = (="hosts")
system:/elektra/mountpoints/\/hosts/definition/positions/get/validation/#0 (="glob")
system:/elektra/mountpoints/\/hosts/definition/positions/set/resolver = (="resolver")
system:/elektra/mountpoints/\/hosts/definition/positions/set/validation/#0 = (="glob")
system:/elektra/mountpoints/\/hosts/definition/positions/set/validation/#1 = (="network")
system:/elektra/mountpoints/\/hosts/definition/positions/set/storage = (="hosts")
system:/elektra/mountpoints/\/hosts/definition/positions/set/precommit/#0 = (="sync")
system:/elektra/mountpoints/\/hosts/definition/positions/set/commit = (="resolver")
system:/elektra/mountpoints/\/hosts/definition/positions/set/rollback (="resolver")
```

This example is very similar to the first one, but the plugin `other_backend` doesn't use the `postorage` and `prestorage` "positions". Instead, there is a `validation` position that is (presumably) called in the appropriate phase. The plugin `other_backend` may also impose its own restrictions on plugins configured for the `validation` position. For example, it may define that such plugins must not generate, remove or modify keys and provide a different position for plugins that do so.

```
system:/elektra/mountpoints/\/hosts/plugins/network/name (="network")
system:/elektra/mountpoints/\/hosts/plugins/backend/name (="db_backend")
system:/elektra/mountpoints/\/hosts/definition/db/driver (="postgres")
system:/elektra/mountpoints/\/hosts/definition/db/server (="127.0.0.1")
system:/elektra/mountpoints/\/hosts/definition/db/port (="5432")
system:/elektra/mountpoints/\/hosts/definition/db/user (="admin")
system:/elektra/mountpoints/\/hosts/definition/db/password (="supersecret")
system:/elektra/mountpoints/\/hosts/definition/phases/set/prestorage/#0 (="network")
```

This example shows an entirely different type of backend. The hypothetical `db_backend` is backed by a database. In this case it is configured for a PostgreSQL database running on `127.0.0.1:5432` to which we connect as user `admin`.

We also configured the `network` plugin to run in the `prestorage` phase of the `set` operation. Which phases can be used and how they must be configured of course depends on `db_backend`.

```
system:/elektra/mountpoints/\/hosts/plugins/yajl/name (="yajl")
system:/elektra/mountpoints/\/hosts/plugins/backend/name (="http_backend")
system:/elektra/mountpoints/\/hosts/url (="https://api.ipify.org/?format=JSON")
system:/elektra/mountpoints/\/hosts/decoder (="yajl")
```

The hypothetical `http_backend` plugin is a read-only backend plugin. In the example above, it is configured to load the URL `https://api.ipify.org/?format=JSON` and use the `yajl` plugin to parse the result into a keyset. Both the HTTP request and the decoding would likely happen in the `storage` phase of the `get` operation. The `resolver` phase could perform an HTTP cache check, for example.

Chapter 304

Plugins Framework

The key database of Elektra basically passes a `KeySet` from plugin to plugin. Plugins can be chained or nested. The library `libelektra-kdb` only interacts with specially designed hook plugins as well as one special type of plugin called a *backend plugin*, usually with other plugins nested below. All other plugins must be called by these backend plugins. Backend plugins then usually implement some kind of plugin chaining.

304.1 Contract

In theory a plugin could do pretty much anything. In practice, however, plugins must meet certain expectations, described by the plugin's contract.
Every plugin should provide a full contract to give information how it will work with other plugins. Most parts of the contract are obligatory. Plugins cannot be loaded without this information. For example, plugins must provide the clause `infos/version`.

304.2 Plugin Instances

304.3 Operations

304.4 Plugin providers

Next to the backend plugins, already introduced before, there a few other common types of plugins that will be explained below.

304.4.1 Backend Plugins

Backend plugins are a special type of plugin. Compared to other plugins they have a more strict contract for what they may and may not do in certain situations. This is because they are the only plugins that are invoked by `libelektra-kdb`. Other plugins are normally called by a backend plugin, although any plugin may call another. To `libelektra-kdb` it doesn't matter what other plugins do, as long as the backend plugins hide unexpected behavior. In other words backend plugins must ensure that to the outside everything behaves according to the contract.
While any plugin could be used as backend plugin, only a select few will behave according to the contract for backend plugins. Therefore, it is important to know which plugins can and cannot be used as backend plugins.
For more information on the contract for backend plugins, take a look at the relevant documentation.
It is also important to note, that not every plugin will be compatible with every backend plugin. Some backend plugins may not support calling other plugins at all, while others may also have their own contracts for the plugins they call. For example, the default backend plugin called `backend` requires a *resolver plugin* and a *storage plugin* to function. Because these are very important types of plugins, they are also explained below.

304.4.2 Resolver Plugins

304.4.3 Storage Plugins

304.4.4 Validation Plugins

Chapter 305

Plugins Ordering

305.1 Outdated

Warning Many of the things described below are outdated. Parts of this document are still valid for the `backend` plugin.

305.2 Introduction

You should first read elektra-plugins to get an idea about plugins.
This document describes how elektra-plugins are ordered with elektra-backends(7).
Multiple plugins open up many ways in which they can be arranged. A simple way is to have one array with pointers to plugins. To store a `KeySet`, the first plugin starts off with the `KeySet` passed to it. The resulting `KeySet` is given to the next plugin. This is repeated for every plugin in the array.
To obtain a `KeySet`, the array of plugins is processed the other way round. An empty `KeySet` is passed to the last plugin. The resulting `KeySet` is handed over to the previous one until the first plugin is executed.
This approach has shown not to be powerful enough to express all use cases by counter-evidence. Logging should take place after the storage plugin performs its actions in both directions. It is not possible to do this with a *single array* processed in a way as described above.

305.3 Separated lists

Two individual lists (get+set list) of plugins solve the described problem. Instead of bidirectional processing of a single list two separate arrays are used. It turns out that error scenarios also make this approach unsuccessful. When a plugin fails, no other plugin must be executed later on because it might depend on the previous plugin to work correctly. In `kdbGet()`, this works well – the update process will be stopped. But during `kdbSet()`, changes to the file system must be reverted. Plugins can leave a lock, temporary file or journal. These resources need to be cleaned up properly.

305.4 Error List

So every backend additionally needs a third array that is executed in error scenarios during `kdbSet()`. Plugins responsible for the cleanup, rollback or error notification are inserted into it.
The resolver plugin requires this error list to do a proper rollback. Another use case is logging after a failure has happened.

305.5 Arrays of linked lists

A disadvantage of using arrays is that they contain a fixed amount of slots. This means that when a certain amount of slots is full, it is no longer possible to add more. Therefore, the plugin arrays are structured as arrays of linked lists.

Each plugin role is assigned its own slot in the array, which contains a linked list. Plugins fulfilling that role are added to the list. That way, much greater and more flexible plugin numbers can be added to a single role.

305.6 Placements

The ordering of plugins inside these three arrays is controlled by elektra-contracts(7). Each of the three arrays has one slot for each role. These slots have names to be referred to in the contract.
Here you see a table that contains all names:

Slot	Error	Get	Set
0	prerollback	getresolver	setresolver
1	rollback	pregetstorage	presetstorage
2	postrollback	getstorage	setstorage
3		postgetstorage	precommit
4			commit
5			postcommit

How the placement is influenced using `infos/placement`, `infos/ordering` and `infos/stacking` is described in `CONTRACT.ini`.

Chapter 306

README

This section contains the developer's manual. It complements the man pages found here.

306.0.1 Overview

- Architecture
- Classes
- Data structures
- Developers's Glossary
- Plugins Framework
- History

306.0.2 Concepts

- KDB Contracts
- Error Handling
- Error Message
- Error Categorization
- Metadata
- Symbol Versioning
- Iterators
- Mountpoints
- Backend Plugins
- Hook Plugins

306.0.3 Internals

- Algorithm
- Plugins Ordering
- KDB Operations

Chapter 307

Symbol Versioning

To enable backwards compatibility, while allowing the API to evolve, we use symbol versioning. This document focuses on the setup Elektra uses. For an explanation of symbol versioning itself, you may start here.

307.1 Infrastructure

The setup used is based on the one used by FreeBSD. The main script `version_gen.awk` is taken from their source (with small modifications to enable Alpine Linux compatibility).
The main script, is called by CMake during the build process. It parses the `symbols.map` files of all libraries (e.g. `/src/libs/elektra/symbols.map`) and combines them into a single file, which is then used by the compiler and the linker.

307.1.1 <tt>versions.def</tt>

All versions of Elektra symbols have to be declared in `versions.def`. Only the versions declared in this file can be used in the `symbols.map` files. No symbols should be declared in this file.
Each version corresponds to a release. After each release of Elektra, a new version must be added to `versions.def`. Only this version may be modified in the `symbols.map` files. Released version MUST NOT change. The newest and therefore unreleased version may change at any point before its release.
The `libelektraprivate_1.0` is special. It MUST always declare the newest version (e.g. `libelektra_0.9`) as its base and it is the only private version (i.e. there is will be no `libelektraprivate_1.1`). The purpose of the private version is to allow the use of internal symbols across compile units. For example `elektraAbort` is part of this version. It should be usable in all of Elektra's modules, but not outside. Currently there is no mechanism to enforce this, however, developers can manually check, if the use any of the symbols declared in `libelektraprivate_1.0`. Any of these symbols may change or removed at any time. No compatibility guarantee is given for these symbols.

307.2 Exporting new symbols

To add a new symbol (probably a function), simply add its name to the `symbols.map` file of the relevant library. Make sure to always add the symbol to the newest version only. Only symbols listed in one of the `symbols.map` files will be accessible by the linker. This means only these symbols can be used outside of their compilation unit. Only export symbols, if this is your intention.
For public API this is an obvious decision. All public API function should be added to newest version that exists at the time of their creation.
For internal API the question is not so obvious. If possible use static symbols. Static symbols cannot be exported via `symbols.map`, so the question is trivial. If the symbol shall be used in different source files (and therefore cannot be static), but only inside a single compilation unit (°), you don't need to add the symbol. If you are unsure, first try it without adding the symbol to `symbols.map` and only if it doesn't work add the symbol to `libelektraprivate_1.0`.
(°) compilation units mostly correspond to libraries

307.3 Updating a symbol

If a symbol is part of the public API (i.e. not in `libelektraprivate_1.0`) and you want to make changes that are ABI incompatible (e.g. changing the signature of a function), you have to update the symbol version.
To do this, simply add the symbol name to the newest version in `symbols.map` and remove it from the version it is currently part of. Each symbol may only be defined in one version. If the symbol is already declared in the latest version, you don't need to do anything, unreleased version may be modified at any point in time.
For additional clarity, you may leave a comment in the old version noting where the symbols was moved to.
To actually implement symbol versioning, a few code changes are necessary. Explaining this is much easier, when using a concrete example.
We will use the function `int foo (Key * k)` and assume the new version will have the signature `int foo (Key * k, int x)`. The old version shall be called `oldversion` and the new version shall be `newversion`.
The old function declaration

```
int foo (Key * k);
```

has to be replaced with the new declarations

```
int foo (Key * k, int x);
int ELEKTRA_SYMVER (foo, v1) (Key * k);
```

The second line declares an old version of `foo`. Note: `v1` may be replaced by any string that is a valid identifier. It is simply their to make the symbol name unique.
In the source files containing the definition of our function, we change the old definition

```
int foo (Key * k)
{
    // old code ...
}
```

to use the compatibility symbol

```
int ELEKTRA_SYMVER (foo, v1) (Key * k)
{
    // old code ...
}
ELEKTRA_SYMVER_DECLARE ("oldversion", foo, v1)
```

To actually enable backwards compatibility, we need to add some assembler `.symver` pseudo instructions. To make this easier, we have the helper macro `ELEKTRA_SYMVER_DECLARE`. We add these lines after the function declaration. Note: there MUST NOT be a `;` after the `ELEKTRA_SYMVER_DECLARE` otherwise there will be errors on systems that don't support symbol versioning
Then we simply write the definition for the new version, as if it was an entirely new function:

```
int foo (Key * k, int x)
{
    // new code ...
}
```

The current version of a symbol always uses the unversioned name of the symbol. That way we default to this version. Only older versions use a versioned name via `ELEKTRA_SYMVER`.

307.4 Removing a symbol

If a symbol becomes deprecated, it should NOT be removed from the `symbols.map` file. This would break backwards compatibility. Instead we remove the default implementation and only leave versions implemented via `ELEKTRA_SYMVER` and `ELEKTRA_SYMVER_DECLARE`. The existing default version must be converted into a versioned symbol for the last supported version.

Chapter 308

Doxygen

Doxygen is used to generate documentation of the API and Markdown files. The output formats can be:

- HTML sites (online documentation)
- Latex files (offline documentation)
- RTFs (MS-Word)
- PostScript files
- hyperlinked PDFs
- Unix man pages

When referencing Markdown files, the links between them will be resolved with the Markdown Link Converter.

308.1 Usage

To invoke the generation with Doxygen, the so-called Doxyfile is needed:

```
doxygen Doxyfile
```

Elektra's Doxyfile is located at <PROJECT_ROOT>/doc/Doxyfile. This file is used for the configuration of the whole documentation generation process. The documentation for the Doxyfile can either be found within the file or at the Doxygen Website.

308.2 Mermaid JS

Mermaid JS is used for generating diagrams within the API. The graphs and diagrams are stored as .mmd files and are located at <PROJECT_ROOT>/doc/images/mermaid.
Mermaid files can be included using the configured alias in the Doxyfile:

```
ALIASES += mermaid{1}="@htmlonly <div class=\"mermaid\"> ^^ @endhtmlonly @htmlinclude \"\1.mmd\" @htmlonly ^^
       </div> @endhtmlonly"
```

The alias shows that the new command mermaid needs one parameter. This parameter is the name of the Mermaid file **without** the file extension.
The command can be used inside the API documentation as follows:
In Markdown files there is currently no support for including Mermaid files within them easily. One solution is to write Mermaid code within Markdown:

````
````sMarkdown
graph TD;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
````s
````

The other solution would be to convert the Mermaid files to SVGs and including them automatically. For this task, Mermaid CLI could be used.
NOTE: Mermaid CLI is currently not integrated in Elektra.
For further information on Mermaid JS and Doxygen, this tutorial had been used to include Mermaid JS.

308.3 Working Locally

When working with Doxygen locally, some obstacles have to be overcome.
First of all, CMake variables are used all over the Doxyfile:

```
...
PROJECT_LOGO = @PROJECT_SOURCE_DIR@/doc/images/logo/logo_color_doxygen.svg
...
```

`@PROJECT_SOURCE_DIR@` is such a variable, they can be detected by the leading and ending @ sign. CMake replaces those variables with meaningful values within the make process of the project.
As developers do not tend to execute the make process each time before testing some changes in the Doxyfile, it is recommended to maintain a copy of the Doxyfile, a local version.
As the main goal is that Doxygen generates useful output, the local version needs to be adapted. This can either be done by searching and replacing variables in the local Doxyfile or using the Doxywizard:

```
doxywizard Doxyfile-local
```

The `Doxywizard` is a GUI that helps to set values in the Doxyfile. The benefit of using the GUI is that each possible command is explained and invalid values for commands are highlighted red. This helps to find those values that lead to exceptions.

NOTE: The local Doxyfile should never be integrated in the version control system. Each change made to the local Doxyfile needs to be integrated to the actual Doxyfile by **using the CMake variables** of the project.

Chapter 309

Get Started

This document is intended for developers who want to get started with developing with Elektra.

309.1 Skill requirements

- Operating system

  We recommend a Unix-based operating system to run Elektra (Linux, BSD, macOS) but `it's also possible to use Windows which is supported but not yet fully tested`.

- Using command-line interface and commands

  The easiest way to compile, install and use Elektra is by using the terminal. We will introduce the basic commands which you will need to run Elektra for the very first time. `It's also possible to use CLion`

- Basic knowledge about git

  `Git` is a distributed version control system to track changes of the source code in a project. We will use a single Git command to get the source code of Elektra.

- Basic knowledge about make/CMake

  Don't panic! `make` or `CMake` are used to generate an executable program from the code. If you are not used to these tools, it's not a problem, we will introduce them to you in later sections.

- We also need your skill set to improve Elektra

  You can contribute to Elektra to improve the source code, website, documentation, translation etc.

309.2 Software requirements

We need to install some basic tools to run Elektra: CMake, Git and essential build tools (make, gcc, and some standard Unix tools; alternatively `ninja` and `clang` are also supported but not described here). Depending on your Linux distribution use the following commands to install these tools:

```
sudo apt-get install cmake git build-essential
```

Or on RPM (Red Hat Package Manager) based systems (like Fedora, openSUSE, CentOS etc.):

```
sudo yum install -y cmake git gcc-c++
```

Or on macOS, most of the build tools can be obtained by installing `Xcode`. Other required tools may be installed using `brew`. First, install brew as described on their website. Then issue the following command to get CMake to complete the basic requirements:

```
brew install cmake git
```

309.3 Installation

If you meet all the software requirements you can get the source code of Elektra by using this command:

```
git clone https://github.com/ElektraInitiative/libelektra.git
```

Run the following commands to compile Elektra with non-experimental plugins where your system happens to fulfill the dependencies:

```
cd libelektra  #navigate to libelektra
mkdir build  && cd build  #create and navigate to the build directory
cmake ..  # watch output to see if everything needed is included
#  optionally run "ccmake .." to get an overview of the available build settings (needs cmake-curses-gui)
cmake --build . -- -j5
```

Optionally you can also run tests, see here for more information:

```
cmake --build . --target run_nokdbtests
```

With these commands you will be able to run the "Hello World!" example, but usually you will need to use some of the plugins, tools and bindings of Elektra. Please take a look at the more detailed compiling documentation. After you completed building Elektra on your own, you can execute these commands to install Elektra (please check the installation documentationfor the many available packages):

```
sudo make install
sudo ldconfig #optional: check installation documentation for more information
```

Installation documentation contains further information about available packages.

Optionally you can also run tests to verify the installed Elektra, see here for more information:

```
kdb run_nokdbtests
```

309.4 Hello first steps!

This section attempts to give a brief introduction on how to use the key database of Elektra. The key database is modified and queried by using the `kdb` tool.

Let's look into an example:

```
sudo kdb mount "hello.spec.ni" "spec:/example/hello" ni
```

This first command shows one of the core concepts: In Elektra you mount a file into the key database by specifying a mountpoint. In this example we mount the file `hello.spec.ni` that shall be our configuration specification to the mountpoint `/example/hello` in the `spec` namespace (which is the namespace used for specifications). We could also say that we persistently mount a new *backend*. The last parameter `ni` specifies that we use the ni plugin to write in the ni format (a specialised variant of INI) to the file. This can also be omitted in which case the less human-friendly dump plugin is used.

Since we did not denote an absolute file path it depends on the namespace where the actual file is stored. However, we can always retrieve the location of a file with the `file` subcommand, e.g., `kdb file "spec:/example/hello"`.

Now let's add some data to our specification:

```
sudo kdb set "spec:/example/hello/what" ""
```

As you see above we extended the the mountpoint `/example/hello` with `/what`. Our specification now expects (but not requires) a `what` key in the configuration. (If you want to actually require it you need to set the metakey `meta:/require`, but more about metakeys below.) Note that the value is empty since we just want to specify the key.

Now let's show the actual power of Elektra by setting metadata to our specification:

```
sudo kdb meta-set "spec:/example/hello/what" default World
```

The metakey `default` (having the value `World`) is now associated with the key `spec:/example/hello/what`. It should be noted that the previous setting of `what` is in this scenario redundant and could be omitted because this is done implicitly.

Now let's mount our specification:

```
sudo kdb meta-set "spec:/example/hello" mountpoint "hello.ni"
sudo kdb spec-mount "/example/hello" ni
```

As you see above, we first specify a mountpoint for our configuration (which is just metadata of the parent key `spec:/example/hello`). Afterwards we use the `spec-mount` subcommand to mount a new backend by a previously mounted specification. Note that there shall not be a namespace given, because `spec-mount` creates mountpoints for all namespaces (except `spec` of course).

Now let's enjoy some configuration magic:

```
kdb get "/example/hello/what"
```

We notice even if we did not set a value for the key `/example/hello/what` we still get `World` back since this is specified as default value. Therein lies the power of Elektra: We can set metadata (like types, RegEx or date validation, etc.) that gets handled by plugins.

If we want to set a custom value we can do it this way:

```
kdb set "user:/example/hello/what" "first few steps"
```

Now a `kdb get /example/hello/what` would return `first few steps`.

Note that when we set a value the namespace of the key should be specified explicitly. If we don't give a proper namespace (like in the get example above) a so-called cascading lookup is done. However, cascading lookups during `kdb set` only work, if there is already an existing value (i.e. there is no ambiguity which key we want to set).

309.5 Further Reading

Start with your very first Elektra application in C and follow these steps: Hello World!
Modify the `website` with the following tutorial. The website is build with angular and hosted with grunt.
Also take a look at the webui.

Chapter 310

Git

310.1 Cheat Sheet: Basic Git Commands

```
git add readme.md   # adds the changes of the file `readme.md` to the staging area
git add .           # adds all changes of files in the current directory (recursively) to the staging area
git add --all       # adds all changes of files in the repository to the staging area
git commit -a       # executes a commit that automatically stages all changed and deleted files before
git checkout -b     # Create a new local branch based on the current branch
git pull            # Pull changes from the current remote branch
git push            # Push changes to the current remote branch
git stash           # Puts all changes in an dirty wirkling directory aside
git stash apply     # Reapplies stashed changes on the current branch
```

310.2 Installation

- Windows
- macOS
- Linux

If you are uncomfortable with command line interfaces, there are also plenty of GUI options available.

310.3 Basic Configuration

To configure git, you can use the `git config` command. You can configure each project individually, just by running `git config <setting> <value>` anywhere within a checked out repository, or for all projects for a user on the local machine using `git config --global <setting> <value>`. We recommend you set the following configuration options, when using git:

```
git config merge.ff false # Git will always create a commit, when trying to merge
git config pull.rebase true # Automatically rebase when pulling
```

Also ensure that you have your username and e-mail configured, so your work can be attributed to you:

```
git config --global user.name "Your Name"
git config --global user.email "yourname@yourdomain.com"
```

310.4 The Commit Message

A commit message should have the following syntax: `component: short change description`
For a clean and meaningful log the commit message should fulfill the following:

- use imperative in the subject line
- the subject line should not be longer than 50 characters
- start the subject line with the module name (e.g. resolver:, cpp bindings:)
- separate subject from body with a blank line

- in the body describe in detail what you did, and possibly why
- metadata like "Fixes #123" should be kept at the bottom of the commit message and definitely not in the title

Most commits should have a longer description in the body.

310.4.1 GitHub: Attributing Co-Authors

If multiple people worked on a commit, GitHub, supports a nice way to represent this. You can attribute them by leaving two blank lines at the end and then add a single line of `Co-authored-by: NAME <NAME@EXAMPLE.COM>` for every other person that worked on it. More Details here

310.5 Remote Branches

To list all remote branches use:
```
git branch -a
```
To checkout a remote branch initially use:
```
git checkout -b <branchname> origin/<branchname>
```
Once you have done this, it will be a local branch, too. Following remote branches should exist:

```
master
```

This is the development branch. Please try to not work directly on it, but instead you should use feature branches. So the only commits on master should be non-fastforward merges from features branches. Commits on master should always compile, and all test cases should pass successfully. (see config option above)

310.5.1 Communication Methods

To communicate with remote branches, Git supports the file protocol (acessible through the local file system), http and ssh. We generally recommend ssh, as it is encrypted and authentication with ssh-keys is simple and secure.

310.5.1.1 GitHub: Adding/creating an SSH Key

GitHub supports both HTTP and SSH for communication. Using SSH, you can remove the annoyance of having to enter your password every time. To learn how to add an SSH-Key follow this guide provided by GitHub.

310.6 Local Branches

You should always make your own feature branch with:
```
git checkout -b <feature-branch-name>
```
On this branch it is not so important that every commit compiles or all test cases run.
To merge a branch use (no-fastforward):
```
git merge --no-ff <branchname>
```
If you already did some commits, but want them in a branch, you can do:
```
git branch foo
git reset HEAD^^  # for 2 commits back
git reset origin/master
git-ref-log # recover
```

310.7 Working with forks

We recommend you use your own fork of the main `libelektra` repository, if you want to contribute. For more information on creating a fork, please take a look at GitHub's tutorial.
Once you have set up and cloned your fork, you need to keep it in sync with the main repository. To do that, we recommend you never directly commit anything to your fork's `master` branch. You also need to add a remote for the main repository:
```
# We assume the remote for your fork is called `origin` (the default).
git remote add upstream https://github.com/ElektraInitiative/libelektra.git
```
When you want to sync changes from the main repository into your fork, you can use these commands:
```
git fetch upstream master:master
git push orign master:master
```

> **Note**: These commands work with any branch checked out. You don't need to switch to the `master` branch first. However, this only works, if you have not modified your `master` branch, i.e. the latest commit in your forked `master` branch was once the latest commit in the main `master` branch.

310.8 Rebasing branches

When you work on a local feature branch, it might happen that a change occurs within the master branch, that introduces merge conflicts. I.e. if you create a pull request, it cannot be merged into master automatically anymore. In these cases, we expect you to rebase your feature branch, so it can be merged automatically once again.
To achieve this, make sure you have your master branch synced up. Then switch to the feature branch you want to update using `git checkout <feature-branch-name>` and then type `git rebase master`. Sometimes, this will result in a merge conflict.
If you already have pushed changes from your feature branch to a remote branch, you'll need to either overwrite it using `git push --force` or push to a new remote branch.

310.8.1 Resolving merge conflicts

To resolve merge conflicts, edit the files that need to be merged manually. You can check which files need to be merged using `git status`. After you are done merging a file manually, you can use `git add <path-to-file>` and when you have merged all files, continue with `git rebase --continue`.

> **Note**: Keep in mind that manual merges within a rebase are potentially destructive. If you accidentally remove changes you've done in your feature branch, you might be able to recover them using `git-reflog`, but they could also be deleted by the `git-gc`. If you're new to manually merging files, consider creating a new branch from your feature branch using `git checkout -b <new-branch-name>` on which you can perform the merge without risk. When you're stuck on a manual merge, you can also always abort the rebase using `git rebase --abort`

310.9 Further resources

- `Git Book`
- `GitHub Docs`

Chapter 311

Goals

The goal of Elektra is to make it trivial to access and specify configuration settings by an API. This helps in achieving the following goals:

- Improve robustness of configuration systems by
  - Avoiding common programming errors through the usage of the API.
  - Getting more guarantees when accessing configuration settings.
  - Rejecting invalid configurations.
  - Avoiding reimplementation of parsers for the same configuration file format.
- Allow software to be better integrated via configuration settings (e.g. via `Freedesktop`).
- Postpone some decisions from programmers to maintainers/administrators:
  - Rejecting unwanted configuration settings.
  - Uniformity of configuration access (logging, vcs commit, notifications).
  - Syntax of the configuration files (with limitations, see below).

Elektra follows the goals below, in order of preference. If goals conflict, the higher goal takes precedence.

311.1 0. Stability

People need to be able to rely on the API/ABI to be stable. They expect their configuration settings to continue working, with minimal maintenance burden. In particular following parts must be immutable:

- The API: Within a major release, the core API can only be extended. Every application that compiled with Elektra `x.0.0` must still compile with any `x.y.z`. Even with new major releases, only small adoptions in the source of applications or plugins might be needed.
- The ABI: Even across major releases, the core ABI must stay compatible. Every application that links with Elektra `x.0.0` will continue to link with any future version of Elektra. We use symbol versioning for that goal.
- Key database and key names: Applications can rely on that whatever they once wrote into the key database, they will continue to get identical key names and values also with later versions of Elektra. Future extensions of the key database (e.g. new plugins) will not interfere.

311.2 1. Goal: Simplicity

Elektra is based on key-value pairs, the simplest form of what could be called a database. Elektra's key value pair uniformity allows with a single concept configuration settings and configuration specifications to be written and to be introspected.
An overly complex system cannot be managed nor understood. Extensibility brings some complex issues, which need to be solved – but in a way so that the user sees either nothing of it or only needs to understand very simple concepts. Special care for simplicity is taken for the users:

- Endusers when reconfiguring or upgrading should never take any notice of Elektra, except that it works more robust and is better integrated.
- Programmers should have multiple ways to take advantage of Elektra so that it flawlessly integrate with their system.
- Plugin Programmers: it should be simple to extend Elektra in any desired way.

311.3 2. Goal: Robustness

Configuration systems today suffer badly from:

- Different behavior on different systems.
- Missing or weak validation.
- Faulty transformations from strings to concrete types.
- No or misleading error messages.
- Undefined behavior.
- Not working migrations from one version to another.

We tackle this problem by introducing an abstraction layer where these problems are dealt with. The goal is that for improvements in these areas only code changes within Elektra are needed (and not within applications using Elektra). This makes the application's code not only portable towards more systems, but also enables global improvements in configuration systems.

311.4 3. Goal: Extensibility

There are many variants of

- Storage formats.
- Frontend integrations.
- Bindings.

Nearly every aspect of Elektra must be extremely extensible. On the other side semantics must be very clear and well-defined so that this extensible system works reproducible and predictable.
Only key-value pairs are the common factor and a way to get and set them, everything else is an extension.

311.5 4. Goal: Performance

Accessing configuration settings has impact on bootup and startup-time of applications. Elektra needs to have similar performance as current solutions. Of particular importance is that `kdbGet` invocations:

- are fast
- have low memory usage

The plugin system has as guideline:

> Only pay for what you need.

311.6 Users

These goals are about Elektra's users, again in order of importance. Again, lower goals need to be ignored if goals are in conflict.

311.6.1 1. Application Developers

Elektra must be easy and robust for application developers to store any configuration settings referable by keys they need to store. After writing configuration settings (`kdbSet`) and reading them again (`kdbGet`) they get the same KeySet (aka "round-trip").

> This means, they must be able to store keys with any name, any string or any binary data as needed for their purpose.

311.6.2 2. Administrators

Administrators should be empowered by good error messages and validation capabilities. Furthermore, they should be able to use their favorite tools and configuration file formats.

> There are principal limitations of nearly all configuration file formats, so Elektra cannot enable that any configuration file format can be used with any application. If maintainers or administrators want to change the configuration file format of some application, they need to carefully test if it works.

311.6.3 3. Maintainers

Elektra must be available everywhere and flexible enough, so that maintainers can integrate different applications by specifying and mounting.

> There might be some restrictions that some applications require some plugins to be mounted for their configuration settings.

311.6.4 4. Possibility to Represent any Configuration File Format

Elektra must be powerful and flexible enough to be able to represent any configuration file format. We support the development of fully-conforming parsers and emitters.

> This means, that given a correctly written storage plugin, a KeySet can be found that represents the configuration settings, its metadata and the hierarchical structure of the configuration file.

311.7 Non-Goals:

- Support semantics that do not fit into the KeySet (key-value pairs) with an `kdbGet()`/`kdbSet()` interface.
- Support for non-configuration issues, e.g., storing key-value data unrelated to configuration settings.
- Elektra is not a distributed configuration management tool: use your favorite configuration management tool on top or a distributed file system below Elektra.

311.7.1 Further Readings

- Continue reading: Who uses Elektra?

Chapter 312

elektra-backends(7) – the backend concept

Elektra has introduced **backends** to support the storage of key databases in different formats. Elektra abstracts configuration so that applications can receive and store settings without carrying information about how and where these are actually stored. It is the purpose of the backends to implement these details.
Since Elektra 0.8 a backend is composed of many plugins.

312.1 MULTIPLE PLUGINS

It's clear that too many features in one backend are problematic. It introduces unwanted external dependencies and leads to less portable backends. Many different aspects clutter the code making the backends unmaintainable. Features of other backends cannot be taken with ease because they are interwoven with other code.
To support the reuse of functionality, they must be useful in different situations. Separation of concerns is required for that. Each backend needs to have a specific purpose rather than implementing the full semantics of Elektra.
It was impossible to implement powerful and feature-rich backends so far because of the lack of modularity. Desirable features like notification and type checking have always been in the developers' heads, but there was no place where it would fit in without making the system unmaintainable, complex and full of unwanted external dependencies.
To solve this dilemma, Elektra uses **multiple plugins** together to build up a backend. The key set processed by one plugin will be passed to the next. This approach allows us to implement separate features in separate plugins. Plugins concerned with reading from and writing to permanent storage are called **storage plugins**. We cleaned existing backends from other tasks so that their only job now is related to the storage. Using this approach, the plugins provide the desired separation of concerns inside a backend.
Each plugin implements a single concrete requirement and it does that well. This architecture allows plugins to have external dependencies. Not every plugin has the burden to be portable anymore. That is no problem because the plugins are separate subprojects and maintainers can decide if they should be built for a specific platform or not. Afterward, users can choose which plugins they want to install and use. And finally, the administrator can choose which of the plugins should be loaded for each backend. If a specific feature is not needed, it is not included and does not cause additional overhead. Given the chosen approach, the core of Elektra can stay minimal. Most functionality can be moved to plugins.
Now let us look at the **development time** with multiple plugins. The programmer will find many reusable plugins. Some of them already fulfil given requirements. While checking the code quality of the plugins, the programmer actually learns how the plugin works. Given that point of view, the programmer will decide to use Elektra because he or she can choose from a pool of existing plugins.

312.2 SEE ALSO

- The tool for mounting a backend is kdb-mount(1)

Chapter 313

elektra-bootstrapping(7) – default backend

One important aspect of a configuration library is the out-of-the-box experience. How does the system work before anything is configured? The optimal situation is that everything fully works, and applications, that just want to load and store configuration, do not see any difference between out-of-the-box behavior and a well-configured, fine-tuned system.
To support that experience, a so-called **default backend** is responsible in the case that nothing was configured so far. It must have a storage that is able to store full Elektra semantics. To avoid reimplementation of storage plugins, a default storage plugin (`storage` or in code `KDB_STORAGE`) is used. A resolver plugin (`resolver` or in code `KDB_RESOLVER`) takes care of the inevitable portability issues. The **default backend** stores configuration in `KDB_DB_FILE`. One can easily avoid the usage of the default backend by simply mounting another backend to `/`.
The mounting configuration (the configuration how to mount the mount points) also needs to be stored somewhere. The so-called **init backend** is responsible for fetching configuration from `system:/elektra`, where the mount points are stored. Again `KDB_STORAGE` and `KDB_RESOLVER` is used, but now they write into the configuration file `KDB_DB_INIT` (`elektra.ecf` by default).
Thus for full and static build variants an exchange at run-time is not possible. Using shared libraries, however, `KDB_STORAGE` and `KDB_RESOLVER` are actually symbolic links (`libelektra-plugin-resolver.so` and `libelektra-plugin-storage.so`) to concrete plugins and thus can be changed without recompilation.
The **init backend** is guaranteed to stay mounted at `system:/elektra` where the configuration for Elektra itself is stored. After mounting all backends, Elektra checks if `system:/elektra` still resides at the default backend. If not, the init backend will be mounted there.

313.1 SUMMARY

To summarize, this approach delivers a good out-of-the-box experience capable of storing configuration. For special use cases, applications and administrators can mount specific backends anywhere except at, and below, `system:/elektra`. On `kdbOpen()`, the system bootstraps itself starting with the init backend.
The default backend consists of a default storage plugin and default resolver plugin. The default resolver has no specific requirements, but the default storage plugin must be able to handle full Elektra semantics. The backend is mounted to root (`/`), so any keys can be stored in it. The implementation of the core guarantees that user and system keys always stay separated.

313.2 TRACEABILITY

- `elektraOpenBootstrap()` implements above algorithm
- `backendOpenDefault()` opens the default backend
- `/src/include/kdbconfig.h.in` contains above `KDB_*` variables
- `src/plugins/CMakeLists.txt` creates the symbolic links
- `scripts/cmake/Modules/LibAddMacros.cmake` `create_lib_symlink` function

313.3 SEE ALSO

- bootstrap decision

Chapter 314

elektra-cascading(7) – of key names

Cascading is the triggering of secondary actions. For configuration it means that first the user configuration is read and if this attempt fails, the system configuration is used as fallback.
The idea is that the application installs a configuration storage with default settings that can only be changed by the administrator. But every user has the possibility to override parts of this *system configuration* regarding the user's needs in the *user configuration*. To sum up, besides system configuration, users have their own key databases that can override the settings according to their preferences.
Thus when a key starts with `/` such cascading will automatically be performed.

314.1 SPEC

Keys in `spec` allow us to specify which keys are read by the application, which fallback it might have and which is the default value using metadata. The implementation of these features happened in `ksLookup`. When cascading keys (those starting with `/`) are used following features are available (in the metadata of respective `spec`-keys):

- `override/#`: use these keys *in favor* of the key itself (note that `#` is the syntax for arrays, e.g. `#0` for the first element, `#_10` for the 11th and so on)
- `namespace/#`: instead of using all namespaces in the predefined order, one can specify which namespaces should be searched in which order
- `fallback/#`: when no key was found in any of the (specified) namespaces the `fallback`-keys will be searched
- `default`: this value will be used if nothing else was found

They can be used like this:

```
kdb set /overrides/test "example override"
sudo kdb meta-set spec:/test override/#0 /overrides/test
```

314.2 CASCADING

When cascading keys (those starting with `/`) the lookup will work in the following way (it can be debugged with `kdb get -v`):

1. In the `spec`-key the `override/#` keys will be considered.
2. If, in the `spec`-key, a `namespace/#` exist, those namespaces will be used.
3. Otherwise, all namespaces will be considered, see here.
4. In the `spec`-key the `fallback/#` keys will be considered.
5. In the `spec`-key the `default` value will be returned.

See application integration for how to use cascading names in the context of applications.
Read more about namespaces.

Chapter 315

elektra-cmerge-strategies(7) – how to merge key sets

Elektra's merge library offers different strategies to resolve conflicts in merges without user interaction. Consequently, Elektra's tools all have access to the following merge strategies:

- `abort`: the merge will abort if any conflict happens and merge the 3 key sets together otherwise.
- `our`: This option forces conflicting keys to be auto-resolved cleanly by favoring `our`. Changes from the other key sets that do not conflict with the `our` version are reflected in the merge result. This works like the recursive merge strategy with the `ours` option of git.
- `their`: This is the opposite of `our`. The merge will use the `their` version when a conflict happens.

If no strategy is specified, the merge will default to the abort strategy.

Chapter 316

elektra-contracts(7) – contracts for plugins

Each plugin in a backend can cause run-time errors. Additionally, the chaining of the plugins can introduce further run-time errors. For example, a plugin can modify keys so that the next plugin cannot process these keys anymore. Or a plugin can omit changes to the keys that are required by the next plugin. To deal with such situations in a controlled way, each plugin exports a contract that describes the interaction with other parts of the backend. A `KeySet` contains the description.
Mounting of backends actually takes place at run-time, we will refer to it as mount-time. The time when applications access the key database, however, will be called run-time.
The contract checker revises contracts of plugins during the mount-time. Afterwards, at run-time, no such type errors can occur. `kdb mount` implements the contract checker. It can refuse to add a plugin to the backend because of a conflict or constraint. As long as not all contracts are satisfied `kdb mount` waits for more plugins to be attached.
The content of the contract is well-specified in `CONTRACT.ini`.
Plugins are the only place to export contracts, but they are not the only party. It is also possible that the administrator extends the requirements to every backend, for example, if notification is required. This implies that every backend has to provide a specific additional service that will be checked using contracts. Plugins are also involved in contracts with Elektra's core. This topic will be discussed in elektra-algorithm in which the algorithm used by Elektra's core is explained.

316.1 Assertions

Contracts as introduced by Meyer define preconditions and postconditions on routines. Assertions handle these conditions by checking them before entering or after leaving a routine. Because most of this can happen only at run time, it often leaves the user alone with an exception.
Elektra goes a step further. Instead of exiting the normal control flow when a precondition is not met, it is the responsibility of a special plugin to handle the situation and make sure that the precondition is met afterwards. Sometimes this is not possible. In these cases, the plugins check the necessary conditions and return with error code when they are not met. As we will see, these situations do not occur very often.
Because of the modular approach, we can have several checkers, and correcting and checking can be combined. Plugins can work together to reach a certain goal.
`KeySet` and `Key` already handle most parts of checking pre- and postconditions imposed on data structures. Elektra's core provides preconditions and weak postconditions for the plugins.

316.2 SEE ALSO

- `CONTRACT.ini`
- plugins-ordering

Chapter 317

Frequently Asked Questions

317.1 I Am Stuck. Where Can I Get Help?

`Please open an issue`. You can simply remove all template text and it is enough if the issue only contains your question. If you are unsure, `label it as question`.
Please do not waste too much time to find something out yourself. Information where people get stuck is valuable to improve Elektra and its documentation. Even if you find out directly after you posted the question: the pointer can be helpful for other people having the same problem.

317.2 Isn't it Easier to Implement a new Parser Than to use Elektra?

No, it is not. And the story does not end with implementing a configuration file parser. You also need:

- operating-system-specific code to locate configuration files
- tools to change the configuration files
- validation to make such changes user-friendly

Every successful project has implemented many features Elektra has. But Elektra has the distinctive advantage that you can pick the features as you need them. Not used plugins do not cause any overhead or dependency. If you need new plugins or bindings, there is a community which can help you. Furthermore, Elektra has a defined API and you can swap implementations as needed.
So it pays off to use Elektra – in the short and in the long term.

317.3 Why Do I Need Elektra If I Already Use Configuration Management Tools?

Short answer: Try

- `ansible-libelektra`
- `chef-libelektra`
- `puppet-libelektra`

to see how useful it can be.
Longer answer:
Elektra abstracts configuration settings, something desperately needed within configuration management. Instead of rewriting complete configuration files, which might create security problems due to ignoring distributions configuration files; Elektra operates precisely on the configuration setting you want to change: leaving others as chosen by the application or distribution. Furthermore, Elektra also allows us to *specify* configuration settings, which again brings benefits for configuration management tools.
Elektra is a radical step needed towards better configuration management: Let us fix how applications access configuration settings, so that we can properly access them, for example, from configuration management tools.
See our vision for an example.

317.4 If Elektra Already Exists so Long, why isn't it more Widespread?

There are two main reasons:

1. Research: First we needed to explore the design space. At that time, Elektra provided little benefit, except for niche applications. Then it was challenging to actually implement the vision.
2. Bootstrapping: Developers would like to have everything smooth and shiny, like packages available as are part of their distributions. But Elektra only gets packaged if there are already application using Elektra. We solved this problem in many intermediate steps, e.g., allow mounting legacy configuration files.

317.5 Do We Retain the Old Way of Configuring Things, i.e. Manually Editing an INI File in /etc?

Absolutely, you can think of libelektra as a small library in C that reads a configuration file and returns a data structure if you do not use any of its advanced features.
In fact, from the view of system-calls, a properly written configuration parser within your application would do exactly the same as Elektra does:

```
stat("/etc/kdb/elektra.ecf", {st_mode=S_IFREG|0644, st_size=1996, ...}) = 0
open("/etc/kdb/elektra.ecf", O_RDONLY)  = 3
read(3, "..., 8191) = 1996
close(3)                                = 0
```

Writing configuration files is much more tricky, as Elektra avoids data loss in the case of concurrent writes, even if the other application does not use Elektra. Elektra uses optimistic writes and rolls backs when it detects that configuration files were modified.

317.6 Do We Retain the Way of Reloading/Restarting the System Service?

Elektra does not interfere with restarting. It is a passive library. It provides some techniques for reloading but they are optional. We recommend, however, that you keep the in-memory and persistent configuration always in sync via immediate writes on changes and immediate reloading after notifications.

317.7 Is This an Actual Problem of Elektra or Is It Just Me?

In case of doubt please open an issue. If the question was already answered or is already in the documentation, we will simply point it out to you.
So do not worry too much, do not hesitate to ask any question. We welcome feedback, only then we can improve the documentation such as this FAQ!

317.8 What Should I Do If I Found a Bug?

Please check the issue tracker if it has already been reported. If it has not, please fill out the template. If you are in doubt, please report it.

317.9 How Can I Contribute to Elektra?

Due to the modular architecture we can accept many contributions as plugins. Please only make sure that the README.md clearly states the purpose and quality of the plugin.
Please start by reading here.

317.10 What Is Elektra's License?

New BSD license which allows us to have plugins link against GPL and GPL-incompatible libraries. If you compile Elektra, e.g., with GPL plugins, the result is GPL. We are reuse compliant.

317.11 Which version should I use?

If you already use 0.8, you might want to continue using it until 1.0 is released. As announced here, the 0.9 series introduces incompatible changes as needed, in particular cleanup for the 1.0 release is done.
For details of versioning principles, see here.

317.12 Who Are the Authors?

List of authors.

317.13 How Can I Advertise Elektra?

- If questions about configuration come up, point users to https://www.libelektra.org
- Display the SVG logos found at https://master.libelektra.org/doc/images/logo
- And already rastered logos at https://github.com/ElektraInitiative/blobs/tree/master/images/l
- Distribute the flyer found at https://github.com/ElektraInitiative/blobs/raw/master/flyers/flye odt
- And of course: talk about it!

Chapter 318

elektra-glossary(7) – glossary of Elektra

- **Configuration settings**: customize applications towards the users' needs. It fulfills following properties:
  - It is provided by the execution environment.
  - It can be changed by the maintainer, user, or system administrator of the software.
  - It consists of a key name, a **configuration value**, and potentially **metadata**.
- A **configuration file**: is a file containing configuration settings.
- **Configuration storage**: makes configuration settings persistent. The application will read the configuration at every start from the configuration storage, but it is only stored if a user changes settings.
- **Key databases**: are used for configuration storages because of these constraints. They can do fast key lookups and the keys can be structured hierarchically by defining separators in the key names.
- **Global key database**: provides global access to all configuration storages of all applications in a system. Abbreviated as `KDB`.
- **LibElektra**: is a set of libraries to access configuration parameters in a global, hierarchical key database.
- **SpecElektra**: is a `specification language` that allows us to describe the content of the global key database.
- **Elektra**: is a framework consisting of LibElektra, SpecElektra, and a collection of tools.
- To **elektrify** an application: to change the application so that it uses LibElektra afterwards.
- **Elektra Initiative**: is a community that develops LibElektra, expands SpecElektra, improves Elektra's tooling and helps to elektrify applications.
- **Option**, more specifically **Command-line option**: is a special argument passed on the command-line. **Short options** are single characters prefixed with '-'; **Long options** are arbitrarily long and start with '–'.
- **Module**: The parts Elektra is composed of, i.e. either lib, plugin, backend, tool.
- **Class**: A group of functions that logically belong together, working on the same type of objects. A library may implement several classes. E.g., `KDB`, `Key` and `KeySet` are the most important classes.
- **Hooks**: Central points in the KDB lifecycle, where specialized plugins are called. Can be used for notification and other purposes.

318.1 Technical Concepts

- Backends: A collection of **plugins** to be **mounted**. A **backend** typically is responsible to read and write a configuration file.
- Bootstrapping: To read the mounting configuration and mount during `kdbOpen()`.
- Cascading: To consider multiple places to look for a key.

- Contracts: Contracts state the purpose, functionality and requirements of **plugins**.
- Mounting: To persistently and permanently include a **backend** in the **global key database**. The **mountpoint** is the key where the backend is mounted to. All keys of the backend are below that key.
- Key name: All keys in the KDB have a name. This name is the keys unique identifier and follows a particular structure. For more information take look at the keyname documentation.
- Key name part: Key names consist of a series parts (and a namespace).
- Key base name: The last part of a key name.
- Key dir name: The key name obtained by omitting both namespace and base name from a key name.
- Namespaces: Allow us to have multiple keys for the same purpose and otherwise the same key name.
- Plugins: The unit of implementation for a feature.
- Metadata: Allows us to describe configuration settings.
- **persistent name/value/metadata**: How it is actually stored, i.e. the state returned by and passed to the `storage` plugins.
- **transient name/value/metadata**: How it is at runtime, i.e. what is returned by `kdbGet` and passed to `kdbSet`.
- **intermediate name/value/metadata**: Any state inbetween the two.

318.2 Session Recording Concepts

- **recording session**: A recording session is a period of time during which changes to the Key Database (KDB) are tracked and accumulated. It begins when recording starts and ends when recording stops. Throughout the recording session, all changes made to the KDB are recorded, including additions, modifications, and deletions of keys and their associated values.
- **part diff**: The changes made during a single `kdbGet` - `kdbSet` round-trip.
- **session diff**: The changes made during the entire recording session.

318.3 Details

- Null Value: The absence of a value, i.e. `keyValue (key) == NULL`.
- **pop**: Used in `ksPop()` and `KDB_O_POP` means to remove a key from a keyset.
- **delete**: or abbr. del, used in `keyDel()`, `ksDel()` and `KDB_O_DEL` means to free a key or keyset. The memory can be used for something else afterwards.
- **remove**: Means that the key-value information in the physical database will be removed permanently. Also used to describe removing a particular key from a keyset.

Chapter 319

elektra-granularity(7) – relation of keys to files

Keys of a backend can only be retrieved as a full key set. Currently, it is not possible to fetch a part of the keys of a backend. So the user needs to cut out the interesting keys with `ksCut()` afterwards. If the keys should be committed again, the whole key set must be preserved. Otherwise, the clipped keys will be removed permanently.
This restriction simplifies storage plugins while it does not limit the user. Other plugins would not be able to fulfil their purpose without the full `KeySet`. For example, a plugin that checks the availability and structure of all keys cannot work with a partial key set.
It is problematic to have too many keys in one backend. The applications would need memory for unnecessary configuration data. Instead, we recommend introducing several mount points to split up the keys into different backends. Splitting up key sets makes sense if any application requests only a part of the configuration. No benefits arise if every application requests all keys anyway.
Let us assume that many keys reside in `user:/sw` and an application only needs the keys in `user:/sw/org/app`. To save memory and get better startup-times for the application, a new backend can be mounted at `user:/sw/org/app`. On the other hand, every mounted backend causes a small run-time overhead in the overall configuration system.
The solution in Elektra is flexible, because the user decides the granularity. It is possible to mount a backend on every single key, so that every key can be requested for itself. If no backends are mounted, all keys reside in the default backend.
To sum up, Elektra's core searches for the nearest mount point and gets the configuration from there. It is possible that the user gets more configuration than requested. The user can decide by means of mounting how much configuration on specific requests are returned.

Chapter 320

elektra-hierarchy(7) – standard hierarchy

320.1 Integrated Mount Points

These mount points are always available.

320.1.1 system:/elektra/modules

Information about currently loaded modules.

320.1.2 system:/elektra/mountpoints

The mount points present in the system.

320.1.3 system:/elektra/version

Version information.

320.2 Info Mountpoints

Use `kdb mount-info` to mount these mount points.

320.2.1 system:/info/elektra/constants

Gives information about how Elektra was build.

320.2.2 system:/info/elektra/uname

System Information given with `uname`.

320.2.3 system:/info/elektra/desktop

System Information about currently running desktop.

320.2.4 system:/info/elektra/metadata

Gives information about which metadata is currently understood by Elektra.
`METADATA.ini` needs to be mounted there.

320.2.5 system:/info/elektra/contract

Gives information about clauses in plugin's contract that is currently understood.
`CONTRACT.ini` needs to be mounted there.

320.2.6 SEE ALSO

- see namespaces tutorial
- elektra-namespaces(7)

Chapter 321

elektra-highlevel-gen(7) – High-level API code-generation advanced features

This document focuses on the advanced features of the high-level API code-generator template. We assume you already familiarized yourself with the basic features explained in `kdb-gen-highlevel(1)`.

321.1 Configuration Options

The parameters that are relevant to the concepts described here are (for the rest see `kdb-gen-highlevel(1)`):

- `embeddedSpec`: allowed values: `full` (default), `defaults`, `none`
- `enumConv`: allowed values: `strcmp`, `switch`, `auto` (default)

Using `embeddedSpec` you can configure how much of the specification is embedded into your application. By default we use `full`. This means the full specification is embedded into your application's binary. Since this can drastically increase the size of the binary, you can also choose `defaults` or `none`. The `defaults` setting embeds a reduced version of the specification, which only contains the metadata required by `elektraOpen`. By setting `embeddedSpec=none` you can also remove this reduced specification.
The advantage of using `full` is that your application is contained in a single executable file. If you don't use `full`, the code-generator produces an additional `.spec.eqd` file and omits specload function (called `exit↩ ForSpecload` by default). This file contains the full specification in quickdump format. You can either mount it directly via `quickdump`, or if you want the features of `specload` use a `specload` configuration like this: `app=/usr/bin/cat args=#0 args/#0="path-to-spec-output-file"`.
Setting `embeddedSpec=none` is only recommended, if you must have the minimal binary size and you know what you are doing. In this case no defaults are passed to `elektraOpen` and defaults are only handled via the `spec` plugin. If the specification/configuration isn't mounted, the getter functions may fail.
The case of a misconfigured mountpoint will be detected automatically and reported as an error. It will cause the initialization function (by default named `loadConfiguration`) to fail, if the specification is not mounted at the expected mountpoint or if the specification was not `spec-mount`ed.

321.2 Enums

We support the mapping of a set of string values to a native C `enum`. To use this feature, you need to write your specification the same way that the enum part of the `type` plugin expects.

```
[myenum]
type=enum
check/enum=#_4
check/enum/#0=none
check/enum/#1=red
check/enum/#2=green
check/enum/#4=blue
default=blue
```

The above specification will generate the following C `enum`:

```
typedef enum
{
    ELEKTRA_ENUM_MYENUM_NONE = 0,
```

```
    ELEKTRA_ENUM_MYENUM_RED = 1,
    ELEKTRA_ENUM_MYENUM_GREEN = 2,
    ELEKTRA_ENUM_MYENUM_BLUE = 4,
} ElektraEnumMyenum;
```

As you can see the integer values of the different enum values are taken from the indices of the `check/enum/#` array. You may also use e.g. `gen/enum/#2/value=1 << 1` to set a different value. The `gen/enum/#/value` values are inserted literally into the C files, so the values must be valid C code. The name of the enum may be configured via `gen/enum/type`. If you want to use an existing `enum` and map its values to strings you can turn the generation of the enum off, by adding `gen/enum/create=0`. In this case you have to add a header that defines the `enum` or `typedef`s it, to the `headers` parameter of the code-generator invocation.

Like with any other key, the code-generator produces `static inline` getter and setter functions for the key. Since there are no generic functions for the conversion of the strings into the enum values, we also generate those:

```
ELEKTRA_KEY_TO_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
ELEKTRA_TO_STRING_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
ELEKTRA_TO_CONST_STRING_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
ELEKTRA_GET_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
ELEKTRA_GET_ARRAY_ELEMENT_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
ELEKTRA_SET_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
ELEKTRA_SET_ARRAY_ELEMENT_SIGNATURE (ElektraEnumMyenum, EnumMyenum);
```

These functions are not generated per key, but per enum type. If multiple keys use the same enum type (both need to define the full metadata, including the full set of values), we only generate one set of these functions.

The getter and setter functions won't be explained here, they work like any of the other getter and setter functions of the high-level API.

The other three functions are used to convert between the string values and the generated `enum`. You may find these useful in your application. You can call them via e.g. `ELEKTRA_KEY_TO (EnumMyenum)`. The difference between `ELEKTRA_TO_STRING` and `ELEKTRA_TO_CONST_STRING` is that the first returns a `char *` allocated via `elektraMalloc`, while the second returns a static `constchar *`.

Both `ELEKTRA_TO_STRING` and `ELEKTRA_TO_CONST_STRING` are always implemented via a straightforward `switch` statement. The implementation of `ELEKTRA_KEY_TO` on the other hand can be changed via the `enumConv` parameter. If you set `enumConv=strcmp`, we will generate a code analogous to:

```
if (strcmp (string, "none") == 0) { /* ... */ }
if (strcmp (string, "red") == 0) { /* ... */ }
if (strcmp (string, "green") == 0) { /* ... */ }
if (strcmp (string, "blue") == 0) { /* ... */ }
```

This code is not really optimal, since we really only need to look at the first character to determine the correct enum value. This is where `enumConv=switch` comes in. With this option, we generate a series of (nested, if necessary) switch/case statements:

```
switch (string[0])
{
case 'b': /* blue */
case 'g': /* green */
case 'n': /* none */
case 'r': /* red */
}
```

Of course this version also has its own problems. Take for example the enum with the values: `blue`, `blueish` and `brown`. With `enumConv=switch` this would generate the following code:

```
switch (string[0])
{
case 'b':
    switch (string[1])
    {
    case 'l':
        switch (string[2])
        {
        case 'u':
            switch (string[3])
            {
            case 'e':
                switch (string[4])
                {
                case 'i': /* blueish */
                }
                /* blue */
            }
            break;
        }
        break;
        case 'r': /* brown */
    }
    break;
}
```

This is already quite hard to read and `blueish` isn't even that long.

To provide a compromise between readability and performance, we default to `enumConv=auto`. This options

uses the switch version, if the depth is less than 3, and the `strcmp` version in all other cases. A depth of `n` means looking at the first `n` characters `string[0], string[1], ..., string[n-1]`. In other words a depth of `n` uses `n` switch statements.

321.3 Structs

The `highlevel` template also has support for structs. By setting `type = struct` on a key, you can enable the generation of a native C `struct` for the keys below it.
We will look at this simple example:

```
[mystruct]
type=struct
check/type=any
default=""
[mystruct/a]
type=string
default=""
[mystruct/b]
type=long
default=8
```

Note: That we set `check/type=any` and `default=""`. This is to avoid problems with the `type` plugin, which doesn't know about `struct`s.
The generated struct looks like this:

```
typedef struct ElektraStructMystruct
{
    const char * a;
    kdb_long_t b;
} ElektraStructMystruct;
```

Similar to enums, you can customise the generated struct via additional metadata:

- Metadata for the key with `type=struct`:
  - `gen/struct/type` can be used to set the name of the generated struct.
  - `gen/struct/create=0` disables the struct generation and only generates the accessor functions. Use this to use structs defined elsewhere. Don't forget to include the needed header in the `headers` parameter.
  - `gen/struct/alloc` (values `0`, `1`) sets whether the struct is *allocating*. This changes how the getter works and also has some other implications. By default structs are non-allocating.
  - `gen/struct/depth` sets at how many levels below the `type=struct` key, we will include in the generated struct. Note that keys ending in `/#` (i.e. array keys) count as one level above. So `mystruct/x/#` would be included with the default `gen/struct/depth=1`.
- Metadata for keys corresponding to fields of the struct:
  - `gen/struct/field` sets the name of the field in the generated struct.
  - `gen/struct/field/ignore=1` ignores this key during struct generation, i.e. we don't create a field for it.
  - `gen/array/sizefield` sets the name of the field used to store the size of arrays. Only useful on array keys. For example, by default the size of the array key `mystruct/x/#` is stored in `xSize`, while the array is accessed via the field `x`.

We will also generate getter and setter functions:

```
ELEKTRA_GET_SIGNATURE (ElektraStructMystruct *, StructMystruct);
// or ELEKTRA_GET_OUT_PTR_SIGNATURE (ElektraStructMystruct, StructMystruct);
ELEKTRA_GET_ARRAY_ELEMENT_SIGNATURE (ElektraStructMystruct *, StructMystruct);
// or ELEKTRA_GET_OUT_PTR_ARRAY_ELEMENT_SIGNATURE (ElektraStructMystruct, StructMystruct);
ELEKTRA_SET_SIGNATURE (const ElektraStructMystruct *, StructMystruct);
ELEKTRA_SET_ARRAY_ELEMENT_SIGNATURE (const ElektraStructMystruct *, StructMystruct);
```

The difference between `ELEKTRA_GET_SIGNATURE` and `ELEKTRA_GET_OUT_PTR_SIGNATURE` is explained in the next section. Both versions are called via `ELEKTRA_GET (...) (...)`.
Allocating structs also generate `ELEKTRA_STRUCT_FREE (/* struct name */)`, which is used to free the allocated memory.

321.3.1 Allocating vs. Non-Allocating

The main difference between allocating and non-allocating structs, is how their getter function works.
Allocating structs use a getter similar to the one primitive types, strings and enums use. It returns a pointer to a newly allocated struct, which has to be freed using the generated `ELEKTRA_STRUCT_FREE` function.
Non-allocating structs meanwhile use a different kind of getter declared via `ELEKTRA_GET_OUT_PTR_SIGNATURE` instead of `ELEKTRA_GET_SIGNATURE`. This version doesn't return a pointer, instead it takes a pointer to an existing struct and only sets its fields. This is why you have to use the convenience macros `elektraFillStruct` and `elektraFillStructV` for these structs.
Non-allocating structs are also more limited than their allocating counterparts. They do not support arrays or struct references. They also cannot be for unions. Their main advantage is that you can use non-allocating structs without (additional) `malloc`/`free`, by providing a stack allocated pointer to the getter function.

321.3.2 Struct references

Structs cannot be nested, but they can reference each other. This allows for complex and possibly recursive structures. Take for example:

```
[person/#]
type=struct
check/type=any
default=""
gen/struct/alloc=1
[person/#/name]
type=string
default=Max
[person/#/mother]
type=struct_ref
check/type=any
default=""
check/reference=recursive
check/reference/restrict=../../../person/#
[person/#/children/#]
type=struct_ref
check/type=any
default=""
[person/#/children]
default=""
check/reference=recursive
check/reference/restrict=../../../person/#
```

This results in a struct like this:

```
typedef struct ElektraStructPerson
{
    struct ElektraStructPerson * mother;
        kdb_long_long_t childrenSize;
        struct ElektraStructPerson ** children;
        const char * name;
} ElektraStructPerson;
```

As you can see an instance of `ElektraStructPerson` may reference different instances. To declare this we must add a key with `type=struct_ref`. We use the metakeys of the `reference` plugin (which should be mounted to validate reference) to define what struct we want to reference. We also again set `check/type=any` and `default=""` to please the `type` plugin.
Struct references are also supported as arrays, in which case the `check/reference` keys must be on a different key than the rest of the metadata, because of how the `reference` plugin works. The example above shows this with `person/#/children` and `person/#/children/#`.
If you access an element of the `person/#` array via the getter function, we will recursively read the references structs. Writing structs that contain struct references or setting `struct_ref` keys directly is not supported.
Struct references can also exist outside of structs and maybe accessed directly via the generated accessor functions. Please, be careful when handling struct references, since invalid references will cause fatal errors.

321.4 Unions

The most advanced feature of the code-generator are unions. Sometimes we want a reference inside a struct, but it is not always to the same struct. For example in a menu structure, we might have a list of entries that are either submenus or actual items that execute a command.

```
[menu/#]
type=struct
check/type=any
default=""
gen/struct/alloc=1
[menu/#/name]
```

```
type=string
default=""
[menu/#/entries/#]
type=struct_ref
check/type=any
default=""
gen/reference/discriminator/enum = MenuEntryType
gen/reference/discriminator/union = MenuEntry
gen/reference/restrict/#0/discriminator = item
gen/reference/restrict/#1/discriminator = menu
[menu/#/entries]
default=""
check/reference=recursive
check/reference/restrict=#1
check/reference/restrict/#0=@/menu/#
check/reference/restrict/#1=@/item/#
[menu/#/discriminator]
type = discriminator
check/type = enum
check/enum = #1
check/enum/#0 = item
check/enum/#1 = menu
gen/enum/type=MenuEntryType
default = menu
[item/#]
type=struct
check/type=any
default=""
gen/struct/alloc=1
[item/#/name]
type=string
default=""
[item/#/command]
type=string
default=""
[item/#/entries]
check/reference/restrict=
[item/#/discriminator]
type = discriminator
check/type = enum
check/enum = #1
check/enum/#0 = item
check/enum/#1 = menu
gen/enum/type=MenuEntryType
default = item
```

As you can see the unions feature requires quite a bit more setup. We will start with `menu/#/entries/#`. It is set to `type=struct_ref` like you would do for normal struct reference, but the accompanying `menu/#/entries` uses `check/reference/restrict` as an array. This tells the `reference` plugin that any of the given reference restrictions are allowed. Therefore we could be referencing one of several structs and the code-generator has to deal with that somehow.

To allow alternative references, we need to define `gen/reference/discriminator/union` and `gen/reference/discriminator/enum` on the key with `type=struct_ref`. The former of these defines the name of the native C `union` the code-generator creates:

```
typedef union {
    struct ElektraStructMenu * item;
    struct ElektraStructMenu * menu;
} MenuEntry;
```

The other required metakey defines which enum shall be used as a discriminator between the union values:

```
typedef enum {
    ELEKTRA_ENUM_MENU_ENTRY_TYPE_ITEM = 0,
    ELEKTRA_ENUM_MENU_ENTRY_TYPE_MENU = 1
} MenuEntryType;
```

Each of the possibly referenced structs must have a discriminator key. This key must be part of the struct, it must have `type=discriminator` and should have `check/type=enum`. All the discriminator keys must also set `gen/enum/type` to the same value as chosen for `gen/reference/discriminator/enum` and all of them have to define the same enum, via the `check/enum/#` array. The values also have to match the values of the `gen/reference/restrict/#/discriminator` metakeys on the `type=struct_ref` key.

The generated structs will then look like this:

```
typedef struct Menu
{
        const char * name;
        kdb_long_long_t entriesSize;
        MenuEntryType * entryTypes;
        MenuEntry * entries;
} Menu;
typedef struct Item
{
        const char * name;
        const cahr * command;
} Menu;
```

As you can see the discriminator field is excluded from the struct itself and stored in a separate array. We do generate getter and free functions for unions, but we don't recommend using them directly. There are no setter functions for unions, because they involve struct references.

Chapter 322

elektra-introduction(7) – an introduction to Elektra

Elektra is a library implementing access to a global key database. The **global key database** provides access to all configuration files found on a system. To elektrify an application means to change the application so that it uses Elektra afterwards.
Information on Elektra can be found on the website. For introduction in the terminology, make sure to read the glossary.

322.1 Motivation

322.1.1 Why Elektra?

Configurations settings are hierarchical data structures of keys, each consisting of a name and a value. They can be used to configure software for the user's needs. Because these settings stay the same across restarts of the program, they need to be stored permanently. In the beginning this was done with primitive text files. Possibilities to structure the text were added later.
Nearly every system developed its own way to read configuration settings. Some got a de facto standard for a desktop environment (kconfig, gconfig) or even an operating system (Windows Registry, Open Directory). But they have a common problem: they are bound to the platform for which they were developed.
That is where Elektra comes in to fill the gap. On the one hand, Elektra is not tied to any platform or operating system. On the other hand, Elektra is powerful enough to be useful immediately for what it is written for: to access configuration.
For further views see why Elektra

322.1.2 Why Is It Important?

The configuration files that represent key databases can have binary or humanly-readable formats. From the latter, an unmanageable number is established. Developers of programs tend to document the format of the configuration file extensively. The configuration file may give a special flavor to a specific program and users frequently need it.
Sometimes limitations in the configuration file even lead to rewrites of software. For example, inetd has a non-modular flat configuration file that is not extensible because of a limited number of rows. In order to extend its functionality, the program had to be rewritten with a new approach to configuration: xinetd emerged. Both of these projects are now almost defined by their configuration files giving them identity and separating them from each other. Elektra has introduced backend to support the storage of key databases in different formats.
Elektra abstracts configuration so that applications can receive and store settings without carrying information about how and where these are actually stored. It is the purpose of the backends to implement these details. What makes the difference is the situation that every program can access any configuration because of the abstraction. In the example of inetd, Elektra allows an elektrified inetd respective xinetd to store its configuration in /etc/inetd.conf respective /etc/xinetd.conf. Additionally, each other program interested in these preferences can access them in a uniform way.
To support a global key database, a mutual agreement on some level is needed. Elektra provides this common layer with its data structures. Each elektrified application lies on top of this abstraction layer and it can talk to each part of the global key database using the classes presented next.

322.1.3 SEE ALSO

- Get a big picture
- Start reading about command-line tools

Chapter 323

elektra-key-names(7) – the names of keys

Every `Key` object with the same name will receive the very same information from the global key database. The name locates a **unique key** in the key database. Key names are always absolute; so no parent or other information is needed. That makes a `Key` self-contained and independent both in memory and storage.
Every key name starts with a namespace, for example `user` or `system`. These prefixes spawn key hierarchies each.
The shared *system configuration* is identical for every user. It contains, for example, information about system daemons, network related preferences and default settings for software. These keys are created when software is installed, and removed when software is purged. Only the administrator can change system configuration.
Examples of valid system key names:

```
system:/
system:/hosts/hostname
system:/sw/apache/httpd/#0/current/num_processes
system:/sw/apps/abc/#0/current/default-setting
```

user configuration is empty until the user changes some preferences. User configuration affects only a single user. The user's settings can contain information about the user's environment, preferred applications and anything not useful for the rest of the system.
Examples of valid user key names:

```
user:/
user:/env/#1/LD_LIBRARY_PATH
user:/sw/apps/abc/#0/current/default-setting
user:/sw/kde/kicker/#0/current/preferred_applications/#0
```

The slash (`/`) separates key names and structures them hierarchically. If two keys start with the same key names, but one key name continues after a slash, this key is **below** the other and is called a *subkey*. For example `user:/sw/apps/abc/current` is a subkey of the key `user:/sw/apps`. The key is not directly below but, for example, `user:/sw/apps/abc` is. Various functions in `keytest` implement ways to determine the relationship between two keys.

323.1 Conventions

For computers Elektra would work without any conventions, because it is possible to rename keys with plugins and link and transform any key-value to any other key-value. Obviously, for humans such chaos would be confusing and harder to use, thus we encourage everyone to use the following conventions:

323.1.1 Arrays

If you want to denote an array, i.e. many unnamed subkeys, use the syntax `#0`, ..., `#_10`. Then simple string comparisons will yield correct results and the names are still very compact.

323.1.2 Application Base Name

As decided here, the key names of software-applications should always start with:

```
/sw/org/myapp/#0/current/name/full
```

- `sw` is for software, `hw` for hardware, `elektra` for internals
- `org` is a URL/organization name to avoid name clashes with other application names. Use only one part of the URL/organization, so e.g. `kde` is enough.

- `myapp` is the name of the most specific component that has its own configuration
- `#0` is the major version number of the configuration (to be incremented if you need to introduce incompatible changes). (Rationale: it is possible to start the old version of the app, using `/sw/org/myapp/#X`, where `X` refers to the previous version number.)
- `current` is the profile to be used. This is needed by administrators if they want to start up multiple applications with different configurations.

323.2 Further Recommendations

- Avoid having your applications root right under `system` or `user`. (rationale: it would make the hierarchy too flat.) See **Application Base Name** above.
- Avoid the usage of characters other than `/`, a-z and 0-9. (rationale: it would allow too many similar, confusing names.) (exceptions: if the user or a technology, decide about parts of the key name, this restriction does not apply, e.g. if the wlan essid is used as part of the key name)
- The only way to separate names is using `/` (no A-Z, no _, no whitespaces) (rationale: there are many different opinions about this topic and having a choice which separator to choose will certainly lead to inconsistencies)
- It is suggested to make your application look for default keys under `/sw/org/myapp/#X/%/` where `X` is a major version number, e.g. `#3` for the 4th version and `%` is a profile (`%` for default profile). This way, from a sysadmin perspective, it will be possible to copy the `system:/sw/myapp/#3/%/` tree to something like `system:/sw/myapp/#3/old/` and keep system clean and organized.

323.3 SEE ALSO

- see application integration tutorial
- see namespaces tutorial
- `key name source file` or `its rendered API documentation`
- elektra-namespaces(7)
- elektra-cascading(7)

Chapter 324

elektra-merge-strategies(7) – how to merge key sets

In elektra-tools a three-way merging was implemented. It can also use be used for two-way merging, e.g. for importing.
Note: For a two-way merge, the `ours` version of the keys is used in place of `base`

324.1 3-WAY

- `base`: The `base` KeySet is the original version of the KeySet.
- `ours`: The `ours` KeySet represents the user's current version of the KeySet. This KeySet differs from `base` for every key you changed.
- `theirs`: The `theirs` KeySet usually represents the default version of a KeySet (usually the package maintainer's version). This KeySet differs from `base` for every key someone has changed.

The three-way merge works by comparing the `ours` KeySet and the `theirs` KeySet to the `base` KeySet. By looking for differences in these KeySets, a new KeySet called `result` is created that represents a merge of these KeySets.

324.2 STRATEGIES

Currently the following strategies exist:

- preserve: Automerge only those keys where just one side deviates from base (default).
- ours: Whenever a conflict exists, use our version.
- theirs: Whenever a conflict exists, use their version.
- cut: Removes existing keys below the resultpath and replaces them with the merged keyset.
- unchanged: (EXPERIMENTAL, only for kdb-mount) Do not fail if the operation does not change anything.
- import: (DEPRECATED, avoid using it) Preserves existing keys in the resultpath if they do not exist in the merged keyset. If the key does exist in the merged keyset, it will be overwritten.

Chapter 325

elektra-metadata(7) – metadata

Metadata is data about data. In earlier versions of Elektra, there has been a limited number of metadata entries suited for `filesys`. For `filesys` this was efficient, but it was of limited use for every other backend. This situation has now changed fundamentally by introducing arbitrary metadata.

325.1 Rationale

Metadata has different purposes:

- Traditionally Elektra used metadata to carry file system semantics. The backend `filesys` stores file metadata (File metadata in POSIX is returned by `stat()`) in a *struct* with the same name. It contains a file type (directory, symbolic link, ...) as well as other metadata like uid, gid, owner, mode, atime, mtime and ctime. into the `Key` objects. This solution, however, only makes sense when each file shelters only one `Key` object.
- The metaname `binary` shows if a `Key` object contains binary data. Otherwise it has a null-terminated C string.
- An application can set and get a flag in `Key` objects.
- Comments and owner, together with the items above, were the only metadata possible before arbitrary metadata was introduced.
- Further metadata can hold information on how to check and validate keys using types or regular expressions. Additional constraints concerning the validity of values can be convenient. Maximum length, forbidden characters and a specified range are examples of further constraints.
- They can denote when the value has changed or can be informal comments about the content or the character set being used.
- They can express the information the user has about the key, for example, comments in different languages. Language specific information can be supported by simply adding a unique language code to the metaname.
- They can represent information to be used by storage plugins. Information can be stored as syntactic, semantic or additional information rather than text in the key database. This could be ordering or version information.
- They can be interpreted by plugins, which is the most important purpose of metadata. Nearly all kinds of metadata mentioned above can belong to this category.
- Metadata is used to pass error or warning information from plugins to the application. The application can decide to present it to the user. The information is uniquely identified by numerical codes. Metadata can also embed descriptive text specifying a reason for the error.
- Applications can remember something about keys in metadata. Such metadata generalizes the application-defined flag.
- A more advanced idea is to use metadata to generate forms in a programmatic way. While it is certainly possible to store the necessary expressive metadata, it is plenty of work to define the semantics needed to do that.

325.2 Usage

Every key-value pair can have an arbitrary number of metakeys with metavalues attached. Identical to keys, metakeys are unique, but only within its key they are attached to.
To create a metakey, use kdb-meta-set(1), to get metadata kdb-meta-get(1).
The preferred way to use metadata is to set all metadata in the `spec` namespace and let the `spec` plugin copy the metadata to all other namespaces.

Chapter 326

elektra-mounting(7) – mounting

Mounting is the process of integrating a backend that reads and writes a specific configuration file into the global key database. Mounting allows you to use different configuration files but also allows you to change the behavior of writing/reading keys to/from the global key database. For example, you need to mount if you want to:

- change the syntax of a configuration file,
- log every change of a configuration file,
- validate a configuration file on every access,
- change the representation (e.g. the date-format or booleans), and
- everything else plugins can do.

Mounting allegorises a common technique for virtual file systems. File systems on different partitions or devices can be added to the currently accessible file system. Mounting is typically used to access data from external media. A more advanced use case presents mounting a file system that is optimised for specific purposes, for example, one that can handle many small files well. Mounting also allows us to access data via network storage. As a result, mounting of file systems has proved to be extremely successful.
Mounting in Elektra specifically allows us to map a part of the global key database to be handled by a different storage. A difference to file systems is that key names express what file names express in a file system. And instead of file systems writing to block devices, backends writing to key databases are mounted into the global key database. Mounting allows multiple backends to deal with configuration at the same time. Each of them is responsible for its own subtree of the global key database.
Mounting works for file systems only if the file system below is accessible and a directory exists at the mount point. Elektra does not enforce such restrictions.
Note, that you cannot mount the same configuration file multiple times. You can, however, use the specification to link between configuration items which gives an impression of a *bind mount*, i.e. having the same configuration values on multiple places.

326.1 SEE ALSO

- See elektra-glossary(7)
- More information about elektra-backends(7)
- The tool for mounting plugins is kdb-mount(1)
- Back to main page if you started from there

Chapter 327

elektra-namespaces(7) – namespaces

327.1 INTRODUCTION

Every key in Elektra needs a unique name so that administrators can refer to them unambiguously. Sometimes, multiple keys denote the same configuration item from different sources, e.g.:

- by a commandline argument
- by a configuration file found relative to the current directory
- by a configuration file found relative to the home directory
- by a configuration file found below `/etc`

To allow such keys to exist in parallel, Elektra uses namespaces.
A namespace has the following properties:

- in-memory Keys start with one of the namespaces
- keys within a namespace are known to stem from a specific configuration source. For example files from the `user` namespace are from the users home directory, **even if** an absolute configuration file name was used.
- `ksLookup()` uses multiple namespaces in a specific default order unless specified otherwise (cascading lookup)

Following parts of Elektra source code are affected by namespaces:

- the key name validation in `keySetName()`
- `keyGetNamespace()` which enumerates all namespaces
- `_Backend` and `split.c` for correct distribution to plugins (note that not all namespaces actually are distributed to configuration files)
- `mount.c` for cascading and root backends
- and of course many unit tests

In the rest of this document all currently available namespaces in the default order are described.

327.2 spec

Unlike the other namespaces, the specification namespace does not contain values of the keys, but instead metadata as described in `METADATA.ini`.
When a cascading key is looked up, keys from the spec-namespace are the first to be searched. When a spec-key is found, the rest of the lookup will be done as specified, probably in a different order than the namespaces enlisted here.

Usually, the spec-keys do not directly contribute to the value, with one notable exception: the default value (metadata `default`, see in cascading below) might be used if every other way as specified in the spec-key failed.
Spec-keys typically include an explanation and description for the key itself (but not comments which are specific for individual keys).
The spec configuration files are below `CMAKE_INSTALL_PREFIX/KDB_DB_SPEC`.
spec is not part of cascading mounts, because the specifications often are written in different syntax than the configuration files.

327.3 proc

Derived from the process (e.g. by parsing `/proc/self` or by arguments passed from the main method):

- program name
- arguments
- environment

Keys in the namespace proc can not be stored by their nature. Thus they are ignored by `kdbGet` and `kdbSet`. They might be different for every invocation of an application.

327.4 dir

Keys from the namespace `dir` are derived from a directory special to the user starting/using the application, e.g.:

- the current working directory for project specific settings, e.g. `.git`
- the directory a server wants to present to the user, e.g. `.htaccess`

Note that Elektra only supports a single special directory per KDB instance. Start a new KDB instance if you need different special directories for different parts of your application. How to change the directory may be different dependent on the resolver, e.g. by using chdir or by setting the environment variable `PWD`.

327.5 user

On multi-user operating systems obviously every user wants her/his own configuration. The user configuration is located in the users home directory typically below the folder `KDB_DB_USER`. Other paths below the home directory are possible too (absolute path for resolver).
Note that Elektra only supports a user directory per KDB instance. Start a new KDB instance if you need different user configuration for different parts of your application. How to change the user may be different dependent on the resolver, e.g. by `seteuid()` or by environment variables like `HOME`, `USER`

327.6 system

The system configuration is the same for every chroot.
The configuration is typically located below `KDB_DB_SYSTEM`. Other absolute paths, e.g. below `/opt` or `/usr/local/etc` are possible too.

327.7 Cascading

Keys that are not in a namespace (i.e. start with an `/`) are called cascading keys. Cascading keys do not stem from a configuration source, but are used by applications to lookup a key in different namespaces. So, multiple keys can contribute to each cascading key name.
Cascading is the same as a name resolution and provides a namespace unification as described in `Versatility and Unix semantics in namespace unification`.
Keys without a namespace can not be stored by their nature. So they are transient: after a restart they are forgotten.
Keys of that namespace are only used by ksLookup when no other suitable key was found. So they have the lowest possible priority, even fallback keys are preferred.
Read more about cascading.

Chapter 328

elektra-related – related configuration systems

Settings and preferences are ubiquitous in every software. For this reason, a vast amount of software has emerged on this topic. Most of the other projects, however, do not devote their work exclusively to configuration.

328.1 Configuration Libraries

We have already mentioned that Elektra provides the features of a configuration library. There are countless libraries out there that parse and some even generate configuration files. But there is a big catch: These libraries do not provide an abstract view of configuration. They require programmers to open the files themselves or at least to remember the path to them. Such information is itself configuration and inherently operating system dependent. Therefore, it is not possible to give default values that just work.
Another feature mostly missing in these libraries is cascading. It is not difficult to implement, but disturbing if it is not available at all or does not work correctly.

328.2 Augeas

Augeas is not intended for applications themselves, so it only creates a new view to configuration files which is not necessarily the same as applications see it. Additionally, Augeas has a rather poor level of abstraction and many syntactical details must be reflected in the configuration tree. Nevertheless, Elektra provides an augeas plugin which allows parts of Elektra's global configuration tree to be implemented using lenses. Lenses are a promising technology, which allow mapping from and to configuration files to be specified with a single program.

328.3 Uniconf

A project that shares some goals with Elektra, but uses a different approach is *Uniconf*. Besides a stand-alone library it supports a daemon mode. With the daemon running, it has the disadvantage of protocol overhead and a single point of failure. On the other hand, Uniconf can also notify applications when configuration files change and can work in a distributed mode. The project does not solve the problem of how to configure the configuration system. So we still have to pass a so-called *moniker string* that contains the knowledge where to find the configuration. The plugin system is flexible, but does not depend on the key name. So it is not possible to use different plugins for different applications and still have a global key database.

328.4 Debconf

Debconf is a set of Debian-tools that separates user interaction from the configuration process for the system's configuration when packages are installed or upgraded. Debconf itself does not change the configuration files. Instead, the administrator is asked questions depending on template files using different frontends. These answers are then cached. The configuration changes themselves are applied by post-install scripts of the particular package. Exactly for this last step Elektra could yield much better results, because it can compare configuration on a per-key basis and thus handle conflicts in a much more fine-grained way.

328.5 Freedesktop.org

The freedesktop.org initiative unifies different desktops using the X Window System in various ways. The main focus, however, lies in KDE and Gnome integration. One of the most disturbing and still unresolved problems is configuration. Elektra intends to fill this gap in the future.
Already available is the so-called XDG Base Directory Specification. Perhaps it will define file formats in the future, but currently only the paths are specified. To do so, it introduces some environment variables that help to find the actual configuration. Only if they are unset or empty, a built-in fallback should be used. If desired, elektrified applications behave in a XDG conforming way. (See the configuration x for the resolver).

Chapter 329

elektra-semantics(7) – Semantics of KDB

The use of arbitrary metadata has extensive effects in Elektra's semantics. They become simpler and more suited to carry key value pairs. The semantics now gives us independence of the underlying file system. So none of the file system's restrictions apply anymore. No constraints on the length of a key name disturbs the user anymore. Additionally, key names can be arbitrarily deep nested. Depth is the number of unescaped `/` in the key name.
The directory concept is enforced by default. Keys can be created everywhere. Keys always can have a value. The only constraint is that key names are unique and occur in one of the namespaces. Every Key has an absolute name. There is no concept of relative names in Elektra's Keys except for metakeys belonging to a key. Every other Key is independent of each other. We just do not care if there is another key below or above the accessed one in the storage or not.
Some applications need specific structure in the keys. Plugins can introduce and enforce relationships between keys. They can implement a type system, check if holes are present and check the structure and interrelations. They may propagate the metadata and introduce inheritance. We see that plugins are able to add more semantics to Elektra.
There are no symbolic links, hard links, device files or anything else different from key value pairs. Again, most of these behaviors can be mimicked using metadata. Especially, links are available using the metadata `override` and `fallback`.
Hidden keys are not useful for Elektra. Instead comments or other metadata contain information about keys that is not considered belonging to the configuration. If hidden keys are desired, we can still write a plugin to filter specific keys out.

329.1 Problems

This section explains why using file system semantics for configuration is not a good idea.

329.1.1 filesys

`filesys` was the first backend. It implemented the principle that every key is represented by a single file. The key name was actually mapped to a file name and the value and the comment was written to that file.
If the backend `filesys` was the ideal solution, Elektra's API (application programming interface) would be of limited use. E.g.cascading, type checking and optional cross-cutting features would be missing. The storage problem itself and the location of a key in a key database would be solved. because well-established APIs for accessing files are available in every applicable programming language.
Elektra 0.7 already supported more than one backend, but `filesys` was the only backend implementing the full semantics.

329.1.2 Limitations of File Systems

Here we will discuss, why the file system's semantics are not well suited for configuration at all.
One file per key turned out to be inefficient because of the file system's practical limitations. In most file systems, a file needs about four kilobytes (Depends on the block size, four kilobytes is a common value often used as default.) of space, no matter how little content is in it. Thus the file system wastes 99.9% of the space if keys have a payload of four bytes. Additionally, every file allocates a file node, which might be limited, too. We can argue, however, that we can use a file system which does not have this problem.

Many additional restrictions occur for portable access. The file name length in POSIX is limited to fourteen characters. Additionally, issues with case sensitivity are likely. The common denominator for all file systems is a surprisingly small one. If, for example, the traditional FAT file system should be supported, file names are limited to eight characters and case insensitivity.
On the one hand, there are many file system features that are not needed for configuration. File systems have a strict hierarchy. It is not possible to create a file in a non-existing directory. We will refer to such a missing object as **hole**. File systems do not support such holes.
A single **root directory** is not a useful concept for configuration. Instead, the system configuration and each user configuration has its own root. These root keys themselves are typically not needed.
There is additional metadata of files which is typically not needed for configuration: atime, mtime, ctime, uid, gid and mode just to name a few. Additional file types, for example, device files, links, fifos and sockets, are not needed either. Features like sparse files are ridiculous for the small strings, that key values typically are.
On the other hand, there are many *features missing* in file systems that we need in a serious key database. Creating a whole hierarchy of files at once atomically is not possible. Ways to achieve this are currently academic and not portable. Directories cannot have any content next to the files below. Swap semantics are missing: it is not possible to rename a file without removing the target first.
To sum up, file systems are not suitable to hold configuration with one entry per file. Instead, they are perfectly suitable to hold larger pieces of information like configuration files.

Chapter 330

elektra-spec(7) – spec namespace

330.1 INTRODUCTION

spec is a special namespace that describes via metadata the semantics of individual keys.
Most importantly it:

1. describes which keys are of interest to the application
2. describes the metadata to be copied to every key
3. describes how the cascading lookup works
4. describes the mount points including the plugins needed for them

It is, however, not limited to this but can express any other key database semantics (new plugins might be necessary, though).

330.2 Application

The most simple use is to enlist all keys that will be used by an application and maybe give a description for them (we use ini syntax in this document):

```
[mykey]
[folder/anotherkey]
description = set this key if you want another behavior
```

So Keys in `spec` allow us to specify which keys are read by the application. The description key will be copied (referred to) to `folder/anotherkey` of any namespace, so that it can easily be accessed.

330.3 Cascading Lookup

Other features are directly implemented in `ksLookup`. When cascading keys (those starting with `/`) are used following features are now available (in the metadata of respective `spec`-keys):

- `override/#`: use these keys *in favor* of the key itself (note that `#` is the syntax for arrays, e.g. `#0` for the first element, `#_10` for the 11th and so on)
- `namespace/#`: instead of using all namespaces in the predefined order, one can specify which namespaces should be searched in which order
- `fallback/#`: when no key was found in any of the (specified) namespaces the `fallback`-keys will be searched
- `default`: this value will be used if nothing else was found

E.g.

```
[promise]
default=20
fallback/#0=/somewhere/else
namespace/#0=user
```

1. When this file is mounted to `spec:/sw/app/#0` we specify, that for the key `/sw/app/#0/promise` only the namespace `user` should be used.
2. If this key was not found, but `/somewhere/else` is present, we will use this key instead. The `fallback` technique is very powerful: it allows us to have (recursive) links between applications. In the example above, the application is tricked in receiving e.g. the key `user:/somewhere/else` when `promise` was not available.
3. The value `20` will be used as default, even if no configuration file is found.

Note that the fallback, override and cascading works on *key level*, and not like most other systems have implemented, on configuration *file level*.

330.4 Validation

You can tag any key using the `check` metadata so that it will be validated.
For example:

```
[folder/anotherkey]
check/validation = abc.*
check/validation/message = def does not start with abc
```

330.5 Mounting

In the spec namespace you can also specify mount points. First you need the metakey `mountpoint` and a configuration file name. Otherwise, it basically works in the same way as the contracts in plugins using `infos` and `config`:

```
[]
mountpoint=file.abc
config/plugin/code/escape = 40
config/plugin/lua#abc/script = abc_storage.lua
infos/author = Markus Raab
infos/needs = resolver_abc rename code lua#abc
infos/recommends = hexcode
```

330.6 SEE ALSO

- see application integration tutorial (towards end)
- see namespaces tutorial
- elektra-namespaces(7)
- elektra-cascading(7)

Chapter 331

kdb-backup – Make a backup of current KDB

331.1 SYNOPSIS

```
kdb backup
```

331.2 DESCRIPTION

This command will backup the `system`, `user` and `spec` configuration. Afterwards a timestamp, which can be used to restore everything, will be printed.
The backup will be done in the `/var/tmp` directory, so make sure the backup is not deleted.

331.3 EXAMPLES

```
kdb backup
#> kdb restore 1500000000
```

331.4 SEE ALSO

- kdb-stash(1)
- kdb-restore(1)

Chapter 332

kdb-basename(1) – Get the basename of a key name

332.1 SYNOPSIS

`kdb basename <key name>`
Where `key name` is the name of the key.

332.2 DESCRIPTION

This command is used to retrieve the basename of a key.

332.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 7: invalid key name passed, see kdb(1) for a list of return codes used by kdb.

332.4 OPTIONS

- `-n`, `--no-newline`: Suppress the newline at the end of the output.

332.5 EXAMPLES

```
kdb basename user:/key/subkey
#> subkey
kdb basename user:/key/subkey/
#> subkey
kdb basename /
#>
```

332.6 SEE ALSO

- See kdb-namespace(1) and kdb-dirname(1) for other namepart extraction operations.
- The Unix analogue [BASENAME(1)].
- To get keys in shell scripts, you can use kdb-sget(1).
- elektra-key-names(7) for an explanation of key names.

Chapter 333

kdb-cache(1) – Enable, disable or clear the cache

333.1 SYNOPSIS

```
kdb cache {enable,disable,default,clear}
```

333.2 DESCRIPTION

This command is used to enable or disable the cache and to revert to the default settings. The default settings will let the system decide whether to use the cache or not. The clear command will remove the generated cache files in a safe way.

333.3 LIMITATIONS

Caches are stored on a per-user basis, therefore the `clear` subcommand can only remove a user's cache files (i.e. not system wide).

333.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

333.5 EXAMPLES

```
# Backup-and-Restore: system:/elektra/cache
# Enable the cache
kdb cache enable
# Disable the cache
kdb cache disable
# Revert to defaults
kdb cache default
# Clear all generated cache files
kdb cache clear
```

Chapter 334

kdb-change-resolver-symlink – Changes the default resolver symlink

334.1 SYNOPSIS

`kdb change-resolver-symlink <resolver>`
Where `resolver` is the name of the new default resolver plugin.

334.2 DESCRIPTION

This command updates the symlink pointing to the default resolver if `resolver` is a valid resolver plugin.

334.3 EXAMPLES

Set default resolver plugin to resolver_fm_hpu_b:
`kdb change-resolver-symlink resolver_fm_hpu_b`

334.4 SEE ALSO

- kdb-change-storage-symlink(1)

Chapter 335

kdb-change-storage-symlink – Changes the default storage symlink

335.1 SYNOPSIS

`kdb change-storage-symlink <storage>`
Where `storage` is the name of the new default storage plugin.

335.2 DESCRIPTION

This command updates the symlink pointing to the default storage if `storage` is a valid storage plugin.

335.3 EXAMPLES

Set default storage plugin to ini:
`kdb change-storage-symlink ini`

335.4 SEE ALSO

- kdb-change-resolver-symlink(1)

Chapter 336

kdb-plugin-check-env-dep – Checks which mount points are influenced by environment variables

336.1 SYNOPSIS

```
kdb check-env-dep
```

336.2 DESCRIPTION

This command checks which mount point is influenced by changes to environment variables. `PATH` and `KDB` are skipped for reasons.

336.3 SEE ALSO

- kdb-mount-list-all-files(1)

Chapter 337

kdb-cmerge(1) – Three-way merge of KeySets

337.1 NAME

kdb-cmerge - Join three key sets together

337.2 SYNOPSIS

`kdb cmerge [OPTIONS] our their base result`

- ourpath: Path to the keyset to serve as `our`
- theirpath: path to the keyset to serve as `their`
- basepath: path to the `base` keyset
- resultpath: path without keys where the merged keyset will be saved

337.3 DESCRIPTION

`kdb cmerge` can incorporate changes from two modified versions (our and their) into a common preceding version (base) of a key set. This lets you merge the sets of changes represented by the two newer key sets. This is called a three-way merge between key sets.
On success the resulting keyset will be saved to mergepath.
On unresolved conflicts nothing will be changed.
This tool currently exists alongside `kdb merge` until it is completely ready to supersede it. At this moment, cmerge will be renamed to merge.

337.4 OPTIONS

The options of `kdb cmerge` are:

- `-f`, `--force`: overwrite existing keys in `result`
- `-v`, `--verbose`: give additional information

Strategies offer fine grained control over conflict handling. The option is:

- `-s <name>`, `--strategy <name>`: which is used to specify a strategy to use in case of a conflict

Strategies have their own man page which can be accessed with `man elektra-cmerge-strategies`.

337.5 RETURN VALUE

- 0: Successful.
- 11: An error happened.
- 12: A merge conflict occurred and merge strategy abort was set.

The result of the merge is stored in `result`.

337.6 THREE-WAY MERGE

You can think of the three-way merge as subtracting base from their and adding the result to our, or as merging into our the changes that would turn base into their. Thus, it behaves exactly as the GNU diff3 tool. These three versions of the KeySet are:

- `base`: The `base` KeySet is the original version of the key set.
- `our`: The `our` KeySet represents the user's current version of the KeySet.
  This KeySet differs from `base` for every key you changed.
- `their`: The `their` KeySet usually represents the default version of a KeySet (usually the package maintainer's version).
  This KeySet differs from `base` for every key someone has changed.

The three-way merge works by comparing the `our` KeySet and the `their` KeySet to the `base` KeySet. By looking for differences in these KeySets, a new KeySet called `result` is created that represents a merge of these KeySets.

337.7 CONFLICTS

Conflicts occur when a key has a different value in all three key sets or when only base differs. When all three values for a key differ, we call this an overlap. Different merge strategies exist to resolve those conflicts.

337.8 EXAMPLES

To complete a simple merge of three KeySets:

````ssh kdb set user:/base "A" #> Create a new key user:/base with string "A" kdb set user:/their "A" #> Create a new key user:/their with string "A" kdb set user:/our "B" #> Create a new key user:/our with string "B" kdb cmerge user:/our user:/their user:/base user:/result kdb get user:/result #>B
```

## 337.9 SEE ALSO

- elektra-cmerge-strategy(7)
- elektra-key-names(7) for an explanation of key names. ````s

# Chapter 338

# kdb-complete(1) – Show suggestions how to complete a given path

## 338.1 SYNOPSIS

`kdb complete [path]`
Where `path` is the path for which the user would like to receive completion suggestion. If `path` is not specified, it will show every possible completion. It's synonymous to calling `kdb complete ""`.

## 338.2 DESCRIPTION

Show suggestions how the current name could be completed. Suggestions will include existing key names, path segments of existing key names, namespaces and mount points. Additionally, the output will indicate whether the given path is a node or a leaf in the hierarchy of keys, nodes end with '/' as opposed to leaves. It will also work for cascading keys, and will additionally display a cascading key's namespace in the output to indicate from which namespace this suggestion originates from.

## 338.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-m`, `--min-depth <min-depth>`: Specify the minimum depth of completion suggestions (0 by default), exclusive and relative to the name to complete.
- `-M`, `--max-depth <max-depth>`: Specify the maximum depth of completion suggestions (unlimited by default, 1 to show only the next level), inclusive and relative to the name to complete.
- `-v`, `--verbose`: Give a more detailed output, showing the number of child nodes and the depth level. Prints additional information in case of errors/warnings.
- `-0`, `--null`: Use binary 0 termination.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 338.4 EXAMPLES

```
# Backup-and-Restore: user:/tests/complete/examples
# Create the keys we use for the examples
kdb set user:/tests/complete/examples/kdb-complete/level1 foo
kdb set user:/tests/complete/examples/kdb-complete/lvl1/lvl2 bar
```

```
kdb set user:/tests/complete/examples/kdb-complete/lvl1/lvl2/lvl3/lvl4/lvl5 fizz
kdb set user:/tests/complete/examples/kdb-complete/buzz fizzBuzz
kdb set user:/tests/complete/examples/kdb-complete/#1 asdf
kdb set user:/tests/complete/examples/kdb-complete/% nothing
# list suggestions for namespaces starting with us, only the current level
kdb complete us --max-depth=1
#> user:/
# list suggestions for namespaces starting with user, only the current level
kdb complete user --max-depth=1
#> user:/
# list suggestions for the namespace user, only the next level as it ends with /
# note the difference to the previous example, which uses no trailing /
kdb complete user:/ --max-depth=1
# STDOUT-REGEX: .+
# list all possible namespaces or mount points, only the current level
kdb complete --max-depth=1
# STDOUT-REGEX: .+
# list suggestions for /tests/complete/examples/kdb-complete, only the current level
kdb complete /tests/complete/examples/kdb-complete --max-depth=1
#> user:/tests/complete/examples/kdb-complete/
# list suggestions for /tests/complete/examples/kdb-complete/, only the next level
# again, note the difference to the previous example which has no trailing /
kdb complete /tests/complete/examples/kdb-complete/ --max-depth=1
#> user:/tests/complete/examples/kdb-complete/%
#> user:/tests/complete/examples/kdb-complete/#1
#> user:/tests/complete/examples/kdb-complete/buzz
#> user:/tests/complete/examples/kdb-complete/level1
#> user:/tests/complete/examples/kdb-complete/lvl1/
# list suggestions for /tests/complete/examples/kdb-complete which are minimum 2 levels
# away from that key, and maximum 4 levels away
kdb complete /tests/complete/examples/kdb-complete/ --min-depth=2 --max-depth=4
#> user:/tests/complete/examples/kdb-complete/lvl1/lvl2/lvl3/
#> user:/tests/complete/examples/kdb-complete/lvl1/lvl2/lvl3/lvl4/
kdb rm -r user:/tests/complete/examples/kdb-complete
```

## 338.5 SEE ALSO

- If you would only like to see existing keys under a given path, consider using the kdb-ls(1) command.
- elektra-key-names(7) for an explanation of key names.

# Chapter 339

# kdb-convert(1) – Convert configuration files using elektra

## 339.1 SYNOPSIS

```
kdb convert [<import-format>] [<export-format>] [<import-file>] [export-file]
```

## 339.2 DESCRIPTION

This command allows a user to convert a file from any format supported by Elektra to any other supported format.
This command relies on the functionality of Elektra but does not actually modify the key database in any way.
This command uses plugins to specify and convert between formats, it is only limited by the plugins available for Elektra.

## 339.3 USAGE

Where `import-format` is the format that the current configuration file is using, `export-format` is the format the user wishes to convert it to, `import-file` is the full path to the current configuration file, and `export-file` is where the converted configuration file should be saved.
If either `import-format` or `export-format` is not specified, the `storage` plugin will be used instead. The `storage` plugin can be configured at compile-time or changed by the link `libelektra-plugin-storage.so`. If either `import-file` or `export-file` are not specified, `stdin` and `stdout` are used respectively.

## 339.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 339.5 EXAMPLES

To convert an Elektra dump file to xml:
```
cat sw.ecf | kdb convert dump xmltool > sw.xml
```
Another way to convert an Elektra dump file to xml:
```
kdb convert dump xmltool /home/user/dump_file.ecf /home/user/xml_file.xml
```

To print an xml file using the `line` format:
`cat ../tests/xml_file.xml | kdb convert xmltool line`
Note that this command won't save the output, it will just display it to `stdout`.

# Chapter 340

# kdb-cp(1) – Copy keys within the key database

## 340.1 SYNOPSIS

```
kdb cp <source> <dest>
```

## 340.2 DESCRIPTION

This command copies key(s) in the Key database. You can copy keys to another directory within the database or even below another key. Note that you cannot copy a key below itself.
The argument `source` is the path of the key(s) you want to copy and `dest` is the path where you would like to copy the key(s) to. Note that when using the `-r` flag, `source` as well as all the keys below will be copied.

## 340.3 LIMITATIONS

Neither `source` nor `dest` can be a cascading key. (Start with `/`). Make sure to select a namespace.

## 340.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 1-10: standard exit codes, see kdb(1)
- 11: No key to copy found.

## 340.5 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-r`, `--recursive`: Recursively copy keys.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-f`, `--force`: Force overwriting the keys.

## 340.6 EXAMPLES

```
# Backup-and-Restore: user:/tests/cp/examples
# Create the keys we use for the examples
kdb set user:/tests/cp/examples/kdb-cp/key key1
kdb set user:/tests/cp/examples/kdb-cp/key/first key
kdb set user:/tests/cp/examples/kdb-cp/key/second key
kdb set user:/tests/cp/examples/kdb-cp/cpkey key1
kdb set user:/tests/cp/examples/kdb-cp/cpkey/first key
kdb set user:/tests/cp/examples/kdb-cp/cpkey/second key
kdb set user:/tests/cp/examples/kdb-cp/cpkeyerror/first key
kdb set user:/tests/cp/examples/kdb-cp/cpkeyerror/second anotherValue
kdb set user:/tests/cp/examples/kdb-cp/another/key one
kdb set user:/tests/cp/examples/kdb-cp/another/value two
# To copy a single key:
kdb cp user:/tests/cp/examples/kdb-cp/key user:/tests/cp/examples/kdb-cp/key2
#>
# To copy multiple keys:
kdb cp -r user:/tests/cp/examples/kdb-cp/key user:/tests/cp/examples/kdb-cp/copied
#>
# If the target-key already exists and has a different value, cp fails:
kdb cp -r user:/tests/cp/examples/kdb-cp/key user:/tests/cp/examples/kdb-cp/cpkeyerror
# RET: 11
# If the target-key already exists and has the same value as the source, everything is fine:
kdb cp -r user:/tests/cp/examples/kdb-cp/key user:/tests/cp/examples/kdb-cp/cpkey
#>
# To force the copy of keys:
kdb cp -rf user:/tests/cp/examples/kdb-cp/key user:/tests/cp/examples/kdb-cp/cpkeyerror
#>
# Now the key-values of /cpkeyerror are overwritten:
kdb export user:/tests/cp/examples/kdb-cp/cpkeyerror mini
#> =key1
#> first=key
#> second=key
# To copy keys below an existing key:
kdb cp -r user:/tests/cp/examples/kdb-cp/another user:/tests/cp/examples/kdb-cp/another/key
#>
kdb rm -r user:/tests/cp/examples/kdb-cp/
```

# Chapter 341

# kdb-dirname(1) – Get the cascading directory name of a key name

## 341.1 SYNOPSIS

`kdb dirname <key name>`
Where `key name` is the name of the key.

## 341.2 DESCRIPTION

This command is used to retrieve the directory name of a key, without any namespace.

## 341.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 7: invalid key name passed, see kdb(1) for a list of return codes used by kdb.

## 341.4 OPTIONS

- `-n`, `--no-newline`: Suppress the newline at the end of the output.

## 341.5 EXAMPLES

```
kdb dirname user:/key/subkey
#> /key
kdb dirname /
#> /
```

## 341.6 SEE ALSO

- See kdb-namespace(1) and kdb-basename(1) for other namepart extraction operations.
- The Unix analogue [DIRNAME(1)].
- To get keys in shell scripts, you can use kdb-sget(1).
- elektra-key-names(7) for an explanation of key names.

# Chapter 342

# kdb-editor(1) – Use your editor for editing KDB

## 342.1 SYNOPSIS

`kdb editor <path> [<format>]`
Where `path` is the destination where the user wants to edit keys and `format` is the format in which the keys should be edited. If the `format` argument is not passed, then the default format will be used as determined by the value of the `sw/kdb/current/format` key. By default, that key contains `storage`. The `storage` plugin can be configured at compile-time or changed by the link `libelektra-plugin-storage.so`. The `format` attribute relies on Elektra's plugin system to properly import the configuration. The user can view all plugins available for use by running the kdb-plugin-list(1) command. To learn about any plugin, the user can simply use the kdb-plugin-info(1) command.

## 342.2 DESCRIPTION

This command allows a user to edit configuration of the key database using a text editor. The user should specify the format that the current configuration or keys are in, otherwise the default format will be used.

## 342.3 RETURN VALUES

- 0: successful.
- 1-10: standard exit codes, see kdb(1)
- 11: could not export configuration.
- 12: could not start editor.
- 13: could not import configuration because of conflicts
- 14: could not import configuration because of error during kdbSet() (Most likely a validation error)

## 342.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-s`, `--strategy <name>`: Specify which strategy should be used to resolve conflicts.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-e`, `--editor <editor>`: Which editor to use.

## 342.5 Strategies

The strategies implemented by the merge framework and are documented in elektra-merge-strategy(7).

## 342.6 KDB

- `/sw/elektra/kdb/#0/current/format`: The default format if none given. Defaults to `storage` if the key does not exist.
- `/sw/elektra/kdb/#0/current/editor`: The default editor, if no `-e` option is given. Defaults to `/usr/bin/sensible-editor`, `/usr/bin/editor` or `/usr/bin/vi` if the key does not exist.

## 342.7 EXAMPLES

To change the configuration in KDB below `user:/ini` with `/usr/bin/vim`, you would use:
```
kdb editor -e /usr/bin/vim user:/ini
```
Or set a new editor as default using:
```
kdb set /sw/elektra/kdb/#0/current/editor /usr/bin/nano
```
To change the configuration in KDB below `user` in xml format, you would use:
```
kdb editor user xml
```

## 342.8 SEE ALSO

- elektra-merge-strategy(7)
- elektra-key-names(7) for an explanation of key names.

# Chapter 343

# kdb-export(1) – Export keys from the key database

## 343.1 SYNOPSIS

```
kdb export <source> [<format>]
```

## 343.2 DESCRIPTION

This command allows a user to export keys from the key database.
Keys are exported to `stdout` in whichever format is specified.
This command can also be used to view full key(s) including their values.

## 343.3 USAGE

Where `source` is the path of the key(s) you want to export. Additionally, the user can specify a format to use by passing it as the option argument `format`.
The `format` attribute relies on Elektra's plugin system to export the keys in the desired format.The user can view all plugins available for use by running the kdb-plugin-list(1) command. To learn about any plugin, the user can simply use the kdb-plugin-info(1) command.
The `storage` plugin can be configured at compile-time or changed by the link `libelektra-plugin-storage.so`.

## 343.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-E`, `--without-elektra`: Omit the `system:/elektra` directory.
- `-c`, `--plugins-config <plugins-config>`: Add a configuration to the format plugin.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 343.5 KDB

- `/sw/elektra/kdb/#0/current/format` Change default format (if none is given at commandline) and built-in default `storage` is not your preferred format.

## 343.6 EXAMPLES

To view your full key database in Elektra's `storage` format:
`kdb export /`

To backup your full key database in Elektra's `storage` format to a file called `full-backup.ecf`:
`kdb export / > full-backup.ecf`

To backup a keyset stored in `user:/keyset` in the `ini` format to a file called `keyset.ini`:
`kdb export user:/keyset ini > keyset.ini`

Change default format to `simpleini`:
`kdb set /sw/elektra/kdb/#0/current/format simpleini`
Create two key values and export them as `xml`:

```
kdb set user:/tests/kdb-export/one one
kdb set user:/tests/kdb-export/two two
kdb export user:/tests/kdb-export/ xml
#> <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
#> <kdb-export>
#>
#>   <one>one</one>
#>
#>   <two>two</two>
#>
#> </kdb-export>
kdb rm -r user:/tests
# cleanup
```

Create two key values and export them with the `xerces` plugin:

```
kdb set user:/tests/kdb-export/one one
kdb set user:/tests/kdb-export/two two
kdb export user:/tests/kdb-export/ xerces
#> <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
#> <kdb-export>
#>
#>   <one>one</one>
#>
#>   <two>two</two>
#>
#> </kdb-export>
kdb rm -r user:/tests
# cleanup
```

## 343.7 Note

- Only storage plugins can be used for formatting.

# Chapter 344

# kdb-file(1) – Displays which file a key is stored in

## 344.1 SYNOPSIS

`kdb file <key name>`

Where `key name` is the name of the key to check.

## 344.2 DESCRIPTION

This command prints which file a given key is stored in.
While many keys are stored in a default key database file, many others are stored in any number of configuration files located all over the system.
This tool is made to allow users to find out the file that a key is actually stored in.
This command makes use of Elektra's `resolver` plugin which the uer can learn more about by running the command `kdb plugin-info resolver`.

## 344.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-n`, `--no-newline`: Suppress the newline at the end of the output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 344.4 EXAMPLES

To find which file a key is stored in:
`kdb file user:/example/key`

## 344.5 SEE ALSO

- elektra-mounting(7)
- elektra-namespaces(7)

- elektra-key-names(7) for an explanation of key names.

# Chapter 345

# kdb-find-tools(1) – The tool for finding tools

## 345.1 SYNOPSIS

```
kdb find-tools [-h] [--warnings] [--good] [--alltags] [-n NAME] [-a AUTHOR] [-d DATE]
               [-t TAGS [TAGS ...]] [-b BRIEF] [-e EXECUTE]
```

## 345.2 DESCRIPTION

If you are looking for a tool, then you have found the right tool to find tools! `kdb find-tools` provides search and list functionality for tools.
Just enter `kdb find-tools` to get a list of names, type and short description of all available tools.
If you are looking for something special, then there are two ways:

1. Tag Search: Type `kdb find-tools --alltags` to get a list of all Tags in use. Then you can search with `kdb -t [TAGS [TAGS ...]]`
2. Full Text Search:
   - `kdb find-tools -n NAME` to search for a script name.
   - `kdb find-tools -b BRIEF` to search for a short text.
   - `kdb find-tools -a AUTHOR` to search for an author.
   - `kdb find-tools -d DATE` to search for a creation date.
   - `kdb find-tools -e EXECUTE` to search for a type.

All methods can be combined. For example if you search all bash scripts which do some configuration work. You can type `kdb find-tools -t configuration -e bash`.

## 345.3 The Right Way to Add Your Script to the Find Tools

Meta Tags as comments in the beginning of a script are parsed. Mate Tags start with an `@`, here is a list of all Meta Tags:

| MetaTag | Meaning |
|---|---|
| @author | Names and Emails (in <>) of the Authors as comma separated list |
| @brief | A Short Description (One Line!) |
| @tags | Comma Separated List of Tags |
| @date | Date when the script was created, use DD.MM.YYYY as format |

Do not mind the '\' at the beginning it is a doxygen escaping.
Beware, that these metatags should be applied at the beginning of the file (in the first 10 rows)!

## 345.4 Example

```
#!/bin/sh
#
# @author Kurt Micheli <kurt.micheli@libelektra.org>
# @brief This is an example of a build script
# @date 31.10.2018
# @tags configure, build
```

## 345.5 Notes

The Metatag System of Epydoc is used ( http://epydoc.sourceforge.net/manual-fields.↩ html#module-metadata-variables) and extended with special tags.

# Chapter 346

# kdb-find(1) – Find keys in the key database

## 346.1 SYNOPSIS

`kdb find <regex>`
Where `regex` is a regular expression which contains the key to find.

## 346.2 DESCRIPTION

This command will list the name of all keys that contain `regex`.

## 346.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-0`, `--null`: Use binary 0 termination.

## 346.4 EXAMPLES

```
# Backup-and-Restore: user:/tests/find
# We use the `dump` plugin, since some storage plugins, e.g. INI,
# create intermediate keys, such as `user:/tests/find/tests/foo`
# for the following test.
sudo kdb mount find.ecf user:/tests/find dump
# Create the keys we use for the examples
kdb set user:/tests/find/tests val1
kdb set user:/tests/find/tests/foo/bar val2
kdb set user:/tests/find/tests/fizz/buzz fizzbuzz
kdb set user:/tests/find/tostfizz val3
kdb set user:/tests/find/tust/level lvl
# list all keys containing /tests/find/t[eo]
kdb find '/tests/find/t[eo]'
#> user:/tests/find/tests
#> user:/tests/find/tests/fizz/buzz
#> user:/tests/find/tests/foo/bar
#> user:/tests/find/tostfizz
# list all keys containing fizz
kdb find 'fizz'
#> user:/tests/find/tests/fizz/buzz
#> user:/tests/find/tostfizz
kdb rm -r /tests/find
sudo kdb umount user:/tests/find
```

## 346.5 SEE ALSO

- If the user would also like to see the values of the keys below `path` then you should consider the kdb-export(1) command.
- elektra-key-names(7) for an explanation of key names.

# Chapter 347

# kdb-gen-highlevel(1) – High-level API code-generator template

## 347.1 SYNOPSIS

`kdb gen highlevel <parentKey> <outputName> [parameters...]`

- `<parentKey>`: the parent key to use, MUST be in the `spec` namespace
- `<outputName>`: the base name of the output files. `.c` will be appended for the source file and `.h` for the header file.
- `[parameters...]`: see below

## 347.2 DESCRIPTION

This command invokes the code-generator with the template for the high-level API.
The input for this template is a specification. Keys below the `parentKey` which have a `type` metakey are considered part of this specification. Every such key must also either have a `default` metadata or alternatively must be marked with the `require` metakey. Keys marked with `require` must be set by the user, otherwise the initialization of the Elektra handle will fail.
The `type` metakey may only have one of the following values: `enum`, `string`, `boolean`, `char`, `octet`, `short`, `unsigned_short`, `long`, `unsigned_long`, `long_long`, `unsigned_long_long`, `float`, `double`, `long_double`, `struct`, `struct_ref` and `discriminator`. Most of these values correspond to the values supported by the high-level API, the remaining values (`enum`, `struct`, `struct_ref`, `discriminator`) are treated specially and are part of advanced concepts. For more information on these concepts take a look at elektra-highlevel-gen(7). If one of the advanced `type` values is used, you should also set `check/type = any`; otherwise the `type` plugin may produce an error.
The template produces at least three output files: `<outputName>.c`, `<outputName>.h` and `<outputName>.mount.sh`. The `.c` file only contains implementations, therefore its precise content will not be documented.
The header (`.h`) file contains the following parts:

1. Generated `enum`s and `struct`s
2. Declarations for generated accessor functions
3. Tags for accessing keys
4. `static inline` accessor functions for all tags
5. Declarations of initialization functions
6. Convenience accessor macros

Anything else that may be part of the header file is not considered public API and may be subject to change at any point in time. There is also no guarantee of full compatibility between Elektra version for the documented parts of the header, however, we try to have as little breaking changes as possible for the six parts listed above.
The `.mount.sh` file is a shell script that can be used to mount the specification for the application. You can either use it as a basis for your own script, or wrap it in another script that correctly sets `APP_PATH` or `SPEC_FILE` (depending on your configuration). If the generated script happens to use the correct paths already, you can of course use it directly as well.
For detailed information about the contents of the header file see elektra-highlevel-gen(7).

## 347.3 PARAMETERS

- `initFn`: Changes the name of the initialization function (default: `loadConfiguration`)
- `helpFn`: Changes the name of the function that prints the generated help message (default: `printHelpMessage`)
- `specloadFn`: Changes the name of the function that checks for "specload mode" (default: `exitForSpecload`)
- `tagPrefix`: Changes the prefix of the generated tags (default: `ELEKTRA_TAG_`)
- `embedHelpFallback`: Switches whether a fallback help message should be embedded; allowed values: `1` (default), `0` If enabled (`1`), a help message will be generated from the specification passed to the code-generator and embedded into the application. This message will be used, if the normal help message could not be generated at runtime.
- `enumConv`: Switches how enum conversion should be done; allowed values: `default` (default), `switch`, `strcmp`
  - `strcmp`: uses a simple series of `if (strcmp(*, *) == 0)` to convert strings into enums
  - `switch`: constructs a series of `switch` statements to convert strings into enums
  - `auto`: uses a `switch` up to a depth of 2, then switches to `strcmp`
- `headers`: Comma-separated (`,`) list of additional header files to include. For each of the listed headers we will generate an `#include "*"` statement
- `genSetters`: Switches whether setters should be generated at all; allowed values: `1` (default), `0`
- `embeddedSpec`: Changes how much of the specification is embedded into the application; allowed values: `full` (default), `defaults`, `none`. see elektra-highlevel-gen(7)

## 347.4 EXAMPLES

The simplest invocation is:
```
kdb gen highlevel /sw/org/app/#0/current config
```
However, it is not recommended having the code-generator read from the KDB, so one should instead use:
```
kdb gen -F ni=spec.ini highlevel /sw/org/app/#0/current config
```
If you don't want to embed the full specification in your binary, we recommend:
```
kdb gen -F ni=spec.ini highlevel /sw/org/app/#0/current config embeddedSpec=defaults
```
For the minimal binary size you may use (this comes with its own drawbacks, see elektra-highlevel-gen(7)):
```
kdb gen -F ni=spec.ini highlevel /sw/org/app/#0/current config embeddedSpec=none
```

## 347.5 SEE ALSO

- kdb(1) for general information about the `kdb` tool.
- elektra-spec(7) for an explanation of the `spec` namespace.

# Chapter 348

# kdb-gen(1) – Elektra's code-generator

## 348.1 SYNOPSIS

`kdb gen <templateName> <parentKey> <outputName> [parameters...]`

- `<templateName>`: one of the templates listed below
- `<parentKey>`: the parent key to use, templates may have certain restrictions on e.g. the allowed namespaces You may also use the special value `-` (see below).
- `<outputName>`: the base name of the output files. If a template produces multiple files, it will append different suffixes (e.g. file extensions) to this base name.
- `[parameters...]`: a list of parameters, the supported parameters depend on the template

## 348.2 DESCRIPTION

This command invokes Elektra's code-generator.
It supports different templates. All templates require a `parentKey` parameter, because this determines the input for the code-generator, as well as an `outputName` parameter to specify the output file(s).
If the given `parentKey` is `-`, instead of actually invoking the code generator, `kdb gen` will just print the filenames of the files that would be written with a valid `parentKey`.
For more information see the `list of templates` below and the man-pages for each of them.

## 348.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 1-10: standard exit codes, see kdb(1)

## 348.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-F`, `--input-file <plugin>=<file>`: Load the file `<file>` with plugin `<plugin>` instead of accessing the KDB.
- `-p`, `--profile <profile>`: Use a different kdb profile.

- `-v`, `--verbose`: Explain what is happening. Gives a complete trace of all tried keys. Very useful to debug fallback and overrides.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 348.5 TEMPLATES

Currently we only have one template:

- `highlevel`: Generates the files needed for using the high-level API with code-generation. More information can be found in kdb-gen-highlevel(1)

## 348.6 EXAMPLES

Examples can be found on the man-pages of each template.

## 348.7 SEE ALSO

- kdb(1) for general information about the `kdb` tool.
- elektra-key-names(7) for an explanation of key names.

# Chapter 349

# kdb-get(1) – Get the value of a key stored in the key database

## 349.1 SYNOPSIS

`kdb get <key name>`
Where `key name` is the name of the key.

## 349.2 DESCRIPTION

This command is used to retrieve the value of a key.
If you enter a `key name` starting with a leading `/`, then a cascading lookup will be performed in order to attempt to locate the key. In this case, using the `-v` option allows the user to see the full key name of the key if it is found.

## 349.3 LIMITATIONS

Only keys within the mount point or below the `<key name>` will be considered during a cascading lookup. A workaround is to pass the `-a` option. Use the command `kdb get -v <key name>` to see if an override or a fallback was considered by the lookup.

## 349.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 1-10: standard exit codes, see kdb(1)
- 11: No key found.

## 349.5 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-a`, `--all`: Consider all of the keys.
- `-n`, `--no-newline`: Suppress the newline at the end of the output.

- `-v`, `--verbose`: Explain what is happening. Gives a complete trace of all tried keys. Very useful to debug fallback and overrides.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 349.6 EXAMPLES

```
# Backup-and-Restore: user:/tests/get/examples
# We use the 'dump' plugin, since some storage plugins, e.g. INI,
# create intermediate keys.
sudo kdb mount get.ecf user:/tests/get/examples/kdb-get dump
sudo kdb mount get.ecf spec:/tests/get/examples/kdb-get dump
# Create the keys we use for the examples
kdb set user:/tests/get/examples/kdb-get/key myKey
kdb meta-set spec:/tests/get/examples/kdb-get/anotherKey default defaultValue
# To get the value of a key:
kdb get user:/tests/get/examples/kdb-get/key
#> myKey
# To get the value of a key using a cascading lookup:
kdb get /tests/get/examples/kdb-get/key
#> myKey
# To get the value of a key without adding a newline to the end of it:
kdb get -n /tests/get/examples/kdb-get/key
#> myKey
# To explain why a specific key was used (for cascading keys):
kdb get -v /tests/get/examples/kdb-get/key
#> got 3 keys
#> searching spec:/tests/get/examples/kdb-get/key, found: <nothing>, options: KDB_O_CALLBACK
#>     searching proc:/tests/get/examples/kdb-get/key, found: <nothing>
#>     searching dir:/tests/get/examples/kdb-get/key, found: <nothing>
#>     searching user:/tests/get/examples/kdb-get/key, found: user:/tests/get/examples/kdb-get/key
#> The resulting keyname is user:/tests/get/examples/kdb-get/key
#> The resulting value size is 6
#> myKey
# Output if only a default value is set for a key:
kdb get -v /tests/get/examples/kdb-get/anotherKey
#> got 3 keys
#> searching spec:/tests/get/examples/kdb-get/anotherKey, found:
       spec:/tests/get/examples/kdb-get/anotherKey, options: KDB_O_CALLBACK
#> The key was not found in any other namespace, taking the default
#> The resulting keyname is default:/tests/get/examples/kdb-get/anotherKey
#> The resulting value size is 13
#> defaultValue
kdb rm user:/tests/get/examples/kdb-get/key
kdb rm spec:/tests/get/examples/kdb-get/anotherKey
sudo kdb umount user:/tests/get/examples/kdb-get
sudo kdb umount spec:/tests/get/examples/kdb-get
```

To use bookmarks:

`kdb get +kdb/format`

This command will actually get `user:/sw/elektra/kdb/#0/current/format` if the bookmarks commands from kdb-set(1) man pages are executed before.

## 349.7 SEE ALSO

- kdb(1) for how to configure the kdb utility and use the bookmarks.
- For more about cascading keys see elektra-cascading(7)
- To get keys in shell scripts, you also can use kdb-sget(1)
- elektra-key-names(7) for an explanation of key names.

# Chapter 350

# kdb-help(1) – Show man page for elektra tools

## 350.1 SYNOPSIS

```
kdb help <tool>
kdb --help <tool>
kdb -H <tool>
```

## 350.2 DESCRIPTION

Show a man page for one of Elektra's tools. Note that `kdb <tool> --help` might have a different behavior, depending on the tool.
Also note that, no additional commandline options are allowed for this kind of invocation. If you want that use `--help` or `-H` after `<tool>`.

## 350.3 KDB

- `/sw/elektra/kdb/#0/current/man`: The man(1) utility to be used. Defaults to /usr/bin/man

## 350.4 EXAMPLES

Show how to set keys: `kdb help set`
Use the info program as man page viewer for the current user: `kdb set user:/sw/elektra/kdb/#0/current/man /usr/bin/info`

# Chapter 351

# kdb-import(1) – Import an existing configuration into the key database

## 351.1 SYNOPSIS

`kdb import <destination> [<format>]`
Where `destination` is the destination where the user wants the keys to be imported into. `format` is the format of the keys that are imported.

## 351.2 DESCRIPTION

If the `format` argument is not passed, then the default format will be used as determined by the value of the `sw/kdb/current/format` key. By default, that key is set to the `storage` format. The `format` attribute relies on Elektra's plugin system to properly import the configuration. The user can view all plugins available for use by running the kdb-plugin-list(1) command. To learn about any plugin, the user can simply use the kdb-plugin-info(1) command.
This command allows a user to import an existing configuration into the key database. The configuration that the user wants to import is read from `stdin`. The user should specify the format that the current configuration or keys are in, otherwise the default format will be used. The default format is `storage` but can be changed by editing the value of the `/sw/elektra/kdb/#0/current/format` key. The `storage` plugin can be configured at compile-time or changed by the link `libelektra-plugin-storage.so`.

## 351.3 CONFLICTS

Conflicts can occur when importing a configuration to a part of the database where keys already exist. Conflicts when importing can be resolved using a strategy with the `-s` argument.
The strategies implemented by the merge framework are documented in elektra-merge-strategy(7).

## 351.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-s`, `--strategy <name>`: Specify which strategy should be used to resolve conflicts.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-c`, `--plugins-config`: Add a configuration to the format plugin.

## 351.5 KDB

- `/sw/elektra/kdb/#0/current/verbose`: Same as `-v`: Explain what is happening (output merged keys).
- `/sw/elektra/kdb/#0/current/format` Change default format (if none is given at commandline) and built-in default is not your preferred format.

## 351.6 EXAMPLES

To import a configuration stored in the XML format in a file called `example.xml` below `user:/keyset`:
`kdb import user:/keyset xmltool < example.xml`
To import a configuration stored in the `ini` format in a file called `example.ini` below `user:/keyset` replacing any previous keys stored there:
`cat example.ini | kdb import -s cut user:/keyset ini`
To import a configuration stored in the `ini` format in a file called `example.ini` below `user:/keyset` keeping any previous keys stored there that aren't present in the newly imported configuration:
`cat example.ini | kdb import -s import user:/keyset ini`
To restore a backup (stored as `sw.ecf`) of a user's configuration below `system:/sw`:
`cat sw.ecf | kdb import system:/sw`
To import a sample `xml` content with the `xerces` plugin:

```
# import two keys from a xml string
kdb import user:/tests/kdb-import/ xerces << "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"
      ?><kdb-import><one>one</one><two>two</two></kdb-import>"
# get the values and verify they got imported correctly
kdb get user:/tests/kdb-import/one
#> one
kdb get user:/tests/kdb-import/two
#> two
kdb rm -r user:/tests/kdb-import
```

To import a sample `xml` content via specifying the file format directly:

```
# import two keys from a xml string
kdb import user:/tests/kdb-import/ xml << "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"
      ?><kdb-import><one>one</one><two>two</two></kdb-import>"
# get the values and verify they got imported correctly
kdb get user:/tests/kdb-import/one
#> one
kdb get user:/tests/kdb-import/two
#> two
kdb rm -r user:/tests/kdb-import
```

## 351.7 SEE ALSO

- elektra-merge-strategy(7)

# Chapter 352

# kdb-install-config-file(1) – Install configuration files in Elektra

## 352.1 SYNOPSIS

`kdb install-config-file <elektra path> <config file> [<format>]`

- `<elektra path>` is a path in Elektra
- `<config file>` is some file on the filesystem

`line` will be used as `<format>` if `<format>` is unspecified.
The script has to be called as administrator (e.g. with `sudo`).

## 352.2 DESCRIPTION

This script installs or merges configuration files from the file system into Elektra. There are two possible scenarios:

1. You use the script for the first time for a file. Then `<elektra path>` is empty. The script
   (a) copies `<config file>` into a special path to preserve origin version.
   (b) mounts `<config file>` into `<elektra path>` as our version. This version can then be safely modified.
2. You have already used the script for a previous version of the file. In this case `<elektra path>` already contains data the script performs a three-way merge (using `kdb cmerge`) between the file at `<config file>` (their), the `<elektra path>` (our) and the preserved key set (base) from step 1 in scenario 1.

## 352.3 EXAMPLES

To install the config file at `~/.config/installing.ini` we can use the following command
`kdb install-config-file user:/tests/installing installing.ini ini`

# Chapter 353

# kdb-list-commands(1) – List commands available to Elektra

## 353.1 SYNOPSIS

`kdb list-commands`

## 353.2 DESCRIPTION

This command will list all internal kdb commands. The output is suitable to be processed from other tools as long as `-v` is not used.

## 353.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Also output some explanation for every command. The command names are bold if color is turned on. In the first line it will add how many commands exist.
- `-0`, `--null`: Use binary 0 termination

## 353.4 EXAMPLES

To get a list of all available commands with their help text: `kdb list-commands -v`

## 353.5 SEE ALSO

- `kdb list-tools` for external kdb commands

# Chapter 354

# kdb-list-tools(1) - List all external tools available to Elektra

## 354.1 SYNOPSIS

```
kdb list-tools
```

## 354.2 DESCRIPTION

This command lists all the external tools that are available to Elektra. In the first line it prints where external tools are located.
The tool itself is extern.

## 354.3 ENVIRONMENT

- `KDB_EXEC_PATH`: Path to additional external tools. Can contain multiple paths separated with :.

## 354.4 OPTIONS

Currently no option provided.

## 354.5 SEE ALSO

- `kdb list-commands` to list internal kdb commands

# Chapter 355

# kdb-ls(1) – List keys in the key database

## 355.1 SYNOPSIS

`kdb ls <path>`
Where `path` is the path in which the user would like to list keys below.

## 355.2 DESCRIPTION

This command will list the name of all keys below a given path.

## 355.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-m`, `--min-depth <min-depth>`: Specify the minimum path depth of the output (0 by default), exclusive and relative to the name to list.
- `-M`, `--max-depth <max-depth>`: Specify the maximum path depth of the output (unlimited by default, 1 to show only the next level), inclusive and relative to the name to list.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-0`, `--null`: Use binary 0 termination.

## 355.4 EXAMPLES

```
# Backup-and-Restore: user:/tests/examples
# We use `dump` as storage format here, since storage plugins such as INI
# automatically add keys between levels (e.g. `user:/tests/examples/kdb-ls/test/foo`).
sudo kdb mount ls.ecf user:/tests/examples dump
# Create the keys we use for the examples
kdb set user:/tests/examples/kdb-ls/test val1
kdb set user:/tests/examples/kdb-ls/test/foo/bar val2
kdb set user:/tests/examples/kdb-ls/test/fizz/buzz fizzbuzz
kdb set user:/tests/examples/kdb-ls/tost val3
kdb set user:/tests/examples/kdb-ls/tost/level lvl
# list all keys below /tests/examples/kdb-ls
kdb ls /tests/examples/kdb-ls
#> user:/tests/examples/kdb-ls/test
#> user:/tests/examples/kdb-ls/test/fizz/buzz
#> user:/tests/examples/kdb-ls/test/foo/bar
#> user:/tests/examples/kdb-ls/tost
```

```
#> user:/tests/examples/kdb-ls/tost/level
# list the next level of keys below /tests/examples/kdb-ls
# note that if the search key ends with a /, it lists the next level
kdb ls /tests/examples/kdb-ls/ --max-depth=1
#> user:/tests/examples/kdb-ls/test
#> user:/tests/examples/kdb-ls/tost
# list the current level of keys below /tests/examples/kdb-ls
# note the difference to the previous example
kdb ls /tests/examples/kdb-ls --max-depth=1
# this yields no output as /tests/examples/kdb-ls is not a key
# list all keys below /tests/examples/kdb-ls with are minimum 1 level (inclusive) away from that key
# and maximum 2 levels away (exclusive)
kdb ls /tests/examples/kdb-ls --min-depth=1 --max-depth=2
#> user:/tests/examples/kdb-ls/test
#> user:/tests/examples/kdb-ls/tost
# list all keys below /tests/examples/kdb-ls/test
kdb ls /tests/examples/kdb-ls/test
#> user:/tests/examples/kdb-ls/test
#> user:/tests/examples/kdb-ls/test/fizz/buzz
#> user:/tests/examples/kdb-ls/test/foo/bar
# list all keys under /tests/examples/kdb-ls in verbose mode
kdb ls /tests/examples/kdb-ls/ -v
#> size of all keys in mount point: 5
#> size of requested keys: 5
#> user:/tests/examples/kdb-ls/test
#> user:/tests/examples/kdb-ls/test/fizz/buzz
#> user:/tests/examples/kdb-ls/test/foo/bar
#> user:/tests/examples/kdb-ls/tost
#> user:/tests/examples/kdb-ls/tost/level
kdb rm -r user:/tests/examples
sudo kdb umount user:/tests/examples
```

## 355.5 SEE ALSO

- If the user would also like to see the values of the keys below `path` then you should consider the kdb-export(1) command.
- elektra-key-names(7) for an explanation of key names.

# Chapter 356

# kdb-merge(1) – Three-way merge of KeySets

## 356.1 SYNOPSIS

`kdb merge [options] ourpath theirpath basepath resultpath`

- ourpath: Path to the keyset to serve as `ours`
- theirpath: path to the keyset to serve as `theirs`
- basepath: path to the `base` keyset
- resultpath: path without keys where the merged keyset will be saved

## 356.2 DESCRIPTION

Does a three-way merge between keysets.
On success the resulting keyset will be saved to mergepath.
On unresolved conflicts nothing will be changed.

## 356.3 THREE-WAY MERGE

The `kdb merge` command uses a three-way merge by default.
A three-way merge is when three versions of a file (or in this case, KeySet) are compared in order to automatically merge the changes made to the KeySet over time.
These three versions of the KeySet are:

- `base`: The `base` KeySet is the original version of the KeySet.
- `ours`: The `ours` KeySet represents the user's current version of the KeySet.
  This KeySet differs from `base` for every key you changed.
- `theirs`: The `theirs` KeySet usually represents the default version of a KeySet (usually the package maintainer's version).
  This KeySet differs from `base` for every key someone has changed.

The three-way merge works by comparing the `ours` KeySet and the `theirs` KeySet to the `base` KeySet. By looking for differences in these KeySets, a new KeySet called `result` is created that represents a merge of these KeySets.

## 356.4 CONFLICTS

Conflicts occur when a Key has a different value in all three KeySets.
Conflicts in a merge can be resolved using a `strategy` with the `-s` option. To interactively resolve conflicts, use the `-i` option.

## 356.5 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-f`, `--force`: Will remove existing keys from `resultpath` instead of failing.
- `-s`, `--strategy <name>`: Specify which strategy should be used to resolve conflicts.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-i`, `--interactive` Interactively resolve the conflicts.

## 356.6 RETURN VALUE

- 0: Successful.
- 11: A conflict occurred during the merge.

## 356.7 EXAMPLES

To complete a simple merge of three KeySets:
```
kdb merge user:/ours user:/theirs user:/base user:/result
```

To complete a merge whilst using the `ours` version of the KeySet to resolve conflicts:
```
kdb merge -s ours user:/ours user:/theirs user:/base user:/result
```

To complete a three-way merge and overwrite all current keys in the `resultpath`:
```
kdb merge -s cut user:/ours user:/theirs user:/base user:/result
```

## 356.8 SEE ALSO

- elektra-merge-strategy(7)
- elektra-key-names(7) for an explanation of key names.

# Chapter 357

# kdb-meta-get(1) – Get the value of a metakey stored in the key database

## 357.1 SYNOPSIS

`kdb meta-get <key name> <metaname>`

Where `key name` is the name of the key and `metaname` is the name of the metakey the user would like to access.

## 357.2 DESCRIPTION

This command is used to print the value of a metakey. A metakey is information stored in a key which describes that key.
The handling of cascading `key name` does not differ to `kdb get`. Make sure to use the namespace `spec`, if you want metadata from there.

## 357.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 11: Key not found. (Invalid `path`)
- 12: Metakey not found. (Invalid `metaname`).

## 357.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-n`, `--no-newline`: Suppress the newline at the end of the output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 357.5 EXAMPLES

To get the value of a metakey called `description` stored in the key `spec:/example/key`:
`kdb meta-get spec:/example/key description`
To get the value of metakey called `override/#0` stored in the key `spec:/example/dir/key`:
`kdb meta-get spec:/example/dir/key "override/#0"`

## 357.6 SEE ALSO

- How to set metadata: kdb-meta-set(1)
- For more about cascading keys see elektra-cascading(7)
- elektra-metadata(7) for an explanation of the metadata concepts.
- elektra-key-names(7) for an explanation of key names.

# Chapter 358

# kdb-meta-ls(1) - Print metakeys associated with a key

## 358.1 SYNOPSIS

`kdb meta-ls <key name>`
Where `key name` is the name of the key.

## 358.2 DESCRIPTION

This command prints the names of all metakeys associated with the key named `key name`.
If no metakeys are associated with the given key, nothing will be printed.

## 358.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-0`, `--null`: Use binary 0 termination.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings. Show which key will be used.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 358.4 EXAMPLE

To see which metakeys are associated with a key:
`kdb meta-ls /example/key`

## 358.5 SEE ALSO

- elektra-metadata(7) for an explanation of the metadata concepts.
- elektra-key-names(7) for an explanation of key names.

# Chapter 359

# kdb-meta-rm(1) – Remove metakey of a key from the key database

## 359.1 SYNOPSIS

`kdb meta-rm <key name> <metaname>`
Where `key name` is the name of the key and `metaname` is the name of the metakey you want to remove.

## 359.2 DESCRIPTION

This command removes a metakey of a key from the Key database.

## 359.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 359.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 11: Key not found. (Invalid `path`)

## 359.5 EXAMPLES

To remove metakey `metakey` of a key:
`kdb meta-rm user:/example metakey`

## 359.6 SEE ALSO

- elektra-metadata(7) for an explanation of the metadata concepts.
- elektra-key-names(7) for an explanation of key names.

# Chapter 360

# kdb-meta-set(1) – Set the value of a metakey

## 360.1 SYNOPSIS

`kdb meta-set <key name> <metaname> [<metavalue>]`
Where `key name` is the name of the key that the metakey is associated with, `metaname` is the name of the metakey the user would like to set the value of (or create), and `metavalue` is the value the user wishes to set the metakey to. If no `metavalue` is given, the metakey will be removed.

## 360.2 DESCRIPTION

This command allows the user to set the value of an individual metakey. If a (non-cascading) key does not already exist and the user tries setting a metakey associated with it, the key will be created with a null value. If a cascading key is given that does not resolve to an actual key, the operation is aborted.
There is some special handling for the metadata atime, mtime and ctime. They will be converted to time_t.

## 360.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-q`, `--quiet`: Suppress non-error messages.
- `-f`, `--force`: Do not perform cascading KDB operations if the key provided has a namespace. For example, this bypasses validation specified in the spec: namespace for the given key.

## 360.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 11: Key not found. (Invalid `path`)
- 12: Setting a non-existing key without a namespace is not possible.

## 360.5 KDB

- `/sw/elektra/kdb/#0/current/verbose`: Same as `-v`: Explain what is happening.
- `/sw/elektra/kdb/#0/current/quiet`: Same as `-q`: Suppress default messages.

## 360.6 EXAMPLES

To set a metakey called `description` associated with the key `user:/example/key` to the value `Hello World!`:

```
kdb meta-set spec:/example/key description "Hello World!"
```

To create a new key `spec:/example/newkey` with a null value (if it did not exist before) and a metakey `namespace/#0` associated with it to the value `system`:

```
kdb meta-set /example/newkey "namespace/#0" system
```

To create an override link for a `/test` key:

```
kdb set /overrides/test "example override"
sudo kdb meta-set spec:/test override/#0 /overrides/test
```

To remove it:

```
sudo kdb meta-set spec:/test override/#0
```

## 360.7 SEE ALSO

- How to get metadata: kdb-meta-get(1)
- elektra-metadata(7) for an explanation of the metadata concepts.
- elektra-key-names(7) for an explanation of key names.

# Chapter 361

# kdb-meta-show(1) – Print all metakeys along with their value

## 361.1 SYNOPSIS

`kdb meta-show <key name>`

Where `key name` is the name of the key the user would like to access.

## 361.2 DESCRIPTION

This command is used to print all metakeys and its values for a given key. A metakey is information stored in a key which describes that key.
The handling of cascading `key name` does not differ to `kdb get`. Make sure to use the namespace `spec`, if you want metadata from there.

## 361.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 1: Key not found. (Invalid `path`)

## 361.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 361.5 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 11: Key not found. (Invalid `path`)

## 361.6 EXAMPLES

```
# Backup-and-Restore: user:/tests/examples
# We use `dump` as storage format here
sudo kdb mount ls.ecf user:/tests/examples dump
# Create the keys we use for the examples
kdb set user:/tests/examples/kdb-meta-show test
kdb meta-set user:/tests/examples/kdb-meta-show meta1 val1
kdb meta-set user:/tests/examples/kdb-meta-show meta2 val2
kdb meta-set user:/tests/examples/kdb-meta-show meta3 val3
kdb meta-set user:/tests/examples/kdb-meta-show meta4 val4
# list all meta keys for /tests/examples/kdb-meta-show
kdb meta-show /tests/examples/kdb-meta-show
#> meta1: val1
#> meta2: val2
#> meta3: val3
#> meta4: val4
kdb rm -r user:/tests/examples
sudo kdb umount user:/tests/examples
```

## 361.7 SEE ALSO

- How to set metadata: kdb-meta-set(1)
- For more about cascading keys see elektra-cascading(7)
- elektra-metadata(7) for an explanation of the metadata concepts.
- elektra-key-names(7) for an explanation of key names.

# Chapter 362

# kdb-mount-java(1) – Mounting Java Plugins

## 362.1 SYNOPSIS

`kdb mount-java [<path> <mount point>] [<plugin> [<config>] [..]]`
All options except `<plugin>` are passed directly to `kdb mount`. For details on these options see `kdb-mount(1)`.
The `<plugin>` arguments are preprocessed before they are passed to `kdb mount`. If a `<plugin>` argument starts with `java:`, it is interpreted as a fully-qualified Java class name. The script replaces these `java:*` arguments with the appropriate arguments for `kdb mount` (see below).
`<plugin>` arguments that start with `kdb:` or have no prefix, are passed to `kdb mount` as is (after removing the optional `kdb:` prefix). The `kdb:` prefix exists to avoid potential name collisions between a Java plugin and C plugin.

## 362.2 DESCRIPTION

This helper command exists, because (unlike a "normal" C plugin) mounting a Java plugin requires multiple arguments. This is because you actually need to mount the `process` plugin with the correct configuration for a Java plugin.
The following command mounts the C plugins `dump` and `type` together with the Java plugin `org.libelektra.plugin.WhitelistPlugin` with the file `config.file` at `user:/tests/process`:

```
sudo kdb mount-java config.file user:/tests/process kdb:dump java:org.libelektra.plugin.WhitelistPlugin type
```

This internally expands to (something like) this complicated snippet:

```
KDB_VERSION="$(kdb --version | sed -nE 's/KDB_VERSION: (.+)/\1/gp')"
sudo kdb mount config.file user:/tests/process dump process "executable=$(command -v java)" 'args=#3'
       'args/#0=-cp'
       "args/#1=/usr/share/java/libelektra-$KDB_VERSION-all.jar:/usr/share/java/process-$KDB_VERSION.jar:/usr/share/java/white
       'args/#2=org.libelektra.process.PluginProcess' 'args/#3=org.libelektra.plugin.WhitelistPlugin' type
```

As you can see, every argument for a Java plugin is replaced by several arguments. Even worse, the arguments depend on how Elektra is installed on your system. The `mount-java` knows where Elektra's JARs are installed (`/usr/share/java` above) and constructs a classpath argument for the JVM. It also chooses the `PluginProcess` class as the main class and finally instructs this class to load the `WhitelistPlugin`.
However, `mount-java` isn't magic and it only works this smoothly for plugins that come bundled with Elektra. If you have external plugins, you must also specify the additional classpath for these plugins like this:

```
export CLASSPATH=/path/to/foo.jar:/path/to/bar.jar
sudo --preserve-env=CLASSPATH kdb mount-java config.file user:/tests/foobar java:org.example.Foo
       java:org.example.Bar java:org.libelektra.plugin.WhitelistPlugin
```

Here we added `/path/to/foo.jar` and `/path/to/bar.jar` to the classpath, so that the classes `org.example.Foo` and `org.example.Bar` can be found. We can still use `org.libelektra.plugin.WhitelistPlugin`, because Elektra's JARs are always added to the classpath.

## 362.3 SEE ALSO

- kdb-mount(1).

# Chapter 363

# kdb-mount-list-all-files – List all mounted files

## 363.1 SYNOPSIS

```
kdb mount-list-all-files
```

## 363.2 DESCRIPTION

This command prints a sorted list of all files under the control of Elektra

## 363.3 EXAMPLES

```
kdb mount
#> none on system:/info/elektra/constants with name system:/info/elektra/constants
#> /usr/local/share/doc/elektra/CONTRACT.ini on system:/info/elektra/contract/#0 with name
       system:/info/elektra/contract/#0
#> none on system:/info/elektra/desktop with name system:/info/elektra/desktop
#> /usr/local/share/doc/elektra/METADATA.ini on system:/info/elektra/metadata/#0 with name
       system:/info/elektra/metadata/#0
#> none on system:/info/elektra/uname with name system:/info/elektra/uname
#> /tmp/test.ini on system:/test with name system:/test
kdb mount-list-all-files
#> none
#> /tmp/test.ini
#> /usr/local/share/doc/elektra/CONTRACT.ini
#> /usr/local/share/doc/elektra/METADATA.ini
```

# Chapter 364

# kdb-mount(1) - Mount a file to the key database

## 364.1 SYNOPSIS

`kdb mount [<path> <mount point>] [<plugin> [<config>] [..]]`

- Where `path` is the path to the file the user wants to mount. See `kdb plugin-info resolver` for details what an absolute and relative path means. See also IMPORTANT below.
- `mountpoint` is where in the key database the new backend should be mounted. For a cascading mount point, `mountpoint` should start with `/`. See also IMPORTANT below.
- A list of such plugins with a configuration for each of them can be given:
  - `plugin` should be an Elektra plugin.
  - Plugins may be followed by a `,` separated list of `keys=values` pairs which will be used as plugin configuration.

## 364.2 DESCRIPTION

This command allows a user to persistently mount a new *backend*. Mounting in Elektra allows the user to mount a file into the current key database like a user may mount a partition into the current file system. This functionality is key to Elektra as it allows users to build a global key database comprised of many different configuration files.
A backend acts as a worker to allow Elektra to interpret configuration files as keys in the central key database such that any edits to the keys are reflected in the file and vice versa. Additionally, the user can use this command to list the currently mounted backends by running the command with no arguments. More about mounting is explained in elektra-mounting(7).

## 364.3 IMPORTANT

This command writes into the `/etc` directory to make the mounting persistent. As such it requires root permissions. Use `kdb file system:/elektra/mountpoints` to find out where exactly it will write to.
Absolute paths are still relative to their namespace (see `kdb plugin-info resolver`). Only system+spec mount points are actually absolute. Read elektra-namespaces(7) for further information.
For cascading mount points (starting with `/`) a mount point for the namespace `dir`, `user` and `system` is created. Each of this mount point uses a different configuration file, either below current directory, below home directory or anywhere in the system. Use `kdb file <path>` to determine where the file(s) are.

## 364.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.

- `-p, --profile <profile>`: Use a different kdb profile.
- `-C, --color <when>`: Print never/auto(default)/always colored output.
- `-v, --verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d, --debug`: Give debug information or ask debug questions (in interactive mode). Prints additional information in case of errors/warnings.
- `-q, --quiet`: Suppress non-error messages.
- `-i, --interactive`: Instead of passing all mounting information by parameters ask the user interactively.
- `-s, --strategy`: (experimental, use with care) By default mounting is rejected if the mountpoint already exists. With the strategy *unchanged* you can change the behavior to be successful if *exactly* the same config would be written (see #1306 why this does not always work correctly).
- `-R, --resolver <resolver>` Specify the resolver plugin to use if no resolver is given, the default resolver is used. See also `below in KDB`.
- `-0, --null`: Use binary 0 termination.
- `-1, --first`: Suppress the first column.
- `-2, --second`: Suppress the second column.
- `-c, --plugins-config <plugins-config>`: Add a plugin configuration for all plugins.
- `-W, --with-recommends`: Also add recommended plugins and warn if they are not available.
- `-f, --force`: Unmount before mounting: Does not fail on already existing mount points.

## 364.5 KDB

- `/sw/elektra/kdb/#0/current/quiet`: Same as `-q`: Suppress default messages.
- `/sw/elektra/kdb/#0/current/resolver`: The resolver that will be added automatically, if `-R` is not given.
- `/sw/elektra/kdb/#0/current/plugins`: It contains a space-separated list of plugins and their configs which are added automatically (by default sync). The plugin-configuration syntax is as described above in the `synopsis`.

## 364.6 EXAMPLES

To list the currently mounted backends:

```
kdb mount
```

To mount a system configuration file using the ini format:

```
kdb mount /etc/configuration.ini system:/example ini
```

Print a null-terminated output of paths and backend names:

```
kdb mount -02 | xargs -0n 2 echo
```

To mount the /etc/file system file with two plugins with a respective configuration option each:

```
kdb mount /etc/file system:/file plugin1 plugin1config=config1 plugin2 plugin2config=con
```

To mount the /etc/file system file with two plugins and setting both to be verbose:

```
kdb mount -c verbose=1 /etc/file system:/file plugin1 plugin2
```

To recode and rename a configuration file using Elektra:

```
kdb mount recode.txt dir:/recode ni rename cut=path iconv to=utf8,from=latin1
```

## 364.7 SEE ALSO

- elektra-glossary(7).
- kdb-spec-mount(1).
- kdb-umount(1).
- kdb-mountOdbc(1).
- elektra-mounting(7).

# Chapter 365

# kdb-mountOdbc(1) - Mount an ODBC data source to the key database

## 365.1 SYNOPSIS

```
kdb mountOdbc <data source name> <user name> <password> <table name> <key
column name> <value column name> <meta table name> <mt key column name>
<mt metakey column name> <mt metavalue column name> <timeout (s)> <mountpoint>
```

- Where `data source name` is the name of the ODBC data source as defined in `odbc.ini` (usually stored at `/etc/unixODBC/`).
  - Only ODBC drivers that support **outer joins** are suited for the Elektra ODBC backend.
- `user name`: The username that should be used for connecting to the ODBC data source. If no username is needed, or it is already specified in the ini-file for the configuration of the datasource, `""` can be provided for this argument.
- `password`: The same rules as for `user name` also apply to this argument.
- `table name`: The name of the table in the data source where the keys and their values should be stored.
- `key column name`: The name of the column where the key-names should be stored. This should be the primary key of the table.
- `value column name`: The name of the column where the key-values (strings) should be stored. This column should support NULL-values.
- `meta table name`: The name of the table where the metadata for the keys should be stored.
- `mt key column name`: The name of the column in the meta table where the key-name should be stored. This should be a foreign key that refers to the column with the key-name of the other table.
- `mt metakey column name`: The name of the column in the meta table where the name of the metakey should be stored. This column, together with the column for the key-name, should define the primary key of the meta table.
- `mt metavalue column name`: The name of the column in the meta table where the value (string) of the metakey should be stored. This column should support NULL-values.
- `timeout (s)`: The timeout (in seconds) that should be used when connecting to the data source. When passing '0', the timeout gets disabled and the application could potentially wait forever. So use this option with care! If you want to use a default timeout, you can just pass `""` for this argument.
- `mountpoint`: The place in the KDB where the ODBC data source should be mounted. The syntax is the same as with the file-based backend, but you can only use `user:/` and `system:/` namespaces as mountpoints for ODBC data sources.

All columns in the data source that are specified via one of the described arguments, should be of type `TEXT`, `CHAR` or `VARCHAR`. The tables can have additional columns. These are not processed by the ODBC backend, but should support `NULL` or `DEFAULT` values.

## 365.2 DESCRIPTION

This command allows a user to persistently mount a new *backend* that refers to an ODBC data source. The concept is the same as for mounting with the file-based backend. So it is recommended to also read to man page for kdb-mount(1). More about mounting is explained in elektra-mounting(7).

## 365.3 IMPORTANT

This command modifies `system:/elektra/mountpoints`. Depending on the location of that part of the KDB, you may need root privileges to execute this command. Use `kdb file system:/elektra/mountpoints` to find out where exactly it will write to.

There is no special command for unmounting ODBC data sources. You can just use the kdb-umount(1) command the same way as with mountpoints for files.

## 365.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional information in case of errors/warnings.
- `-f`, `--force`: Create the mountpoint even if the data source can not be reached at mount-time.

## 365.5 Examples

To mount the sample SQLite data source which is available at /src/plugins/backend_odbc/sampleDb/elektraDB.db:

```
kdb mountOdbc Selektra "" "" elektraKeys keyName keyValue metaKeys keyName
metaKeyName metaKeyValue "" user:/odbcSqlite
```

To specify all arguments:

```
kdb mountOdbc Selektra myUser myPassword elektraKeys keyName keyValue meta
Keys keyName metaKeyName metaKeyValue 12 user:/odbcSqlite
```

## 365.6 SEE ALSO

- kdb-mount(1).
- elektra-glossary(7).
- kdb-spec-mount(1).
- kdb-umount(1).
- elektra-mounting(7).

# Chapter 366

# kdb-mountpoint-info – Print information about the default storage and resolver or a mount point

## 366.1 SYNOPSIS

```
kdb mountpoint-info [mount point]
```

## 366.2 DESCRIPTION

This command will print information about the version, default resolver and default storage plugin. When a mount point is given as argument additional information about the mount point (e.g. configuration) is printed.

## 366.3 EXAMPLES

```
kdb mountpoint-info
#> Version: 0.8.19
#> Default resolver: resolver_fm_hpu_b
#> Default storage: ini
kdb mount /tmp/test.ini system:/test ini hello=ini -c hello=world
kdb mountpoint-info system:/test
#> Version: 0.8.19
#> Default resolver: resolver_fm_hpu_b
#> Default storage: ini
#> Mountpoint: system:/test
#> File: /tmp/test.ini
#>         config:
#>         hello = world
#>         path = /tmp/test.ini
#> getplugins:
#>         #0#resolver
#>         #5#ini#ini#
#>                 config:
#>                 hello = ini
#> setplugins:
#>         #0#resolver
#>         #5#ini
#>         #6#sync#sync#
#>         #7#resolver
#> errorplugins:
#>         #5#resolver_fm_hpu_b#resolver#
```

# Chapter 367

# kdb-mv(1) – Move keys within the key database

## 367.1 SYNOPSIS

`kdb mv <source> <dest>`
Where `source` is the path of the key(s) you want to copy and `dest` is the path where you would like to move the key(s) to. Note that when using the `-r` flag, `source` as well as all of the keys below it will be moved.

## 367.2 DESCRIPTION

This command moves key(s) in the Key database. You can move keys to other paths within the database.

## 367.3 LIMITATIONS

Neither `source` nor `dest` can be a cascading key. (Start with `/`). Make sure to select a namespace.

## 367.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 1-10: standard exit codes, see kdb(1)
- 11: No key to move found.

## 367.5 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-r`, `--recursive`: Work in a recursive mode.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 367.6 EXAMPLES

To move multiple keys:

```
kdb mv -r user:/example1 user:/example2
```

To move a single key:

```
kdb mv user:/example/key1 user:/example/key2
```

# Chapter 368

# kdb-namespace(1) – Get the namespace of a key name

## 368.1 SYNOPSIS

`kdb namespace <key name>`
Where `key name` is the name of the key.

## 368.2 DESCRIPTION

This command is used to retrieve the namespace of a key, including the eventual trailing ':'. For the cascading namespace, the empty string is returned.

## 368.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 7: invalid key name passed, see kdb(1) for a list of return codes used by kdb.

## 368.4 OPTIONS

- `-n`, `--no-newline`: Suppress the newline at the end of the output.

## 368.5 EXAMPLES

```
kdb namespace user:/key/subkey
#> user:
kdb namespace /key/subkey
#>
```

## 368.6 SEE ALSO

- See kdb-basename(1) and kdb-dirname(1) for other namepart extraction operations.
- To get keys in shell scripts, you can use kdb-sget(1).
- elektra-key-names(7) for an explanation of key names.

# Chapter 369

# kdb-plugin-check(1) – Perform internal checks

## 369.1 SYNOPSIS

```
kdb plugin-check [<plugin>]
```

## 369.2 DESCRIPTION

This command is used to perform checks on the key database or an Elektra plugin.
Where the option argument, `plugin` is the plugin that a user wants to check. Use `-c` to pass options to that plugin. If no `plugin` argument is provided a check will be performed on the key database itself. Special values are returned upon exit to represent the outcome of a check.

## 369.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-f`, `--force`: The user can also use this tool to perform write tests. Please note that this can result in configuration files being changed!
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-c`, `--plugins-config <plugins-config>`: Add a plugin configuration in addition to `/module`.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 369.4 RETURN VALUES

There are two different types of checks, a check on a plugin (by specifying the name of a plugin as an argument) or a check on the key database itself.
The outcome of a check on the key database is returned as an exit status. This integer represents an 8-bit pattern. Each bit represents a specific outcome as described below:

- 0: No errors (no output)
- Bit 1: Warning on opening the key database.
- Bit 2: Error on opening the key database.
- Bit 3: Warning on getting the value of a key.

- Bit 4: Error on getting the value of a key.
- Bit 5: Warning on setting the value of a key. (only checked when `-f` is used)
- Bit 6: Error on setting the value of a key (only checked when `-f` is used)
- Bit 7: Warning on closing the key database.
- Bit 8: Error on closing the key database.

So if the following number was returned `9` the user could figure out more detail by considering the bits: `00001001`
The user would know that there was a warning on open and an error on get.
If a plugin name is given, checks will only be done on the given plugin. The returned values for a check on a plugin are returned as much simpler numbers.
Return values on plugin checking:

- 0: Everything ok. (no output)
- 11: No such plugin found or plugin could not be opened.
- 12: Plugin did not pass checks.
- 13: Plugin has warnings.

Please report any output caused by official plugins to `https://git.libelektra.org/issues`.
Since the error code is a return value, it is not automatically displayed to the shell. If the user wants to have the value printed, they must do so manually (by running a command such as `echo $?`).

## 369.5 EXAMPLES

To check the Key Database:
`kdb plugin-check`
To check the Key Database and then print the result:
`kdb plugin-check` followed by:
`echo $?`
To check the Key Database including write checks:
`kdb plugin-check -f` Note that this type of check may change configuration files.
To check the `line` plugin:
`kdb plugin-check line`

## 369.6 SEE ALSO

- For an introductions into plugins, read elektra-plugins.
- To list all plugins use kdb-plugin-list(1).
- For information on a plugin use kdb-plugin-info(1).

# Chapter 370

# kdb-plugin-info(1) – Print information about an Elektra plugin

## 370.1 SYNOPSIS

`kdb plugin-info <plugin> [<clause name>]`

Where `plugin` is the plugin in which the user would like to know information about.
The optional `clause name` argument can be used to just print information from a certain clause.

## 370.2 DESCRIPTION

This command will print out all the information about an Elektra plugin except it's configuration.
This command will also print out any functions that are exported by the plugin.
Information about a plugin will be read from `system:/elektra/modules/`. If the information could not be located there, such as when a plugin is not mounted, the module will be loaded dynamically and then the information will be requested directly from the plugin.
If a user wishes to load the information directly from the plugin, they can force that by using the `-l` option.

## 370.3 RETURN VALUES

This command returns the following exit statuses:

- 0:
  The command was successful.

- 11:
  A `clause name` was specified but not found.

## 370.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.

- `-l`, `--load`: Load plugin even if system:/elektra is available.
- `-c`, `--plugins-config <plugins-config>`: Add a plugin configuration.

## 370.5 EXAMPLES

To print all the information about the `dump` plugin:
```
kdb plugin-info dump
```

To print out the license of the `resolver` plugin directly by forcing it to load:
```
kdb plugin-info -l resolver licence
```

To print out the author of the `line` plugin:
```
kdb plugin-info line author
```

# Chapter 371

# kdb-plugin-list(1) – List plugins available to Elektra

## 371.1 SYNOPSIS

```
kdb plugin-list [provider]
```

## 371.2 DESCRIPTION

This command will either list all available Elektra plugins or all plugins that provide a specific functionality.
The output will be sorted alphabetically unless option `-v` is passed. With option `-v` the output will be sorted by their status, the best plugins will be the last in the list.

## 371.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Also output the number calculated by their `infos/status` clause in the contract. Sorts the output ascending by the calculated status. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-0`, `--null`: Use binary 0 termination

## 371.4 EXAMPLES

To get a sorted list of all available plugins with their status:
`kdb plugin-list -v`
To get a sorted list of all storage plugins:
`kdb plugin-list storage`
To get a sorted list of all plugins that provide `ini` with their status:
`kdb plugin-list -v ini`

## 371.5 SEE ALSO

- `infos/status` is part of the elektra-contracts.md (7)

# Chapter 372

# kdb-record-export(1) – Export recorded changes

## 372.1 SYNOPSIS

```
kdb record-export [<source>] [<format>]
```

## 372.2 DESCRIPTION

This command exports the recorded changes into the specified output format. Keys are exported to `stdout` in whichever format is specified.

## 372.3 USAGE

Where `source` is the path of the key(s) you want to export. Additionally, the user can specify a format to use by passing it as the option argument `format`. The `format` attribute relies on Elektra's plugin system to export the keys in the desired format. The parameter `format` must be the name of a valid storage plugin.
Both `source` and `format` are optional parameters. By default, `source` is `/` and `format` is `ansible`. If you only specify a single argument, we will whether it is a key or the format.

## 372.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-E`, `--without-elektra`: Omit the `system:/elektra` directory.
- `-S`, `--include-recording-session`: Include the recording session in the output.
- `-c`, `--plugins-config <plugins-config>`: Add a configuration to the format plugin.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 372.5 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1).
- 11: An error occurred during exporting.

## 372.6 SEE ALSO

- kdb-record-start(1) on how to start the recording session
- kdb-record-stop(1) on how to stop the recording session
- kdb-record-state(1) on how to get information about the current recording session

# Chapter 373

# kdb-record-reset(1) – Reset the recording session

## 373.1 SYNOPSIS

```
kdb record-reset
```

## 373.2 DESCRIPTION

This command removes all the recorded changes in the KDB. It will NOT undo the changes themselves. This command can be executed while recording is active and when it is not active.

## 373.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 373.4 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1).
- 11: An error occurred during reset.

## 373.5 SEE ALSO

- kdb-record-start(1) on how to start the recording session
- kdb-record-stop(1) on how to stop the recording session
- kdb-record-state(1) on how to get information about the current recording session
- kdb-record-undo(1) on how to undo changes that have been recorded
- kdb-record-rm(1) on how to remove a single key from the recording session

# Chapter 374

# kdb-record-rm(1) – Remove a key from the recording session

## 374.1 SYNOPSIS

```
kdb record-rm <key>
```

## 374.2 DESCRIPTION

This command removes the specified `key` from the recording session. As far as the recording session is concerned, it's as if the key was never changed. If the key is changed again after it was removed from the session, it will be put back into the session.

## 374.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-r`, `--recursive`: Work in a recursive mode. Will also remove all keys below `key`.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 374.4 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1).s
- 11: An error occurred during removal of the key(s).

## 374.5 SEE ALSO

- kdb-record-start(1) on how to start the recording session
- kdb-record-stop(1) on how to stop the recording session

- kdb-record-state(1) on how to get information about the current recording session
- kdb-record-reset(1) on how to remove all keys from the recording session
- kdb-record-undo(1) on how to undo changes that have been recorded

# Chapter 375

# kdb-record-start(1) – Start session recording

## 375.1 SYNOPSIS

```
kdb record-start [<parent_key>]
```

## 375.2 DESCRIPTION

This command starts session recording. If there is already a previous session, this only changes the parent key. If you want to make sure you start at a clean state, use `kdb record-reset` first. By default, all changes to the KDB are recorded. The optional parameter `parent_key` can be used to restrict recording to a specific subtree of the KDB.
Note: when you activate session recording, concurrency of Elektra will be somewhat limited. As long as it is active, a global lock will be created to ensure no two processes will write data simultaneously. The errors reported to applications in such cases are similar to when multiple processes write to the same configuration file. Applications should already handle this case gracefully, and just retry writing their configuration. Therefore, activating recording may reduce performance slightly, but (assuming well-written applications) it should not cause application errors.

## 375.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 375.4 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1).
- 11: An error occurred during starting the recording.

## 375.5 SEE ALSO

- kdb-record-stop(1) on how to stop the recording session

- kdb-record-state(1) on how to get information about the current recording session
- kdb-record-reset(1) on how to clear the recording session

# Chapter 376

# kdb-record-state(1) – Print information about the state of a recording session

## 376.1 SYNOPSIS

```
kdb record-state
```

## 376.2 DESCRIPTION

This command prints out information about the current state of the recording session. This includes:

- Whether recording is currently enabled or not
- What parts of the KDB will be recorded in the future. (The session may already contain changes from other parts of the KDB)
- How many keys have been recorded
- Which keys have been recorded

For now, the output format is not considered stable and may change over time.

## 376.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 376.4 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1)
- 11: An error occurred during gathering the state information.

## 376.5 EXAMPLES

Here is an example output of `kdb record-state`:

```
Recording is active for user:/
Added 1 key(s)
Modified 1 key(s)
Removed 1 key(s)
Added key user:/test/age
Modified key user:/test/name
Removed key user:/test/color
```

## 376.6 SEE ALSO

- kdb-record-start(1) on how to start the recording session
- kdb-record-stop(1) on how to stop the recording session

# Chapter 377

# kdb-record-stop(1) – Stop session recording

## 377.1 SYNOPSIS

```
kdb record-stop
```

## 377.2 DESCRIPTION

This command stops the current recording session. Changes that are made to KDB after this command will no longer be recorded. The recording session will NOT be cleared. You can use `kdb record-start` to resume the session.

## 377.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 377.4 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1)
- 11: An error occurred during stopping the recording.

## 377.5 SEE ALSO

- kdb-record-start(1) on how to start the recording session
- kdb-record-state(1) on how to get information about the current recording session
- kdb-record-reset(1) on how to clear the recording session

# Chapter 378

# kdb-record-undo(1) – Undo the changes performed during a recording session

## 378.1 SYNOPSIS

```
kdb record-undo [<parent_key>]
```

## 378.2 DESCRIPTION

This command will undo all the changes that have been recorded. The optional parameter `parent_key` can be used to restrict the undo operation to a specific subtree of the KDB. After execution, the recording session will no longer contain the undone keys.

## 378.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 378.4 RETURN VALUE

- 0: Successful.
- 1-10: standard exit codes, see kdb(1)
- 11: An error occurred during undoing the recorded changes.

## 378.5 SEE ALSO

- kdb-record-start(1) on how to start the recording session
- kdb-record-state(1) on how to get information about the current recording session
- kdb-record-stop(1) on how to stop the recording session

# Chapter 379

# kdb-remount(1) - Use an existing backend to mount a new file

## 379.1 SYNOPSIS

`kdb remount <new path> <new mount point> <existing mount point>`
Where `new path` is the path to the file the user wants to mount, (Absolute for system files, relative for user files)
`new mount point` is where in the key database the new configuration file should be mounted at, (For a cascading mount pount, `mountpoint` should start with `/`)
and `existing mount point` is the mount point where the existing backend is mounted.

## 379.2 DESCRIPTION

This command allows a user to use an existing backend (such as one from a previous mount) to mount a new configuration file to a new mount point in the key database.

## 379.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information or ask debug questions (in interactive mode). Prints additional debug information in case of errors/warnings.
- `-i`, `--interactive`: Instead of passing all information by parameters ask the user interactively.

## 379.4 EXAMPLES

To mount a new file using an existing backend that is mounted to `system:/example`:
`kdb remount /etc/configuration2.ini system:/example2 system:/example`

# Chapter 380

# kdb-reset-elektra(1) - Resets system:/elektra

## 380.1 SYNOPSIS

```
kdb reset-elektra -f
```

## 380.2 DESCRIPTION

Resets all configuration changes that influence Elektra's core, i.e., keys in `system:/elektra`.

## 380.3 WARNING

These changes cannot be undone, please use with care! This command writes into the `/etc` directory and as such it requires root permissions.

## 380.4 OPTIONS

- `-H`, `--help`: Show the man page.

## 380.5 SEE ALSO

- kdb-backup(1).
- kdb-stash(1).
- elektra-mounting(7).

# Chapter 381

# kdb-reset(1) - Resets the whole KDB

## 381.1 SYNOPSIS

```
kdb reset -f
```

## 381.2 DESCRIPTION

Resets the whole KDB.

## 381.3 WARNING

These changes cannot be undone, please use with care! This command writes into the `/etc` directory and as such it requires root permissions.

## 381.4 OPTIONS

- `-H`, `--help`: Show the man page.

## 381.5 SEE ALSO

- kdb-backup(1).
- kdb-stash(1).
- elektra-mounting(7).

# Chapter 382

# kdb-restore – Restore from backup

## 382.1 SYNOPSIS

`kdb restore <timestamp>`

## 382.2 DESCRIPTION

This command will restore everything backed up by `kdb backup` (or stashed by `kdb stash`) where `timestamp` is the timestamp returned by `kdb backup` (or `kdb stash`).

## 382.3 WARNING

These changes cannot be undone, please use with care! This command writes into the `/etc` directory and as such it requires root permissions.

## 382.4 EXAMPLES

```
kdb set user:/tests/x foo
#>
#> Create a new key user:/tests/x with string "foo"
sudo kdb mount a.ini /a
sudo kdb mount
#> a.ini on /a with name /a
#> none on system:/info/elektra/constants with name system:/info/elektra/constants
#> /usr/local/share/doc/elektra/CONTRACT.ini on system:/info/elektra/contract/#0 with name
       system:/info/elektra/contract/#0
#> none on system:/info/elektra/desktop with name system:/info/elektra/desktop
#> /usr/local/share/doc/elektra/METADATA.ini on system:/info/elektra/metadata/#0 with name
       system:/info/elektra/metadata/#0
#> none on system:/info/elektra/uname with name system:/info/elektra/uname
sudo kdb backup
#> kdb restore 1500000000
kdb get user:/tests/x
#> Did not find key
kdb mount
kdb restore 1500000000
kdb get user:/tests/x
#> foo
kdb mount
#> a.ini on /a with name /a
#> none on system:/info/elektra/constants with name system:/info/elektra/constants
#> /usr/local/share/doc/elektra/CONTRACT.ini on system:/info/elektra/contract/#0 with name
       system:/info/elektra/contract/#0
#> none on system:/info/elektra/desktop with name system:/info/elektra/desktop
#> /usr/local/share/doc/elektra/METADATA.ini on system:/info/elektra/metadata/#0 with name
       system:/info/elektra/metadata/#0
#> none on system:/info/elektra/uname with name system:/info/elektra/uname
sudo kdb umount /a
```

## 382.5 SEE ALSO

- kdb-backup(1)

- kdb-stash(1)

# Chapter 383

# kdb-rm(1) – Remove key(s) from the key database

## 383.1 SYNOPSIS

`kdb rm <path>`
Where `path` is the path of the key(s) you want to remove. Note that when using the `-r` flag, not only the key directly at `path` will be removed, but all of the keys below the path as well.

## 383.2 DESCRIPTION

This command removes key(s) from the Key database.

## 383.3 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 1-10: standard exit codes, see kdb(1)
- 11: No key to remove found.

## 383.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-r`, `--recursive`: Work in a recursive mode.
- `-E`, `--without-elektra`: Omit the `system:/elektra` directory.
- `-f`, `--force`: Do not fail on missing key, nor print if there was a key (-v to still print).
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.

## 383.5 EXAMPLES

To remove a single key:
`kdb rm user:/example/key1`
To remove multiple keys:
`kdb rm -r user:/example`
To remove all keys in `system` except `system:/elektra`:
`sudo kdb rm -rE system`
To not fail when key is missing:
`kdb rm -f user:/maybe/missing`

## 383.6 SEE ALSO

- elektra-key-names(7) for an explanation of key names.

# Chapter 384

# kdb-set(1) – Set the value of a key

## 384.1 SYNOPSIS

`kdb set <key name> <value>`
Where `key name` is the name of the key you wish to set the value of (or create) and `value` is the value you would like to set the key to.

## 384.2 DESCRIPTION

This command allows the user to set the value of an individual key. If a cascading key is used that does not resolve to an existing key, the operation is aborted as it is ambiguous.

## 384.3 EMPTY VALUES

To set a key to an empty value, **""** should be passed for the `value` argument.

## 384.4 NEGATIVE VALUES

To set a key to a negative value, `--` has to be used to stop option processing. (see example below)

## 384.5 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-q`, `--quiet`: Suppress non-error messages.
- `--`: Do not process any following arguments starting with `-` as options.
- `-v`, `--verbose`: Explain what is happening. Prints additional information in case of errors/warnings.
- `-d`, `--debug`: Give debug information. Prints additional debug information in case of errors/warnings.
- `-f`, `--force`: Do not perform a cascading lookup if the key provided has a namespace. For example, this bypasses validation specified in the spec: namespace for the given key.

## 384.6 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 11: Not a valid keyname.
- 12: Setting a non-existing key without a namespace is not possible.

## 384.7 KDB

- `/sw/elektra/kdb/#0/current/verbose`: Same as `-v`: Explain what is happening.
- `/sw/elektra/kdb/#0/current/quiet`: Same as `-q`: Suppress default messages.

## 384.8 EXAMPLES

To set a Key to the value `Hello World!`:

```
kdb set user:/example/key "Hello World!"
```

To set a key to an empty value:

```
kdb set user:/example/key ""
```

To set a key to a negative value:

```
kdb set -- /tests/neg -3
```

To create bookmarks:

```
kdb set user:/sw/elektra/kdb/#0/current/bookmarks ""
```

Followed by:

```
kdb set user:/sw/elektra/kdb/#0/current/bookmarks/kdb user:/sw/elektra/kdb/#0/current
```

## 384.9 SEE ALSO

- kdb(1) for how to configure the kdb utility and use the bookmarks.
- elektra-key-names(7) for an explanation of key names.

# Chapter 385

# kdb-sget(1) – Get the value of a key stored in the key database from a script

## 385.1 SYNOPSIS

`kdb sget <key name> <default-value>`
Where `key name` is the name of the key to retrieve and `default-value` is the value that should be printed if no value can be retrieved.

## 385.2 DESCRIPTION

This command is used to retrieve the value of a key from within a script. When using the kdb tool in a script, the user should use the `sget` command in place of the kdb-get(1) command. The kdb-get(1) command should not be used in scripts because it may return an error instead of printing a value in certain circumstances. The `sget` command guarantees that a value will be printed (unless the user passes faulty arguments). This command will either print the value of the key it retrieves or a default value that the user specifies.

## 385.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.

## 385.4 EXAMPLES

To get the value of a key from a script or return the value `0`:
`kdb sget user:/example/key 0`
To get the value of a key using a cascading lookup or return the value `notfound`:
`kdb sget /example/key "notfound"`

## 385.5 SEE ALSO

- kdb-get(1)
- elektra-key-names(7) for an explanation of key names.

# Chapter 386

# kdb-shell(1) – Start a kdb shell instance

## 386.1 SYNOPSIS

```
kdb shell
```

## 386.2 DESCRIPTION

This command is used to start an instance of the kdb shell.
The kdb shell allows for a user to interactively view, edit, or otherwise work with the key database.

## 386.3 SHELL COMMANDS

The kdb shell offers a number of commands to interact with the key database.

- `kdbGet <name>`: Get the value of a key.
- `kdbSet <name>`: Set the value of a key.
- `keySetName <name>`: Set the name of the current key.
- `keySetMeta <name> <string>`: Set a metakey associated with the current key.
- `keySetString <string>`: Set a string value for the current key.
- `ksAppendKey`: Append the current key to the current keyset.
- `ksCut <name>`: Cut the current keyset.
- `ksOutput`: Prints the keys in the current keyset.

## 386.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.

## 386.5 EXAMPLES

To execute commands from a textfile, you can use:
```
cat commands.txt | kdb shell
```
To have readline functionality (line editing, history, ...), you can use:
```
rlwrap kdb shell
```

## 386.6 SEE ALSO

To learn more about these commands and how they work, refer to the Elektra API Documentation.

# Chapter 387

# kdb-spec-mount(1) - Mount a spec file to the key database

## 387.1 SYNOPSIS

`kdb spec-mount [/<mountpoint>] [<plugin> [<config>] [..]]`

- `mountpoint` is where in the key database the new backend should be mounted to. It must be a cascading mount point, i.e., `mountpoint` must start with `/`.
- `plugin` are extra Elektra plugins to be used (next to the one specified in `spec:/<mountpoint>`).
- Plugins may be followed by a `,` separated list of `keys=values` pairs which will be used as plugin configuration.

## 387.2 DESCRIPTION

This command allows a user to mount a new *backend* described by a previously mounted specification. To mount a specification file to `spec`-namespace first use kdb-mount(7):

```
sudo kdb mount /path/to/some-spec-file.ini spec:/example/mountpoint ni
```

The idea of mounting is explained in elektra-mounting(7). The `spec` namespace contains metaconfiguration that describes the configuration in all other namespaces. The metadata used for the specification is described in `METADATA.ini`.

During `spec-mount` the `spec` keys are searched for relevant metadata:

- For every metadata `mountpoint` an additional cascading mount point will be mounted. The metadata `mountpoint` is usually at the parent key (the top-level key of the specification).
- The `infos/*` and `config/needs` from `CONTRACT.ini`, that are tagged by `usedby = spec`, will work as described there.
- For other metadata suitable plugins are searched and mounted additionally.

For example:

```
kdb meta-get spec:/example/mountpoint mountpoint  # verify that we have a mountpoint here
sudo kdb spec-mount /example/mountpoint  # mounts /example/mountpoint according to
                                         # the specification as found at
                                         # spec:/example/mountpoint
```

## 387.3 IMPORTANT

This command writes into the `/etc` directory and as such it requires root permissions. Use `kdb file system:/elektra/mountpoints` to find out where exactly it will write to.

## 387.4 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Explain what is happening.
- `-q`, `--quiet`: Suppress non-error messages.
- `-R`, `--resolver <resolver>`: Specify the resolver plugin to use if no resolver is given, the default resolver is used. See also `below in KDB`. Note that the resolver will only added as dependency, but not directly added.
- `-c`, `--plugins-config <plugins-config>`: Add a plugin configuration for all plugins.
- `-W`, `--with-recommends`: Also add recommended plugins and warn if they are not available.

## 387.5 KDB

- `/sw/elektra/kdb/#0/current/verbose`: Same as `-v`: Explain what is happening.
- `/sw/elektra/kdb/#0/current/quiet`: Same as `-q`: Suppress default messages.
- `/sw/elektra/kdb/#0/current/resolver`: The resolver that will be added automatically, if `-R` is not given.
- `/sw/elektra/kdb/#0/current/plugins`: It contains a space-separated list of plugins and their configs which are added automatically (by default sync). The plugin-configuration syntax is as described above in the `synopsis`.

## 387.6 EXAMPLES

To mount /example as described in `spec:/example`:

```
sudo kdb spec-mount /example
```

Additionally, add `ini` plugin (instead of some default resolver) with a config for INI:

```
sudo kdb spec-mount /example ini section=NULL
```

## 387.7 SEE ALSO

- elektra-glossary(7).
- kdb-umount(7).
- elektra-mounting(7).
- see application integration tutorial
- see validation tutorial

# Chapter 388

# kdb-stash – Stash away KDB to be restored later

## 388.1 SYNOPSIS

`kdb stash`

## 388.2 DESCRIPTION

This command will stash away the `system`, `user` and `spec` configuration, i.e. it releases all mount points and reset Elektra to a clean state. Afterwards a timestamp which can be used to restore everything will be printed. The backup will be done in the `/var/tmp` directory, so make sure the backup is not deleted.

## 388.3 EXAMPLES

```
kdb set user:/x foo
#>
#> Create a new key user:/x with string "foo"
kdb mount a.ini /a
kdb mount
#> a.ini on /a with name /a
#> none on system:/info/elektra/constants with name system:/info/elektra/constants
#> /usr/local/share/doc/elektra/CONTRACT.ini on system:/info/elektra/contract/#0 with name
       system:/info/elektra/contract/#0
#> none on system:/info/elektra/desktop with name system:/info/elektra/desktop
#> /usr/local/share/doc/elektra/METADATA.ini on system:/info/elektra/metadata/#0 with name
       system:/info/elektra/metadata/#0
#> none on system:/info/elektra/uname with name system:/info/elektra/uname
kdb backup
#> kdb restore 1500000000
kdb get user:/x
#> Did not find key
kdb mount
kdb restore 1500000000
kdb get user:/x
#> foo
kdb mount
#> a.ini on /a with name /a
#> none on system:/info/elektra/constants with name system:/info/elektra/constants
#> /usr/local/share/doc/elektra/CONTRACT.ini on system:/info/elektra/contract/#0 with name
       system:/info/elektra/contract/#0
#> none on system:/info/elektra/desktop with name system:/info/elektra/desktop
#> /usr/local/share/doc/elektra/METADATA.ini on system:/info/elektra/metadata/#0 with name
       system:/info/elektra/metadata/#0
#> none on system:/info/elektra/uname with name system:/info/elektra/uname
```

## 388.4 SEE ALSO

- kdb-restore(1)
- kdb-backup(1)

# Chapter 389

# kdb-test(1) – Run test(s) on the key database

## 389.1 SYNOPSIS

`kdb test <path> [<test-name> ...]`

Where `path` is the path the user wishes to perform the test under. The option `test-name` argument is used to specify which test(s) to run. To run multiple tests, each should be named with a trailing space.
If no `test-name` is provided, all the tests will be run.

## 389.2 DESCRIPTION

This command is used to run part or all of the key database test suite.
These tests allow one to user to verify that a backend is capable of storing and retrieving all kinds of configuration keys and values.

The following tests are available: basic string umlauts binary naming meta

## 389.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.

## 389.4 EXAMPLES

To run all tests below the `user:/tests/example` key:
`kdb test user:/tests/example`

To run the `binary` and `naming` tests:

```
kdb test user:/tests/example binary naming
# RET: 0
# clean-up
kdb rm -r user:/tests/example
```

## 389.5 SEE ALSO

- elektra-key-names(7) for an explanation of key names.

# Chapter 390

# kdb-umount-all(1) - Unmount everything in the key database

## 390.1 SYNOPSIS

```
kdb umount-all -f
```

## 390.2 DESCRIPTION

Unmount all backend and all global plugins from key database.

## 390.3 WARNING

These changes cannot be undone, please use with care! This command writes into the `/etc` directory and as such it requires root permissions.

## 390.4 OPTIONS

- `-H`, `--help`: Show the man page.

## 390.5 SEE ALSO

- kdb-mount(1).
- elektra-mounting(7).

# Chapter 391

# kdb-umount(1) - Unmount a file from the key database

## 391.1 SYNOPSIS

```
kdb umount <name>
```

## 391.2 DESCRIPTION

Unmount backend from key database. This command writes into the `/etc` directory and as such it requires root permissions.

## 391.3 OPTIONS

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.
- `-v`, `--verbose`: Show which mount points were considered.

## 391.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: No errors.
- 11: Mountpoint does not exist.

## 391.5 SEE ALSO

- kdb-mount(1).
- elektra-mounting(7).

# Chapter 392

# kdb-validate(1) - Validate key values

## 392.1 SYNOPSIS

```
kdb validate
```

## 392.2 DESCRIPTION

Validate the values of string keys below a given name using the loaded validation plugins (eg. range or validation) by reading all values, making them dirty by changing to another value, changing back to original and then writing that back to the key database.
This command is useful for validating configuration files against their specifications.
For keys to be validated, they must contain the 'check'-metakeys and the respective plugins for validation must be loaded for the backend that was used while mounting. If a validation is done while using `kdb set` or `kdb get` the same validation is also done by `kdb validate` Only string keys are validated! Binary keys are skipped!
Use `-f` to do a write-test even if the previous read from the key database has issued warnings.

## 392.3 OPTIONS

- `-d`,`--debug`: Give debug information.
- `-f`, `--force`: Force writing the configuration even on warnings.
- `-H`, `--help`: Show the man page.
- `-p <name>`, `--profile <name>`: Use a different profile for kdb configuration.
- `-v`, `--verbose`: Explain what is happening.
- `-V`, `--version`: Print version info.
- `-C <when>`, `--color <when>`: Print `never/auto(default)/always` colored output.

## 392.4 RETURN VALUES

This command will return the following values as an exit status:

- 0: Validated correctly.
- 11: Could not get the values.
- 12: Error occurred while writing back the values.

# Chapter 393

# kdb(1) – key database access tools

## 393.1 INTRODUCTION

**kdb** provides access to the global key database of Elektra via command-line.
Concepts are in man page section 7 and are prefixed with `elektra-`. If you do not yet know about Elektra, you should start reading elektra-introduction(7). CLI Tools of Elektra are in man page section 1 and are prefixed with `kdb-`.
The man pages can also be viewed online at: https://doc.libelektra.org/api/latest/html/pages.html
And the page you are currently reading at: https://doc.libelektra.org/api/latest/html/doc_help_kdb_md.html

## 393.2 OVERVIEW

In this manual we give an overview of the tool suite `kdb`. It is part of Elektra's tools. The tool suite `kdb` consists of individual commands. Most commands are independent and some commands are sharing an executable. Some commands are written as external scripts.
The included commands can be listed via:
`kdb`
External commands can be listed via:
`kdb list-tools`
Only a few commands are enough for daily use. We can retrieve a key by:
`kdb get user:/key`
We store a key permanently with a value given by:
`kdb set user:/key value`
We list all available keys arranged below a key by:
`kdb ls user:/key`
Documentation of plugins is available using the kdb-plugin-info(1) tool:
`kdb plugin-info`
Run `kdb plugin-list` for a list of plugins:
`kdb plugin-list`
Each of these commands has its own man page for further details.

## 393.3 BASIC OPTIONS

Every core-tool has the following options:

- `-H`, `--help`: Show the man page.
- `-V`, `--version`: Print version info.
- `-p`, `--profile <profile>`: Use a different kdb profile, see below.
- `-C`, `--color <when>`: Print never/auto(default)/always colored output.

- `--`: Do not process any following arguments starting with `-` as options.

## 393.4 COMMON OPTIONS

Most tools have the following options:

- `-v`, `--verbose`: Explain what is happening. Also shows Configfile and Mountpoint in case of an error/warning
- `-q`, `--quiet`: Suppress non-error messages.
- `-d`, `--debug`: Shows the line at which an error happened in case an error or warning is issued

## 393.5 KDB

The `kdb` utility reads its own configuration from three sources within the KDB (key database):

1. /sw/kdb/**profile**/ (for legacy configuration)
2. /sw/elektra/kdb/#0/%/ (for empty profile, `%` in Elektra is used to represent emptiness)
3. /sw/elektra/kdb/#0/**profile**/ (for current profile, if no `-p` or `--profile` is given, `current` will be used)

Here the last source where a configuration value is found, wins.
For example, to permanently change verbosity one can use:

- `kdb set /sw/elektra/kdb/#0/current/verbose 1`: Be verbose for every tool.
- `kdb set /sw/elektra/kdb/#0/current/quiet 1`: Be quiet for every tool.

If `%` is passed to `-p` or `--profile`, the KDB will not be consulted for configuration and only the command-line arguments are used.

## 393.6 PROFILES

Profiles allow users to change many/all configuration settings of a tool at once. It influences from where the KDB entries are read. Without a `-p` or `--profile` argument following profiles are used (in the order of preference):

- `current`: Is the profile to be used only if no `-p` argument was used.
- `%`: Is the fallback profile. It will be used if keys cannot be found in the main profile.

For example if you use:
`kdb export -p admin system`
It will read its format configuration from `/sw/elektra/kdb/#0/admin/format`.
If you want different configuration per user or directories, you should prefer to use the `user` and `dir` namespaces. Then the correct configuration will be chosen automatically according to the current user or current working directory.
Sometimes it is useful to start with default options, for example it is not possible to invert the `-q` option. In such situations one can simply select a non-existing profile, then `-q` works as usual:
`kdb mount -p nonexist -q /abc dir:/abc`
If `%` is used as profile name for `-p`, the `kdb` tools disables reading from `KDB` for their own configuration settings. Then, they only use command-line arguments.
To explicitly state the default behavior, we use:
`-p current`
To state that we do not want to read any configuration settings for `kdb` from KDB, we use:
`-p %`

> Note that KDB will still be used to perform the actions you want to perform with the `kdb` tool.

## 393.7 BOOKMARKS

Elektra recommends to use rather long paths because it ensures flexibility in the future (e.g. to use profiles and have a compatible path for new major versions of configuration).
Long paths are, however, cumbersome to enter in the CLI. Thus one can define bookmarks. Bookmarks are keys whose key name starts with `+`. They are only recognized by the `kdb` tool or tools that explicitly have support for it. Your applications should not depend on the presence of a bookmark.
Bookmarks are stored below:
`/sw/elektra/kdb/#0/current/bookmarks`
For every key found there, a new bookmark will be introduced.
Bookmarks can be used to start key names by using + (plus) as first character. The string until the first / will be considered as bookmark.
For example, if you set the bookmark kdb:

```
kdb set user:/sw/elektra/kdb/#0/current/bookmarks
kdb set user:/sw/elektra/kdb/#0/current/bookmarks/kdb user:/sw/elektra/kdb/#0/current
```

You are able to use:

```
kdb ls +kdb/bookmarks
kdb get +kdb/format
```

## 393.8 RETURN VALUES

- 0: successful.
- 1: Invalid options passed.
- 2: Invalid arguments passed.
- 3: Command terminated unsuccessfully without specifying error code.
- 4: Unknown command.
- 5: KDB Error, could not read/write from/to KDB.
- 6: Reserved error code.
- 7: Unknown errors, wrong exceptions thrown.
- 8-10: Reserved error codes.
- 11-20: Command-specific error codes. See man page of specific command.

## 393.9 SEE ALSO

- elektra-introduction(7)
- kdb(1)
- Get a big picture about Elektra

# Chapter 394

# Ideas for Contributing to Elektra

Thank you for being interested in contributing to Elektra!

## 394.1 How Can I Get Started?

We prepared `good first issues` for you.

- We encourage you to improve documentation.
- In the source code, you should look into libs and plugins.
- You can always peek into the TODOs, if you don't know what to do.
- An ideal project is to elektrify some free software.

Elektrify means:

- Patch the application so that it uses Elektra as its configuration system.
- Write a specification that describes how the configuration of the application looks like.

If you are interested in directly improving Elektra, other projects might be of more interest to you:

- Further configuration file formats could be supported.
- The tools could be improved.

## 394.2 What are the Requirements to Participate?

Anyone can join!
First you should familiarize with the Elektra Initiative:

- If you did not yet look at the `home page` please do so!
- Look into the contributing guidelines
- Look into the `issue tracker` and pick an easy task.
- Say hello in the issue you are interested in participating, or create a new issue.
- Create a PR which solves the issue.
- We will give you further feedback.

# Chapter 395

# Rendered Images

Rendered images and logos can be found in https://github.com/ElektraInitiative/blobs

## 395.1 SVG Conversion

https://github.com/shakiba/svgexport

```
svgexport circle_with_shadows_and_background.svg circle.png 400:400
```

# Chapter 396

# Install

## 396.1 Status

The graph below shows an (incomplete) list of available packages for Elektra.

## 396.2 Linux

For the following Linux distributions and package managers 0.9 packages are available:

- Arch Linux (AUR)
- Openwrt
- LinuxBrew

### 396.2.1 Debian/Ubuntu

We provide repositories for latest releases and latest builds from master (suite postfixed with `-unstable`) for following Debian-based distributions:

- Debian Bullseye
- Debian Buster
- Ubuntu Focal
- Ubuntu Bionic

To use our stable repositories with our latest releases, following steps need to be made:

1. Run `sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys F26BBE02F3C315A19BF1F791A9A25CC1CC83E839` to obtain the key.
2. Add `deb https://debs.libelektra.org/<DISTRIBUTION> <SUITE> main` into `/etc/apt/sources.list` where `<DISTRIBUTION>` and `<SUITE>` is the codename of your distributions e.g. `focal`,`bionic`,`buster`, etc.

   This can also be done using:

```
# Example for Ubuntu Focal
sudo apt-get install software-properties-common apt-transport-https
echo "deb https://debs.libelektra.org/focal focal main" | sudo tee /etc/apt/sources.list.d/elektra.list
```

Or alternatively, you can use (if you do not mind many dependences just to add one line to a config file):

```
# Example for Ubuntu Focal
sudo apt-get install software-properties-common apt-transport-https
sudo add-apt-repository "deb https://debs.libelektra.org/focal focal main"
```

If you would like to use the latest builds of master, append `-unstable` to `<SUITE>`.
The `etc/apt/source.list` entry must look like following: `deb https://debs.libelektra.org/<DISTRIBUTION> <SUITE>-unstable main`
E.g. `deb https://debs.libelektra.org/focal focal-unstable main`

1. Run `sudo apt-get update`.

   NOTE: for Ubuntu Bionic the yamlcpp plugin is excluded due to missing dependencies and therefore the package `libelektra5-yamlcpp` is not available.

### 396.2.2 Fedora

We provide repositories for latest releases and latest builds from master (suite postfixed with `-unstable`) for Fedora 33 and Fedora 34
For our stable repository with our latest releases:

```
wget https://rpms.libelektra.org/fedora-34/libelektra.repo -O libelektra.repo;
sudo mv libelektra.repo /etc/yum.repos.d/;
sudo yum update
```

Or alternatively you can use dnf to add this repo:

```
sudo dnf config-manager --add-repo https://rpms.libelektra.org/fedora-34/libelektra.repo
```

For our latest builds from master append `-unstable` to the suite name:

```
wget https://rpms.libelektra.org/fedora-34-unstable/libelektra.repo -O libelektra.repo;
sudo mv libelektra.repo /etc/yum.repos.d/;
sudo yum update
```

Or alternatively you can use dnf to add this repo:

```
sudo dnf config-manager --add-repo https://rpms.libelektra.org/fedora-34-unstable/libelektra.repo
```

### 396.2.3 openSUSE

We provide repositories for latest releases and latest builds from master (suite postfixed with `-unstable`) for openSUSE Leap 15.3
For our stable repository with our latest releases:

NOTE: stable packages will be available as of the `0.9.8` release.

```
sudo zypper ar -f https://rpms.libelektra.org/opensuse-leap-15.3 libelektra
sudo zypper update
```

For our latest builds from master append `-unstable` to the suite name:

```
sudo zypper ar -f https://rpms.libelektra.org/opensuse-leap-15.3-unstable libelektra-unstable
sudo zypper update
```

### 396.2.4 Arch Linux

We provide an package for Arch Linux through the Arch User Repository
You can either install it using any AUR helper or manually using makepkg

#### 396.2.4.1 Manual install

Make you have `base-devel` and `git` installed (`sudo pacman -S base-devel git`)
Then you can install it using the following commands:

```
cd /tmp
git clone https://aur.archlinux.org/libelektra.git
cd libelektra
makepkg -si
```

### 396.2.5 Install

To get all packaged plugins, bindings and tools install:

```
# For Debian based distributions
apt-get install libelektra5-all
# For Fedora based distributions
dnf install libelektra5-all
# For openSUSE
zypper install libelektra5-all
```

For a small installation with command-line tools available use:

```
# For Debian based distributions
apt-get install elektra-bin
# For Fedora based distributions
dnf install elektra-bin
# For openSUSE
zypper install elektra-bin
```

To install all debugsym/debuginfo packages:

```
# For Debian based distributions
apt-get install elektra-dbg
```

```
# For Fedora based distributions
dnf install elektra-dbg
# For openSUSE
zypper install elektra-dbg
```

If you want to install individual debugsym/debuginfo packages:

```
# For Debian based distributions
apt-get install <packagename>-dbgsym # e.g. apt-get install libelektra5-dbgsym
# For Fedora based distributions
dnf debuginfo-install <packagename> # e.g. dnf debuginfo-install libelektra5
# For openSUSE
zypper install <packagename>-debuginfo # e.g. zypper install libelektra5-debuginfo
```

To build Debian/Ubuntu Packages from the source you might want to use:

```
make package # See CPack below
```

## 396.3 macOS

You can install Elektra using `Homebrew` via the shell command:

```
brew install elektra
```

. We also provide a tap containing a more elaborate formula `here`.

## 396.4 Windows

Installation for WSL is described here.
If you prefer native but in functionality limited version you can download MinGW `32-bit` and `64-bit` builds.
Otherwise please refer to the section `OS Independent` below.

## 396.5 OS Independent

First follow the steps in COMPILE.
After you completed building Elektra on your own, there are multiple options how to install it. For example, with make or CPack tools. We recommend using the packages from our build server or that you generate your own packages with CPack.

### 396.5.1 CPack

The current supported systems are: Debian, Ubuntu and Fedora.
Then use:

```
make package
```

which will create packages for distributions where a Generator is implemented.
You can find the generated packages in the `package` directory of the build directory.

> NOTE: If all plugins/bindings/tools a package includes are excluded, the package will not be generated.

#### 396.5.1.1 Debian/Ubuntu

First make sure you have `debhelper` and `d-shlibs` installed:

```
apt-get install debhelper d-shlibs
```

(Otherwise you'll see an error file utility is not available, breaking `CPACK_DEBIAN_PACKAGE_SHLIBDEPS` and `CPACK_DEBIAN_PACKAGE_GENERATE_SHLIBS`.)
On Debian-based distributions you will need to set `LD_LIBRARY_PATH` before generating the package. Simply `cd` into the build directory and run following command:

```
LD_LIBRARY_PATH=$(pwd)/lib:${LD_LIBRARY_PATH} make package -j2
```

To install the packages run this in the `package` directory:

```
sudo apt-get install ./*
```

If any dependency problems appear, run following command to install the missing dependencies:

```
sudo apt-get -f install
```

#### 396.5.1.2 Fedora

For packaging Fedora has `rpmdevtools` as an additional dependency:

```
sudo yum install -y rpmdevtools
```

When configuring CMake it is also required to set the `CMAKE_BUILD_TYPE` to `Release`. For instance, a minimal configuration is

```
cmake .. -DCMAKE_BUILD_TYPE="Release"
make package
```

To install RPM packages we recommend using `yum localinstall` since installing with `rpm` doesn't resolve missing dependencies.

Run following command in the `build/package` directory:

```
sudo yum localinstall *
```

Depending on the system configuration one might also need to configure and run `ldconfig`, for instance

```
sudo echo "/usr/local/lib" » /etc/ld.so.conf
sudo ldconfig
```

### 396.5.2 make

```
sudo make install
sudo ldconfig  # See troubleshooting below
```

To uninstall Elektra use (will not be very clean, e.g. it will not remove directories and `*.pyc` files):

```
sudo make uninstall
sudo ldconfig
```

or in the build directory (will not honor `DESTDIR`!):

```
xargs rm < install_manifest.txt
```

## 396.6 Troubleshooting

### 396.6.1 Error Loading Libraries

If you encounter the problem that the library can not be found (output like this)

```
kdb: error while loading shared libraries:
     libelektra-core.so.4: cannot open shared object file: No such file or directory
```

or:

```
kdb: error while loading shared libraries:
     libelektratools.so.2: cannot open shared object file: No such file or directory
```

you need to place a configuration file at `/etc/ld.so.conf.d/` (e.g. `/etc/ld.so.conf.d/elektra.conf`). Note that under Alpine Linux this file is called `/etc/ld-musl-x86_64.path` or similar, depending on your architecture.

Add the path where the library has been installed (on Alpine Linux this had to be `usr/lib/elektra` for it to work)

```
/usr/lib/local/
```

and run `ldconfig` as root.

## 396.7 Installation Manuals

For some plugins and tools that ship with Elektra, additional installation manuals have been written. You can find them in the tutorial overview.

## 396.8 See Also

- COMPILE.
- TESTING.

# Chapter 397

# Key Names in Elektra

This document is a full explanation of how *key names* work in Elektra.

## 397.1 0. Reference Implementation

In addition to this document, a reference Python implementation can be found in keynames.py. The goal of the Python implementation is not to be fast, or to be used in any way other than as a reference. If there are any discrepancies between this document, the Python implementation and the actual C implementation in `src/libs/elektra/keyname.c`, you should consider them as follows:

1. The C implementation is optimized for speed and thus much harder to implement correctly.
2. In most cases, this document outranks the Python implementation. There may, however, be cases where the language in this document was too vague and the Python implementation is actually correct.
3. If two of the sources agree, the third one is probably incorrect. Although again, if one of the agreeing sources is the C implementation it could still be the case that there is a mistake.

In any case: If you find any discrepancies, please file a bug report at `https://issues.libelektra.org/new`.

> *Note:* Mistakes happen. The goal of the Python implementation was to provide a reference that has a very high likelihood of being correct.

*To Elektra developers:* Feel free to add any unclear or previous incorrect examples to the test cases in `tests/ctest/test_keyname.c`. These tests are very fast (1000+ test cases per second) and the more tests the better.

## 397.2 1. Key Name Parts and Namespaces

Before we dive into key names, we need to talk about key name parts and namespaces.
Each key is part of one of these *namespaces*:

- cascading
- meta
- spec
- proc
- dir
- user
- system
- default

Each of these namespaces has a very specific meaning, explained in Section 1.2.
Apart from the namespace, a key name is just a series of zero or more *key name parts*. Each key name part is just an arbitrary (possibly empty) sequence of non-zero bytes.
So without knowing anything about how key names are written, we could say that there is a key in the namespace "system" with the key name parts "elektra", "version" and "info".

> *Note:* Not every such sequence, is a valid key name. For more information see Section 4

## 397.2.1 1.1. Key Hierarchy

Elektra's keys commonly look like Unix paths:
`/elektra/version/info`

> *Note:* How this representation works exactly and how namespaces come into play, will be explained in the next section. For now, we only care that there is some similarity to Unix paths.

This is on purpose. Elektra's *key database (KDB)* is designed to resemble a Unix filesystem as much as possible. In particular, the KDB has a similar hierarchy. More generally, all key names exhibit this hierarchy. By going back to thinking about a key name as a namespace and a series of key name parts, we can define this *key hierarchy*.
Each namespace has a separate hierarchy. The relation between these, will be explored in the next section. For now, we just look at a single namespace.
In a Unix filesystems, we commonly talk about files and directories. We also say a file is located within a directory. But you might also know that in Unix "everything is a file". This applies to directories as well, but "a file is located within a file" is a bit clunky, so you might say "file `A` is located below file `B`", if `B` is a file within the directory `A`. What makes `A` a directory is just the fact that there can be other files below `A` [1].

### 397.2.1.1 1.1.1. The "is below" Relation

This relation of "is below" is also what defines Elektra's key hierarchy. Based on a key `K` with `n` key name parts, we say:

- A key `Km` is *below* `K`, if `Km` has `n+m` key name parts and the first `n` key name parts of `Km` are equal to the key name parts of `K` (in the same order).
- A key `K1` is *directly below* `K`, if `K1` is below `K` and `K1` has `n+1` key name parts.

Here are a few examples to show how this works in practice (using the Unix-path-like representation teased above):

| Key 1 | Key 2 | Relation |
|---|---|---|
| `/elektra/version/info` | `/elektra/version` | "Key 1" is directly below "Key 2" |
| `/elektra/version/info` | `/elektra` | "Key 1" is below "Key 2" |
| `/elektra` | `/elektra/version/info` | "Key 2" is below "Key 1" |
| `/elektra/version/info` | `/elektra/version/info` | "Key 1" and "Key 2" are equal |
| `/elektra/version/info` | `/elektra/data` | no relation |
| `/elektra/data` | `/elektra/version` | "Key 1" and "Key 2" are siblings |

You can think of the key hierarchy (within a single namespace) as a big tree of keys. Each node in the tree is a single key `K` and the children of the nodes are the keys that are directly below `K`.
The diagram above shows the key hierarchy of the keys in the table above (`A` -> `B` denotes `A` is directly below `B`).

> *Note:* While we could use the directory vs. file terminology for Elektra as well, it is recommended to avoid it. This is because in Elektra, every key may have an associated value. In particular a key may have a value, even if there are other keys below it. This value is **not**, as a beginner might suspect, the set of keys below it, like you could say the value of a directory is the set of files (and directories) within. The value is just another value, like any other key would have.
>
> Instead, we recommend the terms *leaf key* and *non-leaf key*, as these are commonly used for tree-like structures and their definitions fit perfectly.

#### 397.2.1.2 1.1.2. The "parent" Confusion

As an inverse to "is below" we sometimes use "parent". For example:

- `/elektra/version` is directly below `/elektra`, so `/elektra` is the (direct) parent of `/elektra/version`
- `/elektra/version/info` is below `/elektra`, so `/elektra` is a parent of `/elektra/version`
- `/elektra/version/info` and `/elektra/version` are below `/elektra`, so `/elektra` is the common parent.

This terminology can be easy to confuse, as "parent" is used for multiple different things. If the context doesn't make it clear what "parent" means, you might consider more clear terms, e.g. other common terms for tree-like structures. For example, you could use "ancestor" and "direct parent" as a clear differentiation.
There is also the term "(the) parent key". In many cases, this is refers to a very specific key in the given context that is a "common parent" to a certain set of keys. If you are working within such a context, be careful about "a parent key" vs. "the parent key".

### 397.2.2 1.2. Namespaces and Root Keys

We mentioned above that there are different namespaces in Elektra. Now we will explain their meaning.
To recap, Elektra knows these namespaces:

- cascading
- meta
- spec
- proc
- dir
- user
- system
- default

We mentioned above that there are key names with zero key name parts, i.e. just namespace. These are called *root keys* (based on Unix's filesystem root, as well as the root of a tree).
Let us explore them one by one:

- The simplest namespace is the ∗∗"meta"∗∗. The namespace "meta" is used exclusively for meta keys, i.e. keys that are attached to another key as metadata. The key hierarchy of the namespace "meta" is entirely separate from the key hierarchies in the other namespaces. Without external information, you cannot determine from a key name with namespace "meta", which key this metakey is attached to.
- keys with namespace ∗∗"proc"∗∗ only exist in memory and are scoped to the current **proc**ess.
- keys with namespace ∗∗"dir"∗∗ are scoped to a single filesystem **dir**ectory. They will (normally) be stored somewhere within this directory. Currently, there is no way of knowing which filesystem directory a key with namespace "dir" is scoped to. Elektra only uses keys scoped to the current working directory.
- Similarly, keys with namespace ∗∗"user"∗∗ are scoped to a single **user**. They will (normally) be stored somewhere within the user's home directory. Again, there is no way of knowing which user a key is scoped to and Elektra only uses keys scoped to the current user, i.e. the one that executed the application.
- The namespace ∗∗"system"∗∗ is what makes Elektra global. keys with namespace "system" are the same for all users of the system, independent of context. They are stored in system level directory.
- keys with namespace ∗∗"default"∗∗ are special. While you could create them manually, you normally don't want to. It is used for keys with default values for namespace resolution (explained below). keys with this namespace are also in-memory only.

- Then there is **"spec"**. This namespace, like "meta", is separate from the rest. We use it for keys that are part of a specification used to describe other keys. The metadata of every key with namespace "spec" describes a specification for the keys that have the same key name part but a different namespace (except "meta"). So the namespace "spec" has a closer relation to the others than "meta".
- Now just the namespace **"cascading"** remains. This namespace is used for namespace resolution (see below). It is the one that applications and end-users will use most commonly when interacting with Elektra. keys with namespace "cascading" are never stored. Not on disk and normally also not in a `KeySet`.

There is also a certain ranking between the namespaces "proc", "dir", "user", "system" and "default". Namely, that they override each other in exactly this order. Given two key names with identical key name parts, but one with namespace "dir" and one with namespace "user", the one with namespace "dir" should be considered more specific and should be preferred.
A special feature of Elektra is *namespace resolution*. Namespace resolution is the process of finding an appropriate namespace for a key based on a key name with namespace "cascading". It is most commonly used, when you need to find which key in the KDB should be used, based on a series of key name parts.
To resolve the namespace, we just look at each of the namespaces in the ranking defined above. We then use the first namespace where the key actually exists. Namespace resolution is performed, when `ksLookup`/`ksLookupByName` is called with a key name with namespace "cascading" [2]. This is also done, if you call `kdb get` or `kdb set` with a key name with namespace "cascading".

## 397.3 2. Escaped Names

The standard way to represent a key name in Elektra is this:
```
system:/elektra/version/info
```
This can be deconstructed into:

- The namespace: `system`
- The namespace separator: `:`
- A part separator: `/`
- A key name part: `elektra`
- A part separator: `/`
- A key name part: `version`
- A part separator: `/`
- A key name part: `info`
- The invisible null terminator

> *Note:* It might seem strange, that there is a part **separator** before the first part. This makes sense, because then the part separator always introduces a new part. A better fitting description would be "part introducer". But since we commonly call `/` a separator, we will stick to this terminology.

For keys in the namespace "cascading", we omit both the namespace itself and as well as the namespace separator:
```
/elektra/version/info
```
We also have a special rule for the root keys, i.e. the key names with zero key name parts and just a namespace. According to above rules `system:` would be the escaped name of the root key for the namespace "system" and the escaped name for the root key of the namespace "cascading" would be an empty string. But this is not the case. We use `system:/` and `/` instead. The exact details will be explored later, but for now just remember, that an (escaped) key name **always** contains at least one part separator (`/`).
But there is a problem. We said a key name part is "an arbitrary sequence of non-zero bytes". This means "elektra/version" is a key name part as well. Since the slash `/` would conflict with the part separator, we can escape it with a backslash `\` (and `\` can be escaped with another `\`):
```
/elektra\/version/info/back\\slash
```
This can be deconstructed into:

- A part separator: `/`
- A key name part: `elektra/version`
- A part separator: `/`
- A key name part: `info`
- A key name part: `back\slash`
- An invisible null terminator

*Note:* When talking about a single key name part `/` and `\` are never escaped.

Because of this escaping mechanism, we call this the *escaped name* of a key.
Elektra's key names are designed to mimic Unix paths to some extent. To this end we support the commonly used `/.` and `/...` This is one reason, why we need to differentiate between *canonical* key names and *non-canonical* key names.

### 397.3.1 2.1. (Non-)Canonical (Escaped) Key Names

Following the syntax of Unix paths, in Elektra both `/elektra/./version` and `/elektra/version` refer to the same key. Similarly, `/elektra/../version` and `/version` refer to the same key.
To give each key a unique key name, we need to introduce a *canonical (escaped) key name*. For Unix paths, we could say that the canonical path is the shortest possible path that refers to a file. In Elektra this doesn't quite work, but will use this definition for now.

> *Note:* Only escaped key names can be canonical or non-canonical, so we normally omit the "escaped" specifier.

Let us look at a few examples to get a feeling for canonical and non-canonical key names.

| Non-canonical | Canonical |
|---|---|
| `/elektra/./version` | `/elektra/version` |
| `/elektra/../version` | `/version` |
| `/elektra/../../version` | `/version` |
| `/elektra///version` | `/elektra/version` |
| `/elektra//../version` | `/version` |
| `/elektra/./../version` | `/version` |
| `/elektra/../../` | `/` |
| `user:/elektra/../../` | `user:/` |
| `/elektra/version/` | `/elektra/version` |

As you can see, the behavior of `.` and `..` matches that of Unix paths as long as we are using the namesapce "cascading". The namespace of a key name can never be changed via `..`, so `user:/..` is equivalent to `user:/`.
There is also a small difference in the last example. In Unix such paths would also refer to the same file, but in some Unix tools a trailing slash alters the behavior of the tool. In Elektra this is never the case. `/elektra/version/` and `/elektra/version` refer to the same key and are always treated as the same key name.
The only exception are the root keys. The canonical key names for the root keys always end with a `/`. In fact, we will see `later`, that removing the `/` makes the key name invalid.
There also is a completely new addition in Elektra. Elektra has a notion of *array parts*. These are key name parts that denote an array index. How exactly these work, will be explored `later`. For now, we only need to know that they start with an `#` and their canonical form has `n` underscores followed by `n+1` digits.
A few examples for array parts:

| Non-canonical | Canonical |
|---|---|
| `/elektra/#10` | `/elektra/#_10` |
| `/elektra/#1234` | `/elektra/#___1234` |

### 397.3.2 2.2. Other Escape Sequences

We already know, that `/` and `\` have to be escaped in an escaped key name. In addition to these two, there are a few more characters that have to be escaped. However, these additional characters may **only** be escaped under certain conditions.
The characters in question are: `.`, `#`, `%`. The conditions under which these characters can be escaped can be found below.

## 397.4 3. Unescaped Names

While the escaped name of a key is nice for humans, it is not very well suited for machines. The escaping of path separators makes it hard to find the key name parts of a given key name. The escaped name is also not well suited for sorting [3].
Both of these flaws are solved by the *unescaped name*. In unescaped names we use a zero-byte as the part separator. Since a key name part cannot contain a zero-byte, we do not need an escaping mechanism for the path separator.
However, these zero-bytes mean that the unescaped name is not a printable string and therefore not human-readable. This why we will only describe the unescaped name in the deconstructed form.
The escaped names from above correspond to the following unescaped names:

1. `system:/elektra/version/info`
    - The byte representing the namespace "system": `0x07`
    - A part separator: `0x00`
    - A key name part: `elektra`
    - A part separator: `0x00`
    - A key name part: `version`
    - A part separator: `0x00`
    - A key name part: `info`
    - The terminator byte: `0x00`
2. `/elektra/version/info`
    - The byte representing the namespace "cascading": `0x01`
    - A part separator: `0x00`
    - A key name part: `elektra`
    - A part separator: `0x00`
    - A key name part: `version`
    - A part separator: `0x00`
    - A key name part: `info`
    - The terminator byte: `0x00`
3. `/elektra\/version\\/info`
    - The byte representing the namespace "cascading": `0x01`
    - A part separator: `0x00`
    - A key name part: `elektra/version\`
    - A part separator: `0x00`
    - A key name part: `info`
    - The terminator byte: `0x00`
4. `/`:
    - The byte representing the namespace "cascading": `0x01`
    - A part separator: `0x00`

- The terminator byte: `0x00`

*Note:* We use 0xZZ to represent a single byte in hexadecimal. This form is only used when the context makes it clear that it represents a single byte and not a four character string.

The process of turning an escaped name into the corresponding unescaped name is called *unescaping*. Turning an unescaped name back into an escaped name is called *escaping*.
Unescaping works, by simply removing the backslashes `\` that are used as escapes. This applies both to `\/` and `\\` anywhere in key name parts, as well as to the escape sequences that are only used at the start of key name parts, e.g. `\#`.

## 397.5 4. Valid and Invalid Key Names

Not all of the key names described by the above sections are valid under all circumstances. You might also say, that a key name as defined above does not necessarily refer to a key in the KDB. So while it might be a key name, there is no key that actually uses this name. Only *valid key names* refer to a key in the KDB and may be used by a key as its name. The remaining key names are referred to as *invalid key names*.
For unescaped key names, it is pretty simple: Unescaped key names are valid key names, if all of the following are true:

- The first byte is a valid namespace byte, i.e. `0x01` - `0x08`.
- The second byte is a zero-byte `0x00`.
- The last byte is a zero-byte `0x00`.

For this reason unescaped names for root keys are 3 bytes long, with the last two bytes being zero-byte. Between the second and the last byte, we find the key name parts separated by a zero-byte `0x00`.
For escaped key names it is easier to define, what makes an invalid key name, than what makes a valid key name, so we will go this route. An escaped key name is considered an invalid key name, if any of the following are true:

- It is an empty string.
- The last character before the invisible null terminator is a backslash `\`. This is a dangling escape as we expect another character after the escape character `\`.
- It contains a namespace (i.e. the namespace is not "cascading"), but the namespace separator `:` is not followed by a `/`. (This mainly applies to root keys.)
- It contains a namespace separator `:`, but the substring before the first `:` is not one of: `meta`, `spec`, `proc`, `dir`, `user`, `system` and `default`.
- It contains an illegal escape sequence (see below).
- It is the string `/%` or consists of a namespace followed by the namespace separator `:` followed by `/%`. In other words, the first escaped part is translated into an empty unescaped part. The unescaped names for these keys would collide with the root keys `/`, `user:/`, etc.

  *Note:* The C-API does not allow you to construct a `Key` with an invalid key name; for example `keyNew` (and `keyVNew`) will return `NULL`.

### 397.5.1 4.1. Illegal Escape Sequences

The escape sequences `\\` and `\/` are always valid. For the other escape sequences certain conditions must be fulfilled:

- `\.`: can be used at the start of a key name part, if the whole key name part is `\.` or `\...` In other words, `\` can be used to escape the behavior of `.` and `...`
- `\#`: can be used at the start of a key name part, if the key name part would be a non-canonical array part without the `\.` specifically, `\#` can be used, if the key name part matches the regular expression `\\#[1-9][0-9]+` and the digits form a number less than or equal to 9223372036854775807 (= `2^63 - 1`). Meaning, `\` can be used to avoid array part canonicalization.

- \%: can be used, if the whole key name part is \%. That is \% is the escaped version of % (the empty key name part).

It may seem weird that some escape sequences have such specific requirements. This is necessary to create a 1:1 mapping between (canonical) escaped names and unescaped names. Without the restrictions, e.g. both \abc and abc would be unescaped as abc. In addition, the conditions for \# have to be so specific, because \ must only be allowed, if it affects unescaping. For example, we cannot allow \#abc, because that would unescape into #abc, just like #abc.

### 397.5.2 4.2. Array Parts

As mentioned above, Elektra has a notion of array parts. More specifically, certain key name parts will be interpreted as array indices under certain circumstances (see also documentation for arrays).
We already mentioned above, that array parts have canonical and non-canonical forms.
A canonical array part is a hash-sign # followed by n underscores (_) followed by n+1 digits. Additionally, the digits must form a number greater than or equal to 0 and less than or equal to 9223372036854775807 (= 2^63 - 1). The number must not have any leading 0s (except for the number zero itself).
In non-canonical key names, the underscores (_) are omitted. That is, either the correct number of underscores is used or no underscores at all.
Any other key name part that starts with a # is never an array part.
**All** key name parts starting with # are valid. It does not matter, if the key name part is an array part or not. However, some parts of Elektra may expect array parts under certain circumstances. Providing other key name parts under such circumstances, may cause problems. If the context doesn't call for an array part, then array parts behave no different and are treated as plain string just like any other key name part.
Finally, some examples:

| Key name part | Behavior in array | Behavior elsewhere |
|---|---|---|
| #0 | Index of first element | Child named #0 |
| #_10 | Index of 11th element | Child named #_10 |
| #_99 | Index of 100th element | Child named #_99 |
| #__100 | Index of 101st element | Child named #__100 |
| #abc | Possible error [4] | Child named #abc |
| #_100 | Possible error [4] | Child named #_100 |
| #__10 | Possible error [4] | Child named #__10 |
| #10a | Possible error [4] | Child named #10a |

## 397.6 4.3. Reserved Key Names

Apart from invalid key names, which cannot be constructed via the C-API, there are also *reserved key names*. These can be used with the C-API (keyNew returns a valid Key *), but there might be situations, in which Keys with such key names are treated differently or rejected entirely.
Generally speaking, any part of Elektra may define that some key names have special meaning, are not allowed, etc. However, sometimes guaranteed compatibility with other parts of Elektra is required. A good example are storage plugins. A storage plugins should strive to be compatible with as much of Elektra as possible. But since the storage plugin doesn't know anything about other plugins or even the application using Elektra, it is hard to attribute special meaning to certain key names.
This is why there are two types of reserved key name:

1. Any key name that is below system:/elektra: These key names are reserved for Elektra's internals. Each of these keys has a very specific purpose that is defined globally for all of Elektra. Using such a key name automatically carries this meaning. Even outside the context in which Elektra uses these directly, you should never use system:/elektra keys for other purposes.

2. Any key name containing the key name part ®elektra: These key names are reserved, but their meaning depends on the context. Similar to the METADATA.ini file for metadata, some conventions for these key names are defined in reserved name document.

> *Note:* We use UTF-8 here, so `®elektra` is specifically the 9-byte sequence `C2 AE 65 6C 65 6B 74 72 61`.

`Key`s with such key names will *never* be used in the interface between storage plugins and the rest of Elektra. This allows storage plugins to use `®elektra` to encode things that otherwise wouldn't be possible (e.g. values of non-leaf keys).

[1]: This explanation of Unix paths and Unix filesystems, is not entirely accurate. But it is good enough for our purposes, so we will just ignore some details. ↑

[2]: The actual process of resolution process that happens in `ksLookupByName` and `ksLookup` is a bit more complicated. It may involve some keys with namespace "spec" as well. ↑

[3]: For performance reasons, we want to make the comparison between to key names as fast as possible. A good solution is a single `memcmp`. But this doesn't account for the fact that key names represent a hierarchy and that `/` has a special meaning: `/key/sub` should always be sorted after `/key` and before `/key.1`. With `memcmp`, `/key` is first, because it is the shortest and otherwise equal. But then we would get `/key.1` not `/key/sub`, because `/` < `.` in ASCII. This cannot happen with zero-bytes as the separator, because there is no byte with a smaller value. ↑

[4]: We classify this as a "possible error", because not all parts of Elektra will fully validate all conventions and rules around arrays. In particular, a standard `KeySet` itself imposes no restrictions. Therefore, you can use any `KeySet` locally in your application. As soon as you pass the `KeySet` into an external function (e.g. `kdbSet`), however, errors may occur since the `KeySet` may pass through a function that requires `KeySet` which are properly validated against array rules. For more details what conventions and rules exist around arrays, see documentation for arrays. ↑

# Chapter 398

# Man Pages

Warning! These pages are generated from the files in /doc/help.

# Chapter 399

# Markdown Link Converter

The Markdown link Converter, which filters Markdown pages before the processing of doxygen, converts the links in Markdown pages. It is set up as input filter in doxygen, if a Markdown file is desired to be in the API documentation it only must have the extension `.md` and be in the `INPUT` path.
The Markdown link Converter gives each Markdown file a header `{ #header }` which is attached to a title and converts the links to refer to this headers. This conversion happens in 2 passes, which is needed because there can be files with no title.

## 399.1 Usage for Manual Invocation

```
markdownlinkconverter [<cmake-cache-file>] <input-file>
```

**The <input-file> parameter must be an absolute path!**

## 399.2 Conventions

- Links starting with `@ref`, `#` for anchors and `http`, `https` or `ftp` for extern links won't be touched.
- All other links to Markdown or arbitrary source files will be converted.
- All links to folders will be altered to the README.md in the Folder. This feature was introduced to be compatible with GitHub, where you can show the content of a folder in combination with the README.md of the containing folder.
- Anchors won't work in imported Markdown pages.

## 399.3 GitHub Specialities

- GitHub supports source code fences with syntax highlighting which are not recognized by Doxygen. Thus `sh` after the fence is removed for Doxygen.

## 399.4 Link Validation

### 399.4.1 Internal Links

The link validation works with a simple try to `fopen` the file, which the link refers to.

### 399.4.2 External Links

Every link starting with `http`, `https` or `ftp` will be written to a file named `external-links.txt` located in your build folder. With the following syntax:

```
<file>:<line>:0 <url>
```

Note: Due to the nature of the Markdown Link Converter the file can only be opened in append mode. So delete it and rerun the html build process (`make clean` could be needed) to get a list without duplicates.

In the script folder is a script named `link-checker`. This script can be used to validate the links. Broken links will be printed. False positive not excluded (very rare).
This link-checker reads the links from an input file (the synopsis is `link-checker <file>`) and prints the broken ones to the `stderr`. `wget` is used for the check so this program is required to be installed on your system. The link-checker includes the ability to list links on a whitelist which will not be checked and so on not printed to `stderr` in any case. Furthermore, this is the only way to let `http` links pass as they should be prevented by default. However, this does not mean that the `http` links are reachable as due to the lack of encryption, availability tests are not sensible in this case.

## 399.5 Further Improvements (Which Will be Introduced in a Later Version):

- optimize pdf output (also UTF-8 encoding)
- if title contains –, this should be- also remove other fences doxygen does not understand

# Chapter 400

# 0.8.6 Release

- author: Markus Raab
- pubDate: Sat, 21 Jun 2014 12:00:00 +0100
- shortDesc: adds technical previews & other improvements

## 400.1 Introduction

The Elektra Initiative moved to GitHub for following reasons:

- We were seriously missing a bug tracker
- It may make collaboration easier
- Build Server Integration:
  - Build on MergeRequests waits for the first merge request to be built!
  - And the build status is now published for this job

The main entrance point and URL for any advertisement and linking stays
`https://www.libelektra.org`
(which points to GitHub at the moment).
Additionally (not related to GitHub), we now have a `LCOV code coverage report`!

## 400.2 Improvements

The hosts plugin got documentation and several bug fixes. Multiline comments now remove the comment start sequences within the metadata comment. Additionally the kdb tool has an improved error message on invalid filenames. (thanks to Manuel Mausz)
Fix an issue that the resolver plugin did not delete tmpfile in some error situations and add test cases with error plugin.
Fix KS_END in C++ that did not specify the namespace.
"kdb run_all" will now run all tests using Elektra from your system. Because not all testdata will be installed, some test cases fail unless you copy the data manually. Those are:

```
testmod_yajl testmod_augeas testmod_fstab testmod_hosts test_xml
```

Virtually all clang compiler warnings were fixed (thanks to Felix Berlakovich).
kdb mv now is atomic. (API was used wrongly in the tool)
Clang, icc and gcc are now all supported and tested.

## 400.3 Technical Previews

A plugin that logs write operations and errors via the native journald interface was added (thanks to Felix Berlakovich).

A preview of the augeas plugin was added. In CMake add "augeas" to PLUGINS and then run "kdb info augeas" for further instructions. Additionally, see Outlook below (thanks to Felix Berlakovich).
A preview of python3 and lua bindings were added. In CMake enable the options `BUILD_SWIG BUILD_SWIG_LUA BUILD_SWIG_PYTHON` so that they are compiled (thanks to Manuel Mausz).
The code generator was extended to support contextual values, more information about that topic later.
The libtools library is back to life again! It has the same idea as its predecessor, but it is much more powerful. Because of the plugin system libtools is able to export and import any configuration format Elektra supports. Additionally it adds support for mounting backends. While most of its functionality is mature, the API is not final, though.

## 400.4 API Changes

The API and ABI is as always backwards-compatible within the 0.8 series.
This time it is, however, not forward-compatible (that means programs linked against 0.8.6 might not work with 0.8.5), because ksAtCursor() was added.
ksAtCursor() provides the means for an external iterator. This was immediately exploited by an implementation of the C++ iterators. These new C++(11) (reverse)iterators allow multi-pass algorithm over the same KeySet.
ksSort was in the kdb.h header file, even though it never was needed nor available in 0.8. Automatic compatibility checkers may wrongly tell that the API is not compatible, but this is not the case (thanks to Manuel Mausz).
The C++ API changed, including:

- does not wrongly convert garbage to default types using `get<T>`
- getMeta now internally uses `get<T>` and both throw the KeyTypeConversion Exception
- KeySet::at directly allows one to use ksAtCursor()
- KeyTypeConversion (or former KeyBadMeta) is not thrown anymore if key is not available
- Key::hasMeta allows one to directly check if metadata is available

CMake: ENABLE_CXX11 is now default OFF (because gcc 4.6 and older won't work with it). It disables some tests, though. `unique_ptr` will automatically be used instead of `auto_ptr` in this mode.
Note for maintainers: libelektratools now needs to be installed so that the kdb tool will work.

## 400.5 Outlook

One drawback of Elektra is the small number of available storage plugins. While writing storage plugins isn't too hard, it still requires knowledge of Elektra's plugin architecture. In addition it is always necessary to keep both translation directions in mind (from file to Elektra and back). In order to mitigate this issue Felix Berlakovich is working on an Elektra storage plugin which uses Augeas [1] to read and manipulate configuration files. Augeas uses a concept called lenses to translate between files and abstract trees (i.e. trees that hide unimportant details like whitespaces). In brief, what makes lenses special is the fact that they automatically disallow translations that cannot be reversed. This means that one of the translation directions comes for free as it is implicitly described by the other one. Therefore writing lenses is much simpler than programming both directions explicitly as done in traditional Elektra storage plugins. Furthermore many lenses for common applications like Apache, MySQL, DHCPd and many more [2] already exist. Besides these benefits, Augeas lacks many features of Elektra. Although this is mostly intended because Augeas aims to concentrate just on configuration file manipulation some of these features would be valuable for administrators and developers. Some of these mentioned Elektra features include but are not limited to: logging of all configuration changes done, transformation of character encoding as needed, import and export of configuration data in any format, e.g. XML or JSON (Augeas has a similar feature for XML), file conflict detection and several different validations for configuration options (see [3] for a full list of features). With the Augeas storage plugin these features cannot be used with only the applications already integrated into Elektra, but with all applications for which lenses exist. This storage plugin targets system administrators as well as developers, that wish to take advantage of Elektra's features, but were unable to do so because of missing plugins. If no lens exists already, writing a new one requires neither programming skills nor knowledge of Elektra's plugin architecture. Furthermore, implementing a new configuration file format is just a matter of writing a lens instead of writing a full-fledged parser. [1] http://augeas.net/ [2] http://augeas.net/stock_lenses.html [3] https://www.libelektra.org

Since Elektra is written in C, integration is limited to projects written in C and languages compatible with C like C++. In order to change this situation Manuel Mausz is working on the implementation of language bindings using two different techniques. The first technique uses SWIG as a generic generator for various languages. In a nutshell, SWIG is a compiler that takes C/C++ declarations and creates the wrappers needed to access those declarations from other languages. The second technique is called GObject Introspection. In contrast to SWIG, GObject Introspection generates a language independent metadata file. For dynamic languages the GObject Introspection support in the target language will load this metadata file to generate bindings at run-time. In order to provide bindings for static languages a compiler on the metadate file will be used. This is comparable to using SWIG. Manuel will focus on implementing language bindings for Python and Lua though both techniques will support various other languages. As language bindings only allow to call into Elektra's API Manuel will also embed the interpreters of both Python and Lua into two generic plugins. Using these plugins in combination with the languages bindings it will be fully possible to write plugins for Elektra in languages other than C/C++.
While Elektra offers sophisticated methods to store and manipulate configuration, currently there is only a command line tool (kdb) available to interact with the database. This requires the user to study and learn the possible commands and does not allow free exploration of the options of the application. To enable new ways to interact with Elektra, Raffael Pancheri is designing and implementing a Graphical User Interface (GUI). To increase the probability to create a GUI that is indeed usable and beneficial, Pancheri is following the design principles proposed by Ben Shneiderman and Jakob Nielson.
Finally, as you already noticed, Ian Donnelly is working on a semantic 3-way merge to make distribution and package upgrades smoother. This is done as GSOC project. The progress is documented here: http://community.libelektra.org/wp/

## 400.6 Get It!

You can download the release at: http://www.markus-raab.org/ftp/elektra/releases/elektra-0.8.6.tar.gz

- size: 1188337
- md5sum: 4a59824e70a29295e9ef9ae7605d9299
- sha1: 2570710b0057470223611ca00d61a0196e54e7b2
- sha256: e815cf69b070c339784472841aa0ee0b169fab7c78f41cbbd7044f53fa9ed216

Documentation can be found here: https://doc.libelektra.org/api/0.8.6/html/
You can install the debian packages for debian (wheezy, amd64 only, some packages will be added later) by adding following lines to your /etc/apt/sources.list: deb http://markus-raab.org/ftp/wheezy wheezy main and install or upgrade the packages with:

```
sudo apt-get update
sudo apt-get install libelektra-core4 libelektra-full4 libelektra-bin libelektra-dev libelektra-test
      libelektra-xmltool4 libelektra-json4 libelektra-dbus4 libelektra-doc
```

Best regards, Markus

# Chapter 401

# 0.8.7 release

- author: Markus Raab
- pubDate: Mon, 28 Jul 2014 12:00:00 +0100
- shortDesc: adds python2 bindings, 3-way merge & improvements

Again, we managed to have a great feature release with dozens of corrections!

## 401.1 New Features

Thanks to Manuel Mausz for further improving lua, python3 bindings and the new python2 bindings.
The GSoC efforts have their first large contribution to Elektra, the 3 way merge finally arrives with this release. It is still a long way to go, however, because augeas plugins can only be mounted with a workaround and the package integration of the 3-way merge is still in its infancy.
A special thanks to Felix Berlakovich for his contributions to the 3-way merge, including meta merging, conflict resolving strategies and extensive testing.
Additionally, he added the plugins keytometa, ini and greatly improved the glob plugin. These plugins are technical previews and will receive some improvements in the next release, too.
Now a script for tab completion is available here, again thanks to Felix Berlakovich.
The contextual values now got a `tutorial` and small fixes.

## 401.2 Corrections

Thanks to Pino Toscano for fixing a lot of spelling errors, simplify RPATH setting, respect `$HOME` and `$TMPDIR`, improvements of test cases, and his debian-packaging efforts.
In the kdb tool not allowed subfolders are now checked properly and the output of warnings comes before output of the error. This fixes the problem that in the case of a longer list of warnings one did not see the error anymore.
Fix compilation warnings on clang and gcc 4.9. Also improve test coverage on kdb tool and some plugins.
Fix kdb import/export for some plugins (Should now work with any storage plugin again).
kdb run_all should run flawlessly with this release. Remaining problems with not installed test data were fixed. kdb run_all also checks if the test cases do not modify any existing key and keeps a backup if this happened.
Some remaining mem leaks in rare circumstances were fixed. Valgrind should now never report any leaks, if it does, please report the issue.

## 401.3 API Changes

Added delMeta() for C++, because setMeta() with NULL will set the number 0 and not remove the meta.
Arguments of isBelow, isDirectBelow, isBelowOrSame are swapped for better readability. k.isBelow(root) now means the obvious thing. The change only effects the C++ binding, keyIsBelow is unaffected by the change.

## 401.4 Documentation

Specification of metadata and contracts written/greatly improved.
Decisions are introduced again.
Most often the KeySet is ideal, e.g. when doing full iteration or when performing set operations. In some cases, however, a hierarchical data structure fits better. This is especially true for GUIs. Luckily, Keys can be in multiple data structures because of their reference counting.

## 401.5 Other Stuff

We now fully embrace github:

- We use its issue tracker (all issues from local text files were moved there)
- We have rewritten many READMEs to use GitHubs Markdown
- On pull requests the build server checks if the merge would break the build.
- All previous gitorious users are now at github. (Most had an account anyway)

Raffael Pancheri also made progress with its qt-gui. It now features a model that implements great parts of Elektra's features. Unfortunately the model cannot be serialized and thus changes cannot made persistent. Also undo and other important use-cases are still not there. The GUI looks very clean and was evaluated in a SUS study on 23.07.2014. The current implementation can be found here.
Many distributions already have Elektra packages

- Gentoo
- Arch Linux

In some distributions Elektra packages are available, but are not up-to-date. Pino Toscano is working on get them (actually Debian, but others are derived from it) up-to-date.

- Ubuntu
- Debian
- Linux Mint

A special thanks to Kai-Uwe Behrmann for providing packages for CentOS, Fedora, OpenSUSE, RHEL and SLE.

## 401.6 Get It!

You can download the release from:
http://www.markus-raab.org/ftp/elektra/releases/elektra-0.8.7.tar.gz size: 1566800 md5sum: 4996df62942791373b192c793d912b4c sha1: 00887cc8edb3dea1bc110f69ea64f6b700c29402 sha256: 698ebd41d540eb0c6427c17c13a6a0f03eef94655fbd40655c9b42d612ea1c9b
already build API documentation can be found here:
https://doc.libelektra.org/api/0.8.7/html/
Best regards, Markus

# Chapter 402

# 0.8.8 Release

- guid: eca69e19-5ddb-438c-ac06-57c20b1a9160
- author: Markus Raab
- pubDate: Tue, 02 Sep 2014 17:31:42 +0200
- shortDesc: adds 3-way merging, reworked documentation & improved kdb tool

In this release we changed 578 files in 473 commits (68596 insertions(+), 59260 deletions(-) compared to Elektra 0.8.7). We assume thats the largest changeset for any of Elektra's releases up to now. It happened only within a bit more than a month up (0.8.7 was released 28.07.2014).

## 402.1 New features

GSoC finished successfully (thanks Ian and Felix) See http://community.libelektra.org/wp for the latest results. So Elektra now has a 3-way merging framework that is superior to text-based merging in many scenarios (e.g. moving configuration options within a file or with in-line comments) iff a storage plugin creates key names that are not only line numbers. We love to get Feedback!
Writing plugins is now even more comfortable. A plugin writer tutorial was written (thanks Ian): https://github.com/ElektraInitiative/libelektra/blob/master/doc/tutorials/plugins.md The documentation was completely reworked: https://doc.libelektra.org/api/0.8.7/html/group__plugin.html And two new macros allow printf formatting for warnings and errors (`ELEKTRA_ADD_WARNINGF` and `ELEKTRA_SET_ERRORF`).
The ini plugin was greatly improved (tested with samba configurations and added to ALL plugins) and the hosts plugin was rewritten to support ipv6 properly (thanks to Felix).
The constants plugin was added and allows introspection of Elektra's CMake variables. Because such non-file-based plugins (e.g. also uname) do not need resolving, the plugin noresolver was added. It supersedes the success plugin.
Elektra now allows one to correctly fsync its configuration files (sync plugin) and the folders where files are stored (resolver plugin). Just make sure to add the "sync" plugin using kdb mount. The resolver plugin now reads from passwd and no longer needs environment variables. Additionally, the resolver plugin was prepared to support other variants by so-called compilation variants.
The error plugin now allows, next to list all possible errors, to provoke errors when opening plugins. We fixed some issues related to plugins having errors when they initialize themselves.
So following plugins were added: sync noresolver line ini constants Nearly all plugins now have a README.md for further information (thanks to Ian). An overview of all plugin is on with links to them: https://github.com/ElektraInitiative/libelektra/blob/master/src/plugins/
The kdb tools were greatly improved (thanks to Felix):

- added remount tool
- umount now also accepts mountpath
- mount allows one to specify different resolvers
- import now can use merge strategies

- check without arguments checks key database
- mount is now more verbose when validation fails

New/improved scripts/make targets (note that scripts can be executed by kdb scriptname):

- mounting, unmounting scripts were added
- generate template for a new plugin was improved
- configure-debian was added
- added targets `run_all` and `run_memcheck`
- bash completion file now installed
- ucf integration
- merging scripts were added for the usage with ucf
- scripts doing internal checks on source of plugins

## 402.2 Compatibility

This time we had to break compatibility. We did not change the ABI (your application still will be able to use Elektra 0.8.8) and we did not change the API (your application still will compile against Elektra). We changed the third part of our interface: the semantic interface.
The problems were following: keyAddBaseName/keySetBaseName did something obvious when no special characters were in the baseName. But once there were, there are two different interpretations what it should do:

1. add/set a basename, so escape characters that are not canonical in the basename
2. add all parts of the name given (with slashes)

The methods were used in both ways, so it was obvious that something is very wrong. We decided that it should do what the name says, that is add/set a basename (variant 1).
The variant 2, to add any name was added and is called keyAddName() and added as proposal.
(Thank Felix for implementations and Manuel for investigations)
When keys are renamed after adding to a keyset is a bad thing because it destroys the order of the keyset. This is now avoided by keyLock. Use keyDup() to get rid of such locks.
Another, even larger, change is also about ordering of keys in keysets. Elektra now internally has an null-terminated unescaped key name. Ordering of keysets will always happen on this name. The `keyCmp()` tool can be used to check this order. It works very efficiently with `memcmp()` and never gets confused by ASCII ordering of / (because / is 0 in the unescaped key name).
The syntax, semantics and conventions of key names is now documented in detail: https://doc.libelektra.org/api/0.8.8/html/group__keyname.html
ksNew() does now return a keyset with a properly set cursor (ksRewind).
Because its always possible that software relies on bugs the better way to deal with such a situation (as the keySetBaseName() situation described above) is to provide the same function twice. Manuel said he will create a prototype to introduce symbol versioning in Elektra. With that, old customers would still receive the old behavior, but people compiling against a new version would get the new behavior. So in one of the next releases we will also avoid semantic interface changes when there is a valid use case for it (there is none if the program e.g. crashes).
Symbol versioning also allows one to compile against old versions on purpose if you do not want the new behavior.
We have prepared an ABI-test suite, that also checks behavior, for that purpose, but we also improved testing in other parts:

- (New Test strategy)[/doc/TESTING.md]
- New resolver tests for conflicts (needs tty)

If you try to execute test_ks from 0.8.7 with libelektra 0.8.8 it will crash, but not because of any incompatibility, but because of `strcmp` in the test itself gets a null pointer. The pointer is now null, because ksNew correctly rewinds its internal cursor (see above). Amusingly, it says on that line 94 in test_ks.c: // TODO: why is the cursor here?

## 402.3 API Proposals

see above for more information:

- `keyAddName` .. add key name without escaping, like keySetName
- `keyUnescapedName` .. get access to null-separated unescaped name
- `keyLock` .. to allow one to secure keys against modifications

some new ideas:

- `keySetStringF` .. printf format-style changing of the key string
- `elektraKeySetName` .. to allow one to set meta + cascading keys

`elektraArrayIncName()` now works correctly with empty arrays embedded in other arrays (yajl+line plugin)
`elektraArrayValidateName()` was also added, thanks to Felix.
These methods are declared in the file `kdbproposal.h` but do not guarantee any forms of compatibility (they might even be removed).

## 402.4 Issues

Many issues were resolved as you can see in github: https://github.com/ElektraInitiative/libelektra/iss
Alone for the milestone 0.8.8 we closed 17 issues, including those mentioned in "Compatibility". Other issues (not all were tracked on GitHub):

- fix undefined errors in `kdbOpen()` or `kdbClose()`
- Now Python 2+3 work in parallel
- python2 interpreter is found correctly (also on Arch)
- Sentinel now makes sure that you cannot forget `KS_END` to end `ksNew`
- Fixes for architecture-specific problems by Pino
- fix .pc file
- fix compilation problem with `KDB_MAX_PATH_LENGTH`
- tmpnam to mkstemp (security)
- make test data naming consistent (thanks Pino)
- use `LIB_SUFFIX for TARGET_TOOL_EXEC_FOLDER` thanks to Kai Uwe
- Fix search for boost (thank Pino)

## 402.5 Other Stuff

Thanks to Pino Toscano Elektra 0.8.7-4 is now available in Debian Testing: https://packages.debian.org/search?keywords=elektra So it is only a matter of time that other (debian-based) distributions will follow and replace the dusty Elektra 0.7.
Debian Continuous Integration http://ci.debian.net/packages/e/elektra (thanks Pino) greatly complement our tests running on https://build.libelektra.org/
Elektra's buildserver also was improved:

- now also compiles with icc
- runs `make run_memcheck`
- checks if plugins are added correctly in-source

- runs ABI + behavioral tests

Raffael Pancheri now made a merge request for qt-gui https://github.com/ElektraInitiative/libelektra/pull
in which copy, paste and delete of keys already works. It is, however, still work in progress.
Manuel Mausz made great progress in script-based Elektra plugins. He is also working on glib+gobject-introspection based bindings. He investigated some issues, e.g. a crash of the python binding which was only triggered if python3 is build with a specific flag/module combination, see: https://github.com/Elektra↩
Initiative/libelektra/issues/25

## 402.6 Get It!

You can download the release from:
http://www.markus-raab.org/ftp/elektra/releases/elektra-0.8.8.tar.gz

- size: 1644441
- md5sum: fe11c6704b0032bdde2d0c8fa5e1c7e3
- sha1: 16e43c63cd6d62b9fce82cb0a33288c390e39d12
- sha256: ae75873966f4b5b5300ef5e5de5816542af50f35809f602847136a8cb21104e2

already built API documentation can be found here:
https://doc.libelektra.org/api/0.8.8/html/
Best regards, Markus

# Chapter 403

# Augeas and Config::Model

- author: Markus Raab
- guid: eca69e19-5ddb-438c-ac06-57c20b1a9160
- pubDate: Mon, 22 Oct 2014 17:31:42 +0200

A common question is: now we have Augeas for editing config files, why do we need Elektra, Config::Model or something else?
First, it is clear that Augeas is a huge step forward and improved configuration of Linux systems, especially when used with centralized configuration management tools.
Augeas, in short, introduces a special-purpose programming language that allows to transform configuration files into configuration trees and back. This transformation is it strength (so it is easy to add support for a particular legacy configuration files), but also it weakness (the mapping is implementation-defined by a language that is limited to a bit more than regular expressions). Augeas is not able nor is it intended to provide more abstraction over the configuration files. Instead Augeas mirrors the structure of the configuration as closely as possible.
Elektra's goal, instead, is not only to provide access to legacy configuration files, but to provide access to the configuration exactly as the programs itself uses it. So with Elektra, the developers of applications are part of Elektra's ecosystem by providing specifications how their configuration should look like and by writing plugins that define how the configuration is accessed and checked. Ideally, after some time of legacy issues and migration, developers will also not have to care about writing plugins anymore, but just use any available ones (and users of their application can choose any other compatible plugin). What is about to stay is a specification that defines the application's configuration, e.g. in INI (could be any syntax):

```
[/yourapp/file_dialog/show_hidden_files]
type=Boolean
default=true
```

allows other applications to reuse your setting show_hidden_files by referring to above specification. So Elektra not only abstracts from cross-platform-related issues with an consistent API, but also allows us to be aware of other applications' configurations, leveraging easy application integration.
Config::Model shares most of Elektra's goals, especially those regarding validation (you saw the type=Boolean above) and having a unified interface for all programs (this feature is unavoidable with any such approach). The projects mainly differs that Elektra is supposed to be used by the programs themselves (and not only by GUIs and validation tools) and that Elektra uses self-describing data: the specification itself is also in Elektra's key database, stored in metadata and e.g. below system/elektra/mountpoints. In Elektra validators can be written in any language (because the specification is just data) and can enforce constraints on any access (because plugins define the behavior of the key database).

# Chapter 404

# 0.8.9 Release

- guid: 38640673-3e07-4cff-9647-f6bdd89b1b08
- author: Markus Raab
- pubDate: Tue, 04 Nov 2014 10:48:14 +0100
- shortDesc: adds qt-gui, several optimizations & fixes

Again we managed to do an amazing feature release in just two month. In 416 commits we modified 393 files with 23462 insertions(+) and 9046 deletions(-).

## 404.1 Most awaited

The most awaited feature in this release is certainly the *qt-gui* developed by Raffael Pancheri. It includes a rich feature set including searching, unmounting, importing and exporting. A lot of functionality is quite stable now, even though its version is 0.0.1 alpha. If you find any bugs or want to give general feedback, feel free to use the issue tracker of the Elektra Initiative. A screenshot can be found `here` To compile it (together with Elektra), see the README `here`
Manuel Mausz also has been very active and developed glib+gi bindings. These bindings make Elektra more friendly to the glib/gtk/gnome world. Using the gobject introspection python3 and lua bindings were developed. Additionally he got rid of all clang warnings.
Felix Berlakovich also made progress: `the ini plugin` now supports multiline and which can be dynamically turned on and off, i.e. during mounting (thanks to Felix)
Last, but not least, Kai-Uwe ported Elektra to Windows7. MinGW is now one more supported compiler (tested on build-server, see later). Astonishingly, it was only little effort necessary: Basically we only needed a new implementation of the resolver, which is now called *wresolver*. Different from the *resolver* it lacks the sophisticated multi-process and multi-thread atomicity properties. On the plus side we now have a resolver that is very easy to study and understand and still works as file resolver (*noresolver* does not).

## 404.2 Interfaces

ABI/API of the C-API is still completely stable even though under the hood a lot was changed. All test cases compiled against the previous version still run against Elektra 0.8.9.
This is, however, not the case for libtools. For MinGW porting it was necessary to rename an enum related to merging because it conflicted with an already defined MACRO.
For maintainers also some changes are necessary. For MinGW and to actually use the flexibility of the new resolver variants two new CMake Variables are introduced: KDB_DEFAULT_RESOLVER and KDB_DEFAULT_STORAGE.
More importantly for maintainers the CMake variables regarding SWIG bindings are now abandoned in favor to the new variable BINDINGS that works like PLUGINS and TOOLS. Just start with

```
-DBINDINGS=ALL
```

and CMake should remove the bindings that have missing dependencies on your system. Remember that glib and gi (i.e. *gi_python3* and *gi_lua*) bindings were introduced, too. Additionally, the *cpp* binding can now be deactivated if not added to BINDINGS.
Finally, the *gen* tool added a Python package called `support`.

## 404.3 Other Bits

A proof of concept storage plugin `regexstore` was added. It allows one to capture individual configuration options within an otherwise not understood configuration file (e.g. for vimrc or emacs where the configuration file may contain programming constructs).
Most tests now also work with the BUILD_SHARED variant (from our knowledge all would work now, but some are still excluded if BUILD_FULL and BUILD_STATIC is disabled. Please report issues if you want to use uncommon CMake combinations).
A small but very important step towards specifying configuration files is the new proposed API method ksLookupBySpec (and ksLookup implementing cascading search). It introduces a `logical view` of configuration that in difference to the `physical view` of configuration does not have namespaces, but everything is below the root "/". Additionally, contextual values now allow one to be compile-time configured using C++-Policies. These are small puzzle pieces that will fit into a greater picture at a later time.
A (data) race detection tool was implemented. Using it a configurable number of processes and threads it tries to kdbSet() a different configuration at (nearly) the same time.
With this tool the resolver could be greatly be improved (again). It now uses stat with nanosecond precision that will be updated for every successful kdbSet(). Even if the configuration file was modified manually (not using Elektra) the next kdbSet() then is much more likely to fail. Additionally a recursive mutex now protects the file locking mechanism.
The build server now additionally has following build jobs:

- i386 build: We had an i386 regression, because none of the developers seems to use i386 anymore.
- Configure Debian Script: Calls the scripts/configure-debian(-wheezy).
- Local Installation: We had an regression that local installation was not possible because of a bash completion file installed to /etc. This build tests if it is possible to install Elektra in your home directory (and calls kdb run_all afterwards)
- Test bindings: Compiles and tests ALL bindings.
- Mingw: Compiles Elektra using mingw.

Many more examples were written and are used within doxygen. Most snippets now can also be found in compilable files:

- `keyNew examples`
- `keyCopy examples`
- `C++ deep dup`
- `How to put Key in different data structures`
- `Mount some config files using augeas`
- `Mount system information`

Most plugins now internally use the same CMake function `add_plugin` which makes plugin handling more consistent.
Felix converted the METADATA spec to ini files and added a proposal how comments can be improved.

### 404.3.1 Refactoring:

- reuse of utilities in gen code generator
- the gen support library is now in its own package (`support`)
- refactor array handling
- internal comparision functions (keyCompareByName)

### 404.3.2 Optimization:

- lookupByName does not need to allocate two keys
- lookups in generated code
- prefer to use allocation on stack

### 404.3.3 Fixes:

- disable cast that segfaults on i386 (only testing code was affected)
- fix keyAddBaseName in xmltool and testing code
- support non-system installation (e.g. in home directory)
- rewrote test cases to use succeed_if_same to avoid crashes on null pointers
- allow one to use python 2.6 for kdb gen
- improve exception messages
- use memcasecmp (fix lookup ignoring case)
- fix memory leaks (ini)
- text messages for some warnings/errors
- fix many issues regarding CMake, more variants of setting CMake options are now allowed.
- CMake policies fixes allow us to use CMake version > 3

## 404.4 Get It!

You can download the release from here

- size: 1936524
- md5sum: 001c4ec67229046509a0cb9eda223dc6
- sha1: 79ea9b83c08ed4c347ed0100b5e0e2d3309b9d04
- sha256: e0895bba28a27fb37f36f59ef77c95235f3a9c54fb71aa6f648566774d276568

already built API documentation can be found here
For more information, see https://www.libelektra.org
Best regards, Markus

# Chapter 405

# 0.8.10 Release

- guid: 6ce57ecf-420a-4a31-821e-1c5fe5532eb4
- author: Markus Raab
- pubDate: Tue, 02 Dec 2014 18:37:51 +0100
- shortDesc: adds XDG/OpenICC compatibility & java binding

Hello,
we are delighted to announce our latest feature release providing major updates in:

- compatibility with standards,
- tooling,
- plugins (hosts, rename),
- Qt-Gui and
- a new Java binding

## 405.1 XDG Compatibility

Elektra now is fully XDG 0.8 compliant. Following changes were necessary:

- newly created configuration files for user/ now have the permissions 0600
- newly created configuration directories for user/ now have the permissions 0700
- existing configuration files will retain their permissions.
- The default path to store user configuration is now ~/.config
- A new resolver variant x (for user and system) is introduced
  - implements handling of XDG environment variables
  - ignores empty dirs and absolute paths in envvar
- add new shell based test suite for (xdg)-resolver

For example, we could use `resolver_fm_xhp_x`:

```
kdb mount --resolver=resolver_fm_xhp_x file.dump /example dump
kdb file user/example
kdb file system/example
```

Will show you that for both user+system the resolver respects XDG environment variables, e.g. above lines will print:

```
/home/m/.config/file.dump
/etc/xdg/file.dump
```

Of course, any attempts to get and set keys below user/example and system/example will also be in these files.
The letters after _ describe the variant of the resolver:

- `f` .. file-based locking
- `m` .. mutex based locking (for multiple KDB per process)
- for user configuration (after next _)
  - `x` .. first check `XDG_CONFIG_HOME` environment
  - `h` .. then check `HOME` environment
  - `p` .. then fall back to passwd
- for system configuration (after next _)
  - `x` .. check all paths in `XDG_CONFIG_DIRS` and falls back to `/etc/xdg`

A lot of such resolver variants are added when `-DPLUGINS=ALL` is used. Of course you can create new variants with different behavior by adding them to PLUGINS.
To make your application (that uses Elektra) XDG aware, you have nothing to do: you get it to free. Make sure to always use cascading lookup. Additionally, an XDG conforming application should not write system/ keys.

## 405.2 OpenICC Compatibility

Based on that, Elektra now also implements the draft for the OpenICC specification.
The mount command looks like quite complicated, but it consists of simple parts:

```
kdb mount --resolver=resolver_fm_xhp_x \
  color/settings/openicc-devices.json /org/freedesktop/openicc \
  yajl rename cut=org/freedesktop/openicc
```

We already know the first two lines: we use the XDG resolver already introduced above. Only the file name and the path where it should be mounted differs.
The plugin yajl is a storage plugin that reads/writes json. The plugin rename was the missing link to support OpenICC (thanks to Felix Berlakovich for closing this gap). It is needed, because every OpenICC file starts like this:

```
{ "org": { "freedesktop": { "openicc": {
```

Because the backend is mounted at /org/freedesktop/openicc, it would lead to keys below /org/freedesktop/openicc/org/freedesktop/op which we obviously do not want. So we simply get rid of the common prefix by cutting it out using the rename plugin.
Of course this renaming functionality can be used in every situation and is not limited to OpenICC.

## 405.3 Tools

A large number of old and new tools were added, mostly for convenience e.g.:

```
kdb mount-openicc
```

saves you from writing the long mount command we had in the previous section.
To get a list of all tools that are installed, now the command (which is also an external tool and as such currently not displayed in kdb –help):

```
kdb list-tools
```

is available. Do not be surprised: on typical installations this will be a large list. You can run each of these tools by using kdb <command>. Most of the tools, however, are part of the test suite, which you can run using:

```
kdb run_all
```

Other tools are "old friends", e.g. convert-fstab written in 2006 by Avi Alkalay still works:

```
kdb convert-fstab | kdb import system/filesystems xmltool
```

It will parse your /etc/fstab and generate a XML. This XML then can be imported. Other convert tools directly produce kdb commands, though.
kdb now uses KDB itself for many commands:

- /sw/kdb/current/resolver .. You always want a different default resolver than that was compiled in as default when mounting backends?
- /sw/kdb/current/format .. If you are annoyed by the default format dump format for import/export.
- /sw/kdb/current/plugins .. If you always forget to add some plugins when mounting something.

By default the plugin "sync" is added automatically (it makes sure that fsync is executed on config files, the directory is already done by the resolver), you should not remove it from /sw/kdb/current/plugins otherwise the next mount command will not add it. To preserve it use a space separated list, e.g.:

```
kdb set user/sw/kdb/current/plugins "sync syslog"
```

Last, but not least, kdb get now supports cascading get:

```
kdb get /sw/kdb/current/plugins
```

This feature allows you to see the configuration exactly as seen by the application.
Other options:

- -123 options for hiding nth column in `kdb mount`
- hide warnings during script usage of `kdb mount`
- -0 option accepted in some tools (null termination)
- Mount got a new -c option for backend configuration. For example -c cut=org/freedesktop/openicc would be the parameter cut for all plugins. Have a look at #146 if you want to use it.

## 405.4 Compatibility

The core API (kdb.h), as always, stayed API/ABI compatible. The only changes in kdb.h is the addition of `KEY_CASCADING_NAME` and `KEY_META_NAME`. So applications compiled against 0.8.10 and using these constants, will not work with Elektra 0.8.9.
The constants allow us to create following kinds of keys:

- empty names: this was always possible, because invalid names (including empty names) did not cause keyNew to abort
- metanames: this is a new feature that allows us to compare key names with metakeys
- cascading names: names starting with / have the special meaning that they do not specify which namespace they have. When such names are used for
  - kdbGet() and kdbSet() keys are retrieved from all namespaces
  - ksLookup() keys are searched in all namespaces
  - ksLookupByName() is now just a wrapper for ksLookup(). The method does not do much except creating a key and passing them to ksLookup().

Usage in C is:

```
Key *c = keyNew("/org/freedesktop", KEY_CASCADING_NAME, KEY_END);
Key *m = keyNew("comment/#0", KEY_META_NAME, KEY_END);
```

The same functionality exists, of course, in available in all bindings, too.
Changes in non-core API are:

- xmltool now does not output default (unchanged) uid,gid and mode
- ksLookupBySpec from kdbproposal.h was removed, is now integrated into ksLookup
- extension keyNameGetNamespace was removed
- the hosts comment format has changed
- the default resolver has changed (uses passwd)
- kdb::tools::Backend::Backend constructor, tryPlugin and addPlugin have changed:
- mountname is now automatically calculated
- addPlugin allows us to add a KeySet to validate plugins with different contracts correctly
- C++ binding now throws std::bad_alloc on allocation problems (and not InvalidName)

## 405.5 CMake

It is now possible to remove a plugin/binding/tools by prefixing a name with "-". The new "-element" syntax is accepted by TOOLS, BINDINGS and PLUGINS. It is very handy in combination with ALL, e.g.:
`-DPLUGINS="ALL;-xmltool"`
will include all plugins except xmltool.

## 405.6 Improved comments

Comment preserving was improved a lot. Especially, the hosts plugin was rewritten completely. Now multiple different comment styles can be intermixed without losing information. E.g. some INI formats support both ; and # for comments. With the new comments it is possible to preserve that information and even better: applications can iterate over that information (metadata).
To mount the new hosts plugin use (if you already have mounted it, you have nothing to do):
```
kdb mount /etc/hosts system/hosts hosts
```
The hosts plugin now separates from ipv4 and ipv6 which makes the host names canonical again, e.g.:
```
kdb get system/hosts/ipv4/localhost
kdb get system/hosts/ipv6/localhost
```
To access the inline-comment, use:
```
kdb getmeta system/hosts/ipv4/localhost "comment/#0"
```
For other meta information, see:
```
kdb lsmeta system/hosts/ipv4/localhost
```
Additionally, a small API for specific metadata operations emerges. These operations will be moved to a separate library and will not stay in Elektra's core library.

## 405.7 Proposal

- lookup options:
  - `KDB_O_SPEC` uses the lookup key as specification
  - `KDB_O_CREATE` creates a key if it could not be found
- `elektraKeyGetMetaKeySet` creates a KeySet from metadata
- `elektraKsFilter` allows us to filter a KeySet arbitrarily (not only `keyIsBelow` in case of `ksCut`). It reintroduces more functional programming.
- `keyGetNamespace` was reintroduced. In one of the next versions of Elektra we will introduce new namespaces. `keyGetNamespace` allows the compiler to output a warning when some namespaces are not handled in your C/C++ code.

## 405.8 Java binding

Elektra now fully supports applications written in Java and also Plugins written in the same language.
The `new binding was developed using jna.` For the `plugin interface JNI` was used. We developed already `some plugins`.

## 405.9 Qt-Gui

Raffael Pancheri released the version 0.0.2 of the Qt-Gui:

- added Backend Wizard for mounting
- user can hover over TreeView items and quickly see key name, key value and metakeys
- it is now easily possible to create and edit arrays
- added header to MetaArea for better clarity
- many small layout and view update fixes

## 405.10 Further stuff and small fixes

- Two new error/warnings information: mountpoint and configfile. It is added automatically and all tools will print it.
- C++ I/O for key(s) now allows null terminator next to new-line terminator
- fix error plugin: now use `on_open/trigger_warnings` to be consistent
- fix metaset: now correctly append new key
- arrays are also available when compiled with mingw (but tests still have to be excluded for successful compilation)
- fix #136
- fix long help text in `kdb check`
- signed release tags are now used

## 405.11 Get It!

You can download the release from here

- size: 1915277
- md5sum: 2b16a4b555bc187562a0b38919d822a1
- sha1: 08b1d0139fc5eb0d03c52408478e68b91b1825dc
- sha256: 526e2ed61e87d89966eb36ddad78d8139b976e01ce18aab340d8a1df47132355

already built API documentation can be found here

## 405.12 Stay tuned!

Subscribe to the new RSS feed to always get the release notifications.
Permalink to this NEWS entry
For more information, see https://www.libelektra.org
Best regards, Markus

# Chapter 406

# 0.8.11 Release

- guid: 7d4647d4-4131-411e-9c2a-2aca39446e18
- author: Markus Raab
- pubDate: Fri, 03 Apr 2015 02:39:37 +0200
- shortDesc: adds specification namespace & numerous improvements

From the beginning of the Elektra Initiative, Elektra aimed at avoiding hard coded information in the application and to make the application's configuration more transparent. While avoiding any paths to files was reality from the first released Elektra version, now also hard coding default values, fallback mechanisms and even Elektra's paths to keys can be avoided.

## 406.1 How does that work?

Elektra 0.8.11 introduces a so-called specification for the application's configuration. It is located below its own namespace `spec` (next to user and system).
Once the base path is known, the user can find out all Elektra paths used by an application, using:

```
kdb ls spec/basepath
```

Keys in `spec` allow us to specify which keys are read by the application, which fallback it might have and which is the default value using metadata. The implementation of these features happened in `ksLookup`. When cascading keys (those starting with `/`) are used following features are now available (in the metadata of respective `spec`-keys):

- `override/#`: use these keys *in favor* of the key itself (note that `#` is the syntax for arrays, e.g. `#0` for the first element, `#_10` for the 11th and so on)
- `namespace/#`: instead of using all namespaces in the predefined order, one can specify which namespaces should be searched in which order
- `fallback/#`: when no key was found in any of the (specified) namespaces the `fallback`-keys will be searched
- `default`: this value will be used if nothing else was found

This technique does not only give you the obvious advantages, but also provides complete transparency how a program will fetch a configuration value. In practice that means that:

```
kdb get "/sw/app/#0/promise"
```

will give you the *exact same value* as the application uses when it lookups the key `promise`. Many `if`s and hard coded values are avoided, we simply fetch and lookup the configuration by following code:

```
Key *parentKey = keyNew("/sw/app/#0", KEY_CASCADING_NAME, KEY_END);
kdbGet(kdb, ks, parentKey);
ksLookupByName(ks, "/sw/app/#0/promise", 0);
```

We see in that example that only Elektra paths are hard coded in the application. But those can be found out easily, completely without looking in the source code. The technique is simple: append a logger plugin and the KDB base path is printed to:

- stdout in the case of the plugin tracer

- syslog in the case of the plugin syslog
- journald in the case of the plugin journald

What we do not see in the program above are the default values and fallbacks. They are only present in the so specification (namespace `spec`). Luckily, the specification are key-value pairs, too. So we do not have to learn something new, e.g. using the ni plugin we can specify:

```
[promise]
default=20
fallback/#0=/somewhere/else
namespace/#0=user
```

1. When this file is mounted to `spec/sw/app/#0` we specify, that for the key `/sw/app/#0/promise` only the namespace `user` should be used.
2. If this key was not found, but `/somewhere/else` is present, we will use this key instead. The `fallback` technique is very powerful: it allows us to have (recursive) links between applications. In the example above, the application is tricked in receiving e.g. the key `user/somewhere/else` when `promise` was not available.
3. The value `20` will be used as default, even if no configuration file is found.

Note that the fallback, override and cascading works on *key level*, and not like most other systems have implemented, on configuration *file level*.

## 406.2 Namespaces

The specification gives the namespaces clearer semantics and purpose. Key names starting with a namespace are connected to a configuration source. E.g. keys starting with:

- `user` are keys from the home directory of the current user
- `system` are keys from the `/etc` directory of the current system
- `spec` are keys from the specification directory (configurable with `KDB_DB_SPEC`, typically `/usr/share/elektra/speci`

When a key name starts with an `/` it means that it is looked up by specification. Such a cascading key is not really present in the keyset (except when a default value was found). They are neither received nor stored by `kdbGet` and `kdbSet`.
Applications shall only lookup using cascading keys (starting with `/`). If no specification is present, cascading of all namespaces is used as before.
Elektra will (always) continue to work for applications that do not have a specification. We strongly encourage you, however, to write such a specification, because:

- it helps the administrator to know which keys exist
- it documents the keys (including lookup behavior and default value)
- and many more advantages to come in future releases..

For a tutorial how to actually elektrify an application and for more background to Elektra, `read this document`.
For a full list of proposed and implemented metadata, `read this document`.

## 406.3 Simplification in the merging framework

As it turned out the concept of very granular merge strategies was hard to understand for users of the three-way merging framework that emerged in the last year's GSoC. While this granularity is desirable for flexibility, we additionally wanted something easy to use. For that reason merge configurations were introduced. These are simply preconfigured configurations for a merger that arrange required strategies for the most common merging scenarios. Especially they make sure that meta merging is handled correctly.
Have a look at the changes in the example `/src/libs/tools/examples/merging.cpp` for an glimpse of the simplifications.

A big thanks to Felix Berlakovich!
The header files will be installed to /usr/include/elektra/merging, but they are subject to be changed in the future (e.g. as they did in this release).
From the merging improvements some minor incompatibility happened in `kdb import`. Not all merging strategies that worked in 0.8.10 work anymore. Luckily, now its much simpler to choose the strategies.

## 406.4 API

The main API kdb.h has two added lines. First a new method was added:

```
ssize_t keyAddName(Key *key, const char *addName);
```

This method is already used heavily in many parts. Contrary to `keySetBaseName` and `keyAddBaseName` it allows us to extend the path with more than one Element at once, i.e. `/` are not escaped.
The other new line is the new enum value `KEY_FLAGS`. This feature allows bindings to use any flags in keyNew without actually building up variable argument lists. (Thanks to Manuel Mausz)
As always, API+ABI is stable and compatible.

## 406.5 Proposed

Many new functions are proposed and can be found in `the doxygen documentation` and in `kdbproposal.h`.
Noteworthy is the method `keyGetNamespace` which allows us to query all namespaces. Since this release we changed every occurrence of namespaces (except documentation) with switch-statements around `keyGetNamespace`. This allows us to add new more namespaces more easily. (Although its currently not planned to add further namespaces.)
Finally, a bunch of new lookup options were added, which might not be useful for the public API (they allow us to disable the specification-aware features mentioned in the beginning).

## 406.6 Obsolete and removed concepts

### 406.6.1 umount

The concept that backends have a name (other than their mountpoint) is now gone. Backends will simply be named with their escaped mountpath below system/elektra/mountpoints without any confusing additional name.
Unmounting still works with the mountpath.
Removing this concept makes Elektra easier to understand and it also removes some bugs. E.g. having mountpoints which do not differ except having a _ instead of a `/` would have caused problems with the automatic name generation of Elektra 0.8.10.
Old mountpoints need to be removed with their 0.8.10 name (_ instead of `/`).

### 406.6.2 directory keys

Additionally, the so-called directory keys were also removed. Elektra was and still is completely key-value based. The `/` separator is only used for mountpoints.

### 406.6.3 fstab

The plugin fstab is also improved: Slashes in mountpoints are escaped properly with the internal escaping engine of keyAddBaseName() (i.e. without any problematic `/` replacements).

### 406.6.4 dirname

getDirName() was removed from C++, gi-lua, gi-python2, gi-python3, swig-lua, swig-python2 and swig-python3. It was never present in C and did not fit well with keyBaseName() (which returns an unescaped name, which is not possible for the dirname). (Thanks to Manuel Mausz)

### 406.6.5 invalid parent names

While empty (=invalid) names are still accepted as parentName to `kdbGet` and `kdbSet` for compatibility reasons, but the parentKey

```
Key *parentKey = keyNew("/", KEY_END);
```

should be used instead (if you want to get or store everything). They have identical behavior, except that invalid names (that cannot be distinguished from empty names) will produce a warning. In the next major version invalid parentNames will produce an error.

## 406.7 KDB Behavior

It is now enforced that before a kdbSet() on a specific path a kdbGet() on that path needs to be done. This was always documented that way and is the only way to correctly detect conflicts, updates and missing configuration files. Error #107 will be reported on violations.
Additionally, the handling with missing files was improved. Empty keysets for a mountpoint now will remove a file. Such an empty file is always up-to-date. Removing files has the same atomicity guarantees as other operations.
The concurrency behavior is at a very high level: as expected many processes with many threads can each concurrently write to the key database, without any inconsistent states: This is noted here because Elektra works on standard configuration files without any guarding processes.
Filesystem problems, e.g. permission, now always lead to the same errors (#9, #75, #109, #110), regardless of the storage plugin.

## 406.8 Qt-Gui 0.0.6

Raffael Pancheri was very busy and did a lot of stabilizing work:

- Added Markdown converter functionality for plugin documentation
- Integrated help (Whats this?)
- Added credits to other authors
- do not show storage/resolver plugins if a plugin of that kind has been selected
- added menu to newkey toolbar button to allow new array entries
- added option to include a configuration keyset when adding a plugin
- show included keys when creating the plugin configuration
- Added all storageplugins to namefilters
- Reimplement ErrorDialog
- Added undo/redo of all commands and correctly update the view
- modified ToolTip size
- Color animation on search results
- Refactored Buttons to accept shortcuts
- Updated Translations
- Colors are now customizeable
- Many small fixes

The gui is already used and the remaining small bugs (see github) are going to be fixed soon. One of the highlights is undo for nearly every action, so nothing prevents you from trying it out!
A huge thanks to Raffael Pancheri for his contributions. His thesis can be found at here.

## 406.9 Bug fixing

- fix issues with escaped backslashes in front of slashes
- atomic commits across namespaces work again
- memleak on ReadFile error in ni plugin
- `kdb getmeta` reports errorcode if key, but no meta was found
- `ksLookup` now will also work if a key of the keyset is used as search-key (aliasing problem fixed by dup() on namelock)
- resolver regex does not match to wrongly written plugins
- jna plugin is now named libelektra-0.8.11.jar, with proper symbolic link to current version, for every CMake version
- fix bashism (`$RANDOM`)
- new keys are correctly renamed, fixes OpenICC (thanks to Felix Berlakovich)
- comments in host keys are correctly restored (thanks to Felix Berlakovich)
- output stream in type checking is no longer locale dependent (thanks to Manuel Mausz)
- CMake uninstall works again
- simplify `CMAKE_DL_LIBS` (thanks to Manuel Mausz)

## 406.10 Further gems

- Examples were improved, added (e.g. cascading, namespace) and included in `Doxygen documentation`.
- `METADATA specification` was nearly completely rewritten (thanks to Felix Berlakovich)
- benchmarks were greatly enhanced (run-time+heap profiling), and some important performance improvements were done
- All plugins now use the CMake function `add_plugin` (thanks to Ian Donnelly for most of the work)
- data directory (keysets as C-files) is now shared between different kinds of test suites.
- many more tests were added, e.g. distribution tests, KDB API tests; and allocation tests were readded
- now all kdb commands accept cascading keys.
- More compiler warning-flags are added and many warnings are fixed
- cleanup of old unused `keyName` methods
- The key `system/elektra/mountpoints` itself was always created and a left-over on a freshly installed system after the unit tests run the first time. The physical presence of the key is now irrelevant and it won't be created automatically.
- Bash completion was greatly improved (thanks to Manuel Mausz)
- Configure scripts were refactored and are now much shorter (thanks to Manuel Mausz)
- New Debian build agents were added that are magnitutes faster than the old ones (a big thanks to Manuel Mausz)
- Many KDB tests, written in C, lua and python were added (thanks to Manuel Mausz)
- SWIG3 is preferred when available
- add the plugin counter that counts how often the methods of a plugin are called

- `kdb list-tools` is now advertised in `kdb --help`
- macOS support was greatly improved, thanks to Peter Nirschl and Kai-Uwe Behrmann. The feature rich resolver, now also works for macOS. wresolver is now only needed for mingw.
- Elektra still compiles with gcc (also mingw variants), icc and clang.

## 406.11 Further Notes

With 471 files changed, 27978 insertions(+), 11512 deletions(-) this release is huge. With 773 commits over four month much more changes happened which did not find their place in these release notes, even though the notes are much less detailed than usual.
Thanks for all contributions that are not enlisted here!
For any questions and comments, please contact the Mailing List or elektra@markus-raab.org.

## 406.12 Get It!

You can download the release from here

- name: elektra-0.8.11.tar.gz
- size: 2022129
- md5sum: c53a8151aab760851842d745e904a4f8
- sha1: d7929d17d1a6529089d156f1910d87f678b84998
- sha256: c20fefcfba62cc906228f9b55d1f411ef8f884ff9d75774a0dd4f8eb8f5b48f6

This release tarball now is also available signed by me using gpg
already built API documentation can be found here

## 406.13 Stay tuned!

Subscribe to the new RSS feed to always get the release notifications.
Permalink to this NEWS entry
For more information, see https://www.libelektra.org
Best regards, Markus

# Chapter 407

# 0.8.12 Release

- guid: 98770541-32a1-486a-98a1-d02f26afc81a
- author: Markus Raab
- pubDate: Sun, 12 Jul 2015 20:14:09 +0200
- shortDesc: adds directory namespace & other improvements and fixes

Again we managed to release with new features, many build system fixes and also other improvements.

## 407.1 dir namespace

This namespace adds per-project or per-directory (hence the name) configurations. E.g. think how Git works: not only /etc and ~ are relevant sources for configuration but also the nearest .git directory.
This technique is, however, much more widely useful than just for git repositories! Nearly every application can benefit from such a per-dir configuration. Its almost certain that you have already run into the problem that different projects have different guidelines (e.g. coding conventions, languages, whitespace requirements, line breaks, ..). Obviously, thats not a per-user configuration and its also not a per-file issue (thats how its usually solved today). So in fact you want, e.g., your editor to have an additional per-project layer to choose between such settings.
The technique is useful for nearly every other tool:

- different color palettes in gimp, inkscape,..
- different languages for LibreOffice
- different security settings for media players, interpreters (e.g. when in Download folder)
- per-folder .htaccess in apache or other web servers
- any other per-dir configuration you can imagine..

It is simple to use, also for the administrative side. First, change to the folder to your folder (e.g. where a project is):
```
cd ~/projects/abc
```
Then add some user (or system or spec) configuration to have some default.
```
kdb set user/sw/editor/textwidth 72
```
Then verify that we get this value back when we do a cascading lookup:
```
kdb get /sw/editor/textwidth
```
The default configuration file for the dir-namespace is `pwd`/KDB_DB_DIR/filename:
```
kdb file dir/sw/editor/textwidth
```

- KDB_DB_DIR can be modified at compile-time and is `.dir` per default
- filename can be modified by mounting, see below, and is `default.ecf` by default

We assume, that the project abc has the policy to use textwidth 120, so we change the dir-configuration:
```
kdb set dir/sw/editor/textwidth 120
```
Now we will get the value 120 in the folder ~/projects/abc and its subdirectories (!), but everywhere else we still get 72:
```
kdb get /sw/editor/textwidth
```
Obviously, that does not only work with kdb, but with every elektrified tool.

### 407.1.1 mount files in dir namespaces

For cascading mountpoints, the dir name is also automatically mounted, e.g.:
```
kdb mount editor.ini /sw/editor ini
```
But its also possible to only mount for the namespace dir if no cascading mountpoint is present already:
```
kdb mount app.ini dir/sw/app tcl
```
In both cases keys below dir/sw/editor would be in the INI file `.dir/editor.ini` and not in the file `.dir/default.ecf`.

### 407.1.2 dir together with spec namespace

In the project P we had the following issue: We needed on a specific computer the configuration in /etc to be used in favor of the dir config.
We could easily solve the problem using the specification:
```
kdb setmeta spec/sw/P/current/org/base override/#0 /sw/P/override/org/base
```
Hence, we could create system/sw/P/override/org/base which would be in favor of dir/sw/P/current/org/base. So we get system/sw/P/override/org/base when we do:
```
kdb get /sw/P/current/org/base
```
Alternatively, one could also use the specification:
```
kdb setmeta spec/sw/P/current/org/base namespace/#0 user
kdb setmeta spec/sw/P/current/org/base namespace/#1 system
kdb setmeta spec/sw/P/current/org/base namespace/#2 dir
```
Which makes dir the namespace with the least priority and system would be preferred. This was less suitable for our purpose, because we needed the override only on one computer. For all other computers we wanted dir to be preferred with only one specification.

### 407.1.3 Conclusion

The great thing is, that every elektrified application, automatically gets all the mentioned features. Not even a recompilation of the application is necessary.
Especially the specification provides flexibility not present in other configuration systems.

## 407.2 Qt-Gui 0.0.7

Raffael Pancheri again did a lot of stabilizing work:

- show errormessage on exception when starting gui
- Correctly update keyAreaView property when selecting item in TreeView
- Fix crash when creating key in MountingWizard
- Remove information on successful export
- Show error dialog on failed import
- Remove namefilters (every syntax can have any file extension)
- other namespaces (including dir) are included

The GUI can be handy for many purposes, e.g. we use it already as xml and json editor. Note that there are still some bugs.

## 407.3 Other fixes

- constants now additionally gives information about SPEC and DIR.
- Doku about CMake variables `ELEKTRA_DEBUG_BUILD` and `ELEKTRA_VERBOSE_BUILD` fixed, thanks to Kurt Micheli
- Fixed compilation of `ELEKTRA_DEBUG_BUILD` and `ELEKTRA_VERBOSE_BUILD`, thanks to Manuel Mausz

- Example with error handling added, thanks to Kurt Micheli
- Add design decision about global plugins
- Split dependencies document to individual README.md, thanks to Ian Donnelly
- Fix nearly all compilation warnings of SWIG, thanks to Manuel Mausz
- CMake: Fix gtest to be build if `BUILD_TESTING` activated, but not `ENABLE_TESTING`
- CMake: Allow compilation without BUILD_STATIC
- Explain compilation options more, thanks to Kai-Uwe Behrmann for asking the question
- CMake: always build examples, allow one to only build documentation
- add common header file for C++ plugins (used by plugins struct and type)
- fix compilation of race tool under oS-11.4 thanks to Kai-Uwe Behrmann
- CMake: find python3 correctly
- CMake: fix BUILD_SHARED_LIBS
- Doxygen: remove `HTML_TIMESTAMP` to make build reproduceable
- Doxygen: rewrite of main page+add info about all five namespaces
- CMake: allow one to use qt-gui with qt built with -reduce-relocations
- fix kdb ls, get to list warnings during open
- during kdbOpen() use Configfile: to state phase
- add -f option to kdb check+improve docu
- improve readability of warning output
- run_all always uses dump for backups
- line plugin round-trips correctly
- untypical resolvers have their non-existant filename handled correctly + sync ignored them correctly
- cmake-3.0 fixes
- cascading merging, a big thanks to Felix Berlakovich for the last minute fix

## 407.4 Compatibility

As always, the API and API is fully compatible. Because nothing was added, its even possible to link against an older version of Elektra (if compiled against 0.8.12).
In plugins some small changes may effect compatibility:

- in rename the handling of parent key is different (see #206)
- resolving of spec absolute and relative paths are no more handled identical. Instead absolute paths will be searched absolutely, while relatives are below KDB_DB_SPEC (as before). This behavior is consistent to the behavior of the other namespaces.

These two points are also the only unit tests that fail when Elektra 0.8.12 is used with 0.8.11 unit tests.

## 407.5 Build Server

- special GitHub command to build bindings "jenkins build bindings please", thanks to Manuel Mausz
- open build service update For `OpenSUSE, CentOS, Fedora, RHEL and SLE` Kai-Uwe Behrmann kindly provides packages `for download`.

## 407.6 Get It!

You can download the release from here and now also here on GitHub

- name: elektra-0.8.12.tar.gz
- size: 2102450
- md5sum: a40a33ae6661ebfa096378f0986ede6c
- sha1: 3594ef58b6e3b0ffa9589d787679b6e739fbb0dd
- sha256: 562432bea9455a61ff6e6b3263078ea9b26bef2ed177a04b5f9b181d605bc021

This release tarball now is also available signed by me using gpg
already built API documentation can be found here

## 407.7 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the Mailing List the issue tracker on GitHub or by mail elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Markus

# Chapter 408

# 0.8.13 Release

- guid: 3c00a5f1-c017-4555-92b5-a2cf6e0803e3
- author: Markus Raab
- pubDate: Thu, 17 Sep 2015 17:32:16 +0200
- shortDesc: adds elektrify-getenv & other improvements

Again we managed to release with many new features, many fixes and also other improvements.

## 408.1 Elektrify-getenv

getenv(3) is one of the most popular ways to retrieve configuration, even though it has many known problems:

- no standard way to modify it
- relogin (or restart of shell) necessary
- names are flat (no hierarchical structure)
- cannot be set for individual applications
- different in at, cron and similar scripts

With elektrify-getenv we wrote a solution which solves most of the problems. We use the `LD_PRELOAD` technique to *additionally* retrieve values from Elektra, and not only the environment.
You simply can do:

```
kdb set user/env/override/HTTP_PROXY "http://my.proxy:8080"
```

This will set the `HTTP_PROXY` environment variable to `http://my.proxy:8080`. Configuration can be retrieved with `kdb get`:

```
kdb get user/env/override/HTTP_PROXY
lynx   # or start another www-browser with the newly set HTTP_PROXY
```

Or using the man pages:

```
kdb elektrify-getenv man man --elektra:MANWIDTH=40
```

Will use MANWIDTH 40 for this invocation of man man. This feature is handy, if an option is only available by environment, but not by command-line arguments, because sometimes environment variables are not trivial to set (e.g. in Makefiles).
Some more examples:

```
kdb set user/env/override/MANOPT -- "--regex -LC"
kdb elektrify-getenv getenv MANOPT   # to check if it is set as expected
kdb getenv MANOPT   # if /etc/ld.so.preload is active
```

So is this the final solution for configuration and manual elektrification of applications is not needed anymore?
The answer is: no and yes.
It is quite satisfactory for configuration that is inherently sharable (not different from one application to another) *and* needs the environment semantics, i.e. some subprocesses should have different configuration than others, e.g. in a specific terminal.
But it might not be a good solution for your own application, because libgetenv(3) implies many architectural decision, that other elektrified applications would decide differently, e.g.:

- it uses global variables (getenv(3) has no handle)
- it uses mutex for multi-threading safety
- the API getenv(3) only returns `char*` and has no support for other data types

For more information see `src/libgetenv/README.md`

## 408.2 Compatibility

As always, the API and API is fully forward-compatible, i.e. programs compiled against an older 0.8 versions of Elektra will continue to work.
Because `keyUnescapedName` and `keyGetUnescapedNameSize` is added in this release, it is not backward-compatible, i.e. programs compiled against 0.8.13, might *not* work with older 0.8 libraries.
The function `keyUnescapedName` provides access to an unescaped name, i.e. one where `/` and `\\` are literal symbols and do not have any special meaning. `NULL` characters are used as path separators. This function makes it trivial and efficient to iterate over all path names, as already exploited in all bindings:

- `jna (java)`
- `lua`
- python2
- `python3`

Other small changes/additions in bindings:

- fix key constructor, thanks to Manuel Mausz
- add copy and deepcopy in python (+examples,+test cases), thanks to Manuel Mausz
- dup() in python3 returned wrong type (SWIG wrapper), thanks to Toscano Pino for reporting, thanks to Manuel Mausz for fixing it

Doxygen 1.8.8 is preferred and the configfile was updated to this version.
The symbols of nickel (for the ni plugin) do not longer leak from the Elektra library. As such, old versions of testmod_ni won't work with Elektra 0.8.13. A version-script is now in use to only export following symbols:

- `kdb*`
- `key*`
- `ks*`
- `libelektra*` for module loading system
- `elektra*` for proposed and other functions (no ABI/API compatibility here!)

In this release, ENABLE_CXX11 was changed to `ON` by default.
Note that in the next release 0.8.14 there will be two changes:

- According to `issue #262`, we plan to remove the option ENABLE_CXX11 and require the compiler to be C++11 compatible. If you have any system you are not able to build Elektra with -DENABLE_CXX11=ON (which is the default for 0.8.13) please report that immediately.
- the python3 bindings will be renamed to python

By not having to care for pre-C++11 compilers, we hope to attract more developers. The core part is still in C99 so that Elektra can be used on systems where libc++ is not available. Many new plugins are still written in C99, also with the purpose of not depending on C++.

## 408.3 Python Plugins

A technical preview of `python3` and python2 plugins has been added.
With them its possible to write any plugin with python scripts.
Note, they are a technical preview. They might have severe bugs and the API might change in the future. Nevertheless, it is already possible to, e.g. develop storage plugins with it.
They are not included in `ALL` plugins. To use it, you have to specify it:
`-PLUGINS="ALL;python;python2"`
Thanks to Manuel Mausz for to this work on the plugins and the patience in all the last minute fixes!

## 408.4 Qt-gui 0.0.8

The GUI was improved and the most annoying bugs are fixed:

- only reload and write config files if something has changed
- use merging in a way that only a conflict free merge will be written, thanks to Felix Berlakovich
- made sure keys can only be renamed if the new name/value/metadata is different from the existing ones
- fixed 1) and 2) of #233
- fixed #235
- fixed qml warning when deleting key
- fixed qml typerror when accepting an edit

A big thanks to Raffael Pancheri!

## 408.5 KDB Tool

The commandline tool `kdb` also got some improvements. Most noteworthy is that `kdb get -v` now gives a complete trace for every key that was tried. This is very handy if you have a complex specification with many fallback and override links.
It also shows default values and warnings in the case of context-oriented features.
Furthermore:

- Add `-v` for setmeta
- Copy will warn when it won't overwrite another key (behavior did not change)
- improve help text, thanks to Ian Donnelly

## 408.6 Documentation Initiative

As Michael Haberler from `machinekit` pointed out it was certainly not easy for someone to get started with Elektra. With the documentation initiative we are going to change that.

- The discussion in `GitHub issues` should clarify many things
- We start writing man pages in ronn-format(7), thanks to Ian Donnelly for current work
- Kurt Micheli is woring on improved doxygen documentation + pdf generation
- Daniel Bugl already restructed the main page
- Daniel Bugl also improved formatting
- doc: use

**Return values**

| | |
|---|---|
| *more,thanks* | to Pino Toscano |

- doxygen: fix template to use `@` and not `\\`.
- SVG logo is preferred, thanks to Daniel Bugl
- doc: use

**Return values**

| | |
|---|---|
| *more,thanks* | to Pino Toscano |

- many typo fixes, thanks to Pino Toscano
- fix broken links, thanks to Manuel Mausz, Daniel Bugl and Michael Haberler

Any further help is very welcome! This call is especially addressed to beginners in Elektra because they obviously know best which documentation is lacking and what they would need.

## 408.7 Portability

`kdb-full` and `kdb-static` work fine now for Windows 64bit, thanks to Manuel Mausz. The wresolver is now more relaxed with unset environment.
All issues for macOS were resolved. With the exception of elektrify-getenv everything should work now, thanks to Mihael Pranjic:

- fix mktemp
- testscripts
- recursive mutex simplification
- clearenv ifdef

and thanks to Daniel Bugl:

- RPATH fixed, so that `kdb` works

furthermore:

- fix `__FUNCTION__` to `__func__` (C99), thanks to Pino Toscano
- avoid compilation error when JNI_VERSION_1_8 is missing
- fix (twice, because of an accidental revert) the TARGET_CMAKE_FOLDER, thanks to Pino Toscano

Thanks to Manuel Mausz for to testing and improving portability!

## 408.8 Packaging and Build System

- `0.8.12 packaged+migrated to testing`, thanks to Pino Toscano
- fix build with external gtest, thanks to Pino Toscano
- switch from FindElektra.cmake to ElektraConfig.cmake, thanks to Pino Toscano
- use `cmake_parse_arguments` instead of `parse_arguments`, thanks to Manuel Mausz

## 408.9 Further Fixes

- Key::release() will also work when Key holds a null-pointer
- Key::getName() avoids std::string exception
- support for copy module was introduced, thanks to Manuel Mausz
- be more POSIX compatible in shell scripts (`type` to `command -v` and avoid `echo -e`) thanks to Pino Toscano
- fix vararg type for KEY_FLAGS, thanks to Pino Toscano
- fix crash of example, thanks to Pino Toscano
- add proper licence file for Modules (COPYING-CMAKE-SCRIPTS), thanks to Pino Toscano
- fix XDG resolver issue when no given path in XDG_CONFIG_DIRS is valid
- make dbus example work again
- fix compiler warnings for gcc and clang
- fix Valgrind suppressions
- Installation of GI binding is fixed, thanks to Dāvis
- make uninstall is fixed and documentation improved

## 408.10 Notes

There are some misconceptions about Elektra and semi structured data (like XML, JSON). Elektra is a key-value storage, that internally represents everything with key and values. Even though, Elektra can use XML and JSON files elegantly, there are limitations what XML and JSON can represent. XML, e.g., cannot have holes within its structure, while this is obviously easily possible with key-value. And JSON, e.g., cannot have non-array entries within an array. This is a more general issue of that configuration files in general are constrained in what they are able to express. The solution to this problem is validation, i.e. keys that does not fit in the underlying format are rejected. Note there is no issue the other way round: special characteristics of configuration files can always be captured in Elektra's metadata.

## 408.11 Get It!

You can download the release from `here` and now also `here on GitHub`

- name: elektra-0.8.13.tar.gz
- size: 2141758
- md5sum: 6e7640338f440e67aba91bd64b64f613
- sha1: ca58524d78e5d39a540a4db83ad527354524db5e
- sha256: f5c672ef9f7826023a577ca8643d0dcf20c3ad85720f36e39f98fe61ffe74637

This release tarball now is also available `signed by me using gpg`
already built API documentation can be found `here`

## 408.12 Stay tuned!

Subscribe to the `RSS feed` to always get the release notifications.
For any questions and comments, please contact the `Mailing List` the issue tracker `on GitHub` or by mail `elektra@markus-raab.org`.
`Permalink to this NEWS entry`
For more information, see `https://libelektra.org`
Best regards, Markus

# Chapter 409

# 0.8.14 Release

- guid: 519cbfac-6db5-4594-8a38-dec4c84b134f
- author: Markus Raab
- pubDate: Thu, 19 Nov 2015 17:48:14 +0100
- shortDesc: adds massive documentation improvements & new or overhauled plugins

Again we managed to release with many new features and plugins (lua, enum, list, crypto, csvstorage, conditionals, mathcheck, filecheck, logchange) many fixes, and especially with a vastly improved polished documentation.

## 409.1 Documentation Initiative

The Documentation Initiative is a huge success and now the documentation of Elektra is in a state where someone (preferable a linux guru), never heard of Elektra, still can use Elektra only by reading man pages.
There are now many ways to show a man page:

- `on GitHub`
- `in the API documentation`
- by using `kdb --help` or `kdb help <command>`
- by using `man kdb`

### 409.1.1 Help system

Ian Donnelly wrote man pages for all the tools delivered with Elektra. Additionally, nearly all README.md are now also converted to man pages and also to Doxygen.

### 409.1.2 Doxygen Filter

Kurt Micheli did an amazing work with a new doxygen filter. The filter allows all Elektra Markdown pages to be also included in the doxygen documentation. Thus all technical concepts are now explained in Markdown pages, this filter is essential.
But even more, the filter also includes all man pages written for the tools, giving a nice html view for them. (In addition to the Markdown rendering on GitHub).
Enjoy the `result`.
A big thanks to Kurt Micheli!

### 409.1.3 Further Documentation fixes

- getenv debugging documentation was improved
- typo fix: Specify, thanks to Pino Toscano

- Design decisions Definition of Bool, capabilities and Publish Subscribe (thanks to Daniel Bugl)
- Improve iconv docu
- usage examples for many plugins
- improve README for line plugin (thanks to Ian Donnelly)
- add documentation about dependencies for some plugins (thanks to Ian Donnelly)
- create many new links within the documentation

## 409.2 Simplicity

We shifted our goals a bit: We want to prefer simplicity to flexibility. Not because we do no like flexibility, but because we think we achieved enough of it. Currently (and in future) you can use Elektra:

- obviously as primitive key-value storage
- with specifications and dozens of plugins driven by it
- with code generation
- ...

But we cut flexibility regarding:

- namespaces are only useful for configuration (not for arbitrary key-value)
- the semantics of metadata
- mounting functionality
- error code meanings are fixed, if a resolver detects a conflict, our defined error must be used
- of course ABI, API

## 409.3 Qt-gui 0.0.9

Raffael Pancheri again updated his qt-gui to version 0.0.9 (beta) with important of fixes and improvements:

- Fixes for Qt 5.5
- Handling of merge-conflicts improved
- Avoid rewriting on merge-conflicts
- Allow QML to destroy C++ owned model
- Dialog at startup
- Reduce memory footprint
- add man page

A bit thanks to Raffael Pancheri!

## 409.4 Compatibility

As always, the API and API is fully forward-compatible, i.e. programs compiled against an older 0.8 versions of Elektra will continue to work.
The behavior of some plugins, however, changed:

- the INI plugin, the section handling was improved.
- in the NI plugin, the symbol Ni_GetVersion vanished
- in the resolver plugin files of other namespaces which are not mounted are not resolved anymore

### 409.4.1 Build System

ENABLE_CXX11 does not exist anymore, it is always on. We do not care about 199711L compilers anymore, which makes development easier, without losing any actually used platform.
Some programs that are only used in-source are not installed anymore. (by Pino Toscano)
Python and Lua plugins are enabled now in `-DPLUGINS=ALL`.
Python3 plugin was renamed to python.

## 409.5 Lua Plugin

Manuel Mausz add a lightweight alternative to the python plugin: the lua plugin. In a similar way, someone can write scripts, which are executed on every access to the key database.
To mount a lua based filter, you can use:

```
kdb mount file.ini /lua ini lua script=/path/to/lua/lua_filter.lua
```

Even though it works well, it is classified as technical preview.
Thanks to Manuel Mausz for this plugin!

## 409.6 Cryptography Plugin

In this technical preview, Peter Nirschl demonstrates how a plugin can encrypt Elektra's values. In test cases it is already able to do so, but for the end user an easy way for key derivation is missing.
A big thanks to Peter Nirschl!

## 409.7 INI Plugin

The INI plugin got a near rewrite. Now it handles many situations better, has many more options and features, including:

- preserving the order
- using keys as metadata
- many new test cases
- fix escaping

Thanks to Thomas Waser for this work!

## 409.8 Mathcheck plugin

This plugin allows you to check and even calculate keys from other keys using an polish prefix notation. It supports the typical operations `+ - / *` and `<, <=, ==, !=, =>, >, :=`. To mount, check and calculate values, one would use:

```
kdb mount mathcheck.dump /example/mathcheck dump mathcheck
kdb setmeta user/example/mathcheck/k check/math "== + a b"
kdb setmeta user/example/mathcheck/k check/math ":= + a b"
```

For details see the documentation.
Thanks to Thomas Waser for this important plugin!

## 409.9 List Plugin

Currently, Elektra has some limitations on how many plugins can be added to certain placement. Because of the rapidly growing number of plugins, some combinations are not possible anymore.
This plugin tackles the issue, by delegating the work to an arbitrary number of subplugins. As a bonus, it works lazily and thus might avoid the loading of some plugins all together.
Thanks to Thomas Waser for this plugin!

## 409.10 Conditionals

Brings `if` inside Elektra. It lets you check if some keys have the values they should have.

```
kdb mount conditionals.dump /tmount/conditionals conditionals dump
kdb set user/tmount/conditionals/fkey 3.0
kdb set user/tmount/conditionals/hkey hello
kdb setmeta user/tmount/conditionals/key check/condition "(hkey == 'hello') ? (fkey == '3.0')" # success
kdb setmeta user/tmount/conditionals/key check/condition "(hkey == 'hello') ? (fkey == '5.0')" # fail
```

For details see the documentation.
Again, thanks to Thomas Waser for this plugin!

## 409.11 Csvstorage Plugin

You can now mount csv-files. To mount `test.csv` simply use:

```
kdb mount test.csv /csv csvstorage
```

There are many options, e.g. changing the delimiter, use header for the key names or predefine how the columns should be named. For details see the documentation.
Thanks to Thomas Waser!

## 409.12 Filecheck plugin

This plugin lets you validate lineendings, encodings, null, bom and unprintable characters.
The also new plugin lineendings is already superseded by the filecheck plugin.
Thanks to Thomas Waser!

## 409.13 Enum plugin

The Enum plugin checks string values of Keys by comparing it against a list of valid values.
Thanks to Thomas Waser!

## 409.14 Elektrify Machinekit.io

We are proud that Machinekit starts using Elektra.
Alexander Rössler is digging into all details, and already enhanced the DBUS Plugin for their needs. DBus now can emit a message for every changed key.
A big thanks to Alexander Rössler!

## 409.15 Bugfixes

- libgetenv did not reinitalized its mutexes on forks
- add needSync also in C++ binding
- handle removed current working directories (fallback to /)
- avoid segfault on missing version keys (when doing `kdb rm system/elektra/version`)
- fix glob plugin + kdb mount with `config/needs usage`
- fix different handling of strerror_r in macOS (thanks to Daniel Bugl)
- do not change the users parentKey in early-error scenarios
- do not try to interpret some binary keys as function keys

## 409.16 Other Gems

- getenv example: do not link to elektra/elektratools, thanks to Pino Toscano
- fixes in other examples
- avoid useless UTF-8 chars and fix typos, thanks to Kurt Micheli
- fix kdb check return code (open fail)
- pdf now also allows UTF-8 characters if added to elektraSpecialCharacters.sty, thanks to Kurt Micheli
- libgetenv: lookup also used for layers
- handle wrong arguments of metals better, thanks to Ian Donnelly
- Improvement of error messages in the augeas plugin
- `kdb set` avoids fetching unnecessary namespaces
- verbose unmount
- logchange: small demonstration plugin to show how to log added, removed and changed keys
- setmeta will use spec as default
- libtools: avoid useless getName, add verbosity flag for findBackend
- Improve iconv error messages
- That mount needs permissions to /etc should now really be obvious with new error message
- many fixes in the template for new plugins

## 409.17 Get It!

You can download the release from `here` and now also `here on GitHub`

- name: elektra-0.8.14.tar.gz
- size: 2252008
- md5sum: a87cd3845e590bf413959dfd555e3704
- sha1: 2360603c347ae3f3a28e827eb9260ff0b9881e46
- sha256: af681a38c9c2921b8d249f98ab851c3d51371735471d8a1f833a224c4446fe2e

This release tarball now is also available `signed by me using gpg`
already built API documentation can be found `here`

## 409.18 Stay tuned!

Subscribe to the `RSS feed` to always get the release notifications.
For any questions and comments, please contact the `Mailing List` the issue tracker `on GitHub` or by mail `elektra@markus-raab.org`.
`Permalink to this NEWS entry`
For more information, see `https://libelektra.org`
Btw. the whole release happened with `elektrify-getenv` enabled.
Best regards, Markus

# Chapter 410

# 0.8.15 Release

- guid: 1ab4a560-c286-46d2-a058-1a8e7e208fe8
- author: Markus Raab
- pubDate: Tue, 16 Feb 2016 17:47:00 +0100
- shortDesc: adds improved mounting, split of library & configuration validation

In case you do not yet know about it, here is an abstract about Elektra:
Elektra serves as a universal and secure framework to access configuration parameters in a global, hierarchical key database. Elektra provides a mature, consistent and easily comprehensible API. Its modularity effectively avoids code duplication across applications and tools regarding configuration tasks. Elektra abstracts from cross-platform-related issues and allows applications to be aware of other applications' configurations, leveraging easy application integration.
See `https://libelektra.org`

## 410.1 Overview

This is one of the largest release up to now. It includes many user-visible improvements. Some highlights:

- Mounting is vastly improved: think about Debian's "dpkg" to "apt"-like functionality
- In previous releases you could already specify the structure of the configuration. Now you can also automatically enforce this property.
- We split the shared library `libelektra` into smaller parts. Now users can link against the parts of the library they need.
- As always, the ABI and API is fully forward-compatible.
- The release contains improvements in the `bootstrapping process`.
- We improved the `ini`, `rename` and `crypto` plugins.
- The tool `kdb` now supports bookmarks and profiles.
- The new tool `kdb editor` allows you to edit KDB configuration in your favorite text editor.
- We are glad of the new packages for Debian, Arch Linux and OpenWRT.

The same text as follows is also available `here as html` and `here on GitHub`

## 410.2 Global Mount

Sometimes you simply want some functionality for the whole key database. For example, you want to enable logging or notification of configuration changes. In previous versions, you had to change every mountpoint individually. Even more problematic, every mountpoint created its individual logs and notifications, without any way for someone to know if an application has issued its last log/notification.
These problems are now solved by global plugins. The same plugins are reused for this purpose. Also the mounting works nearly in the same way, you only have to omit the configuration file and the mountpoint:
```
kdb global-mount syslog journald dbus
```
Voilà, from now on every configuration change gets logged to syslog and journald. Additionally, every application gets notified via dbus.
If you want it more verbose for debugging, you can easily do so by:
```
kdb global-mount logchange counter
```
Which gives you detailed information to standard output which keys were changed/edited/deleted. Additionally, Elektra counts how often the `counter` plugin is invoked.
The underlying work for the global plugins, i.e. hooks in the core libraries and the `list` plugin that allows us to use many plugins without bloating the core was done by Thomas Waser.
It was already possible in earlier versions of Elektra to specify the configuration of your program. Until now, this specification could be mainly used to to generate code as described here. This release adds two more interesting options:

1. the spec plugin, and
2. the spec mount.

## 410.3 Spec Plugin

The most important global plugin that is now newly introduced with this release (thanks to Thomas Waser) is the `spec` plugin. By default it will be added for every global-mount. So for a new installation make sure you executed at least once:
```
kdb global-mount
```
The spec plugin is a global plugin that copies metadata from the `spec`-namespace to other namespaces. That means, it reads the specification, and makes sure that the configuration conforms to it. The actual validation is done by the many validation plugins already present.
Lets start by saying a key is a long and must have at least the value 10:
```
kdb setmeta spec/example/longkey check/type long
```
Then we can create a key in a different namespace and see if the `spec` plugin applies the metadata correctly:
```
kdb set /example/longkey 25
kdb lsmeta /example/longkey
```
Should now at least print `check/type`. By itself, this is useful for documentation of keys. For example, the application developer says:
```
kdb setmeta spec/example/longkey description "Do not change"
kdb setmeta spec/example/longkey example 30
```
The user can retrieve this documentation by:
```
kdb getmeta /example/longkey description
```
But we want `check/type` to be not only a documentation, but also enforced.

## 410.4 Spec Mount

Using `kdb setmeta` extensively and always looking out that all plugins are mounted correctly is error-prone. So instead, one can directly mount the plugins as specified. For the example above one simply needs:
```
kdb setmeta spec/example mountpoint example.ecf
kdb spec-mount /example
```
Now, when we try to modify `/example/longkey` it will be validated:
```
kdb set /example/longkey a
#> Error (#52) [...] long [not] matched [...] a
```
Based on the present metadata, the correct plugins (in this case `type` because of the metadata `check/type`) will be loaded.
We can also create a whole specification file, first mount the specification and then the mountpoint according the specification, e.g when we have `battery.ini` in the folder `$(dirname $(kdb file spec))` with following content:
```
[]
```

```
mountpoint = battery.ini
infos/plugins = ini
[level]
check/enum = 'critical', 'low', 'high', 'full'
```

Then we can use:

```
# mount the file itself:
kdb mount battery.ini spec/example/battery ni
# make sure all plugins are present (e.g. enum for check/enum):
kdb spec-mount /example/battery
```

Now lets verify if it worked:

```
kdb lsmeta /example/battery/level
# we see it has a check/enum
kdb getmeta /example/battery/level check/enum
# now we know allowed values
kdb set /example/battery/level low
# success, low is ok!
kdb set /example/battery/level wrong
# fails, not one of the allowed values!
```

The main idea of the spec-mount is: search a plugin for every specification (metadata) found in the spec-namespace.

## 410.5 General Mount Improvements

In earlier versions `kdb mount` failed when plugin dependencies were not satisfied. Now dependencies will automatically be fulfilled, e.g.

```
kdb mount /etc/modules system/modules line
```

In earlier versions you would have got an error because of the missing `null` plugin. Now it simply adds the needed plugins.

The plugins given in the command-line used to be real plugins. Now also so-called providers are accepted.

To make providers useful we need to actually know which plugin is the best candidate. To get the information we added a `infos/status` clause in the contract. In this clause the plugin developer adds many details how well the plugin is tested, reviewed, documented, maintained and so on. Based on this information, the best suited plugin will be chosen. For example, you now can use:

```
kdb mount /etc/security/limits.conf system/limits augeas \
        lens=Limits.lns logging
```

And the best suitable logger will automatically be chosen.

The configuration variable `/sw/kdb/current/plugins` now allows us to pass plugin configuration with the same syntax as the plugin specification passed on the commandline. A subtle difference is that thus the shell-splitting of arguments is missing, it is not possible to include whitespaces in the plugin configuration that way.

Now it is also possible to include the same plugin multiple times and also give them individual names. This feature is essential for script-based plugins, e.g. you now might add:

```
kdb mount file.x /example/mountpoint \
        lua#resolver script=resolver.lua \
        lua#storage script=storage.lua
```

Furthermore, `kdb mount` now supports recommendations, which can be enabled with `--with-recommends`. E.g. supplied to the mount command using augeas above, comments will automatically transformed to metadata to avoid cluttering of the real configuration.

## 410.6 Library Split

Up to now, Elektra consisted only of a single shared library, `libelektra.so`. Not all symbols in it were relevant to end users, for example, some were only needed by plugins. Others were only proposed and not yet part of the stable API. And finally, other symbols were not needed in some situations, e.g. the plugins do not need the `kdb` interface.

Thus, we split `libelektra.so`, so that only coherent parts are in the same library:

- `libelektra-core.so` only contains the `KeySet` data structure and module loading. Every binary using Elektra should link against it.
- `libelektra-kdb.so` contains the missing `KDB` symbols. Together with the `core` they contain everything declared in `kdb.h`. Michael Zehender plans to have multiple variants of `libelektra-kdb.so` that use different kinds of concurrency. Headerfile: <kdb.h>
- `libelektra-ease.so` adds functionality missing in `core` to make the life for C programmers easier. New headerfile: `kdbease.h`

- libelektra-proposal.so adds functionality proposed for core. It directly uses internal structures of core, thus they always need to have exactly the same version. Headerfile: kdbproposal.h

The source code is now better organized by the introduction of a libs folder. The explanation of every library can be found in /src/libs/.
The rationale of the library split is documented here. Shortly put, it was needed for further evolution and allowing us to grow and enhance without getting a fat core.
Thanks to Manuel Mausz for fixing many bugs related to the library split and also adapting all the bindings for it.

### 410.6.1 Benchmark

To show that the split does not make Elektra slower, Mihael Pranjić made a small benchmark. The real time of benchmarks/large and revision 634ad924109d3cf5d9f83c33aacfdd341b8de17a

1. kdb-static: Median :0.8800
2. kdb-full: Median :0.8700
3. kdb: Median :0.8700

So it seems that the split does not influence the time of running elektrified processes.
*Times were measured with time(1) and the median is calculated for 21 runs of benchmarks/large. This was done using scripts/benchmark_libsplit.sh*

## 410.7 Compatibility

As always, the ABI and API is fully forward-compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile every program without errors (API).
We added keyGetNamespace which allows us to handle all namespaces in a switch and to iterate all namespaces. This is essential, when a new namespace is added, thus then the compiler can warn you about every part where the new namespace is not yet considered. We currently do not plan to add a new namespace, but better be safe than sorry.
The internal function keyCompare now also detects any metadata change.
libtools was nearly rewritten. Even though it is mostly API compatible (you should not use the low-level Backend anymore but instead use the BackendBuilder), it is certainly not ABI compatible. If you have an undefined symbol: _ZN3kdb5tools7Backend9addPluginESsNS_6KeySetE you need to recompile your tool. Even the merging part has ABI incompatibility (different size of _ZTVN3kdb5tools7merging14NewKey↩StrategyE). Unfortunately, we still cannot guarantee compatibility in libtools, further changes are planned (e.g. implementing mounting of lazy plugins).
The python(2) and lua interfaces changed, an additional argument (the plugin configuration) is passed to open.
The INI plugin was rewritten, so many options changed in incompatible ways.
The default format plugin (e.g. for import/export) is no longer hard coded to be dump. Instead KDB_DEFAULT_↩STORAGE will be used. To get KDB_DEFAULT_STORAGE you can use the constants plugin:

```
kdb mount-info
kdb get system/info/elektra/constants/cmake/KDB_DEFAULT_STORAGE
```

Thanks to Manuel Mausz plugins do no longer export any method other than elektraPluginSymbol. It now will fail if you directly linked against plugins and did not correctly use their public interface. Please use the module loading and access functions via the contract.
The CMake and Pkgconfig Files now only link against elektra-core and elektra-kdb. If you used some symbols not present in kdb.h, your application might not work anymore.
libelektra.so is still present for compatibility reasons. It should not be used for new applications. Some unimportant parts, however, moved to the "sugar" libraries. E.g. the symbol elektraKeyCutNamePart is no longer part of libelektra.so.

### 410.7.1 Bootstrapping

When you use kdbOpen in Elektra as first step it reads mountpoint configuration from /elektra. This step is the only hard coded one, and all other places of the KDB's tree can be customized as desired via mounting.
The bootstrapping now changed, so that /elektra is not part of the default configuration default.ecf (or as configured with KDB_DB_FILE), but instead we use elektra.ecf (or as configured with KDB_DB_INIT).

This behavior solves the problem that `default.ecf` was read twice (and even differently, once for `/elektra` and once for `/`).
To be fully compatible with previous mountpoints, Elektra will still read mountpoints from `default.ecf` as long as `elektra.ecf` is not present.
To migrate the mountpoints to the new method, simply use:

```
kdb upgrade-bootstrap
```

which basically exports `system/elektra/mountpoints`, then does `kdb rm -r system/elektra/mountpoints` so that `default.ecf` will be without an mountpoint and thus the compatibility mode does not apply anymore. As last step it will import again what it exported before.
Details are here

## 410.8 Plugins

We already highlighted the new `spec` plugin, but also other plugins were improved at many places. Small other changes are:

- Conditionals now also support `assign/condition` syntax, thanks to Thomas Waser
- Lua and Python are not tagged experimental anymore. They now correctly add their configuration to the open-call.
- The plugin `yajl` (the json parser and generator) now also accepts the type `string` and yields better warnings on wrong types.
- Improved error message in the `type` plugin.

Larger changes were done in the following plugins:

### 410.8.1 INI

The INI plugin was rewritten and a huge effort was taken so that it fully-round-trips and additionally preserves all comments and ordering. Currently, it is brand new. It is planned that it will replace `dump` in the future.
Currently is has some minor problems when used as KDB_DEFAULT_STORAGE. You can avoid most problems by mounting a different file into root:

```
kdb mount root.ini /
```

Read here about the details.
A huge thanks to Thomas Waser.

### 410.8.2 Rename

Thanks to Thomas Waser `rename` is now fixed for cascading keys. Additionally, it does not change the `sync` status of the keys so that notification plugins work properly afterwards.

### 410.8.3 Crypto

The crypto plugin is a facility for securing sensitive Keys by symmetric encryption of the value. It acts as a filter plugin and it will only operate on Keys, which have the metakey „crypto/encrypt“ set.
The key derivation is still work-in-progress, so the plugin does not work with kdb yet. A planned method for key derivation is to utilize the gpg-agent.
For now the plugin requires the following Keys within the plugin configuration in order to work properly:

1. /crypto/key - the cryptographic key (binary 256 bit long)
2. /crypto/iv - the initialization vector (binary 128 bit long)

Please note that this method of key input is for testing purposes only and should never be used in a productive environment!
Thanks to Peter Nirschl, especially the patience for also supporting different platforms where dependencies need to be handled differently.

## 410.9 KDB

A technical preview of a new tool was added: `kdb editor` allows you to edit any part of Elektra's configuration with any editor and any syntax. It uses 3-way merging and other stable technology, but it currently does not provides a way to abort editing. So you only should use it with care.
The tool `kdb list` now searches in the rpath for libraries and thus will also find plugins not present at compile-time (using `glob`). Additionally, it sorts the plugins by `infos/status` score, which can also be printed with `-v`. The last plugins printed are the ones ranked highest.
When running as root, `kdb` will now use the `system` namespace when writing configuration to cascading key names.
Long paths are cumbersome to enter in the CLI. Thus one can define bookmarks now. Bookmarks are key names that start with `+`. They are only recognized by the `kdb` tool or tools that explicitly have support for it. Applications should not depend on the presence of a bookmark. For example, if you set the bookmark kdb:

```
kdb set user/sw/elektra/kdb/#0/current/bookmarks
kdb set user/sw/elektra/kdb/#0/current/bookmarks/kdb user/sw/elektra/kdb/#0/current
```

You are able to use:

```
kdb ls +kdb/bookmarks
kdb set +kdb/format ini
```

The kdb tool got much more robust when the initial configuration is broken, no man page viewer is present or Elektra was installed wrongly.
The `--help` usage is unified and improved.
The new key naming conventions are now used for configuration of the `kdb`-tool itself: `/sw/elektra/kdb/#0/%/` and `/sw/elektra/kdb/#0/current/` are additionally read. The option `-p/--profile` is now supported for every command, it allows to fetch configuration from `/sw/elektra/kdb/#0/<profile>/` instead of `current`. KDB is more robust when it cannot fetch its own configuration due to KDB errors.

## 410.10 Coding Guidelines

Thanks to Kurt Micheli the code guidelines are now properly documented. Thanks to René Schwaiger we also provide a style file for clang-format.
Furthermore we unified and fixed the source:

- only use @ for doxygen commands
- always use `elektra*` functions for allocation
- added a file header for every file

## 410.11 C++11 migration

Since we now only use C++11, we applied `clang-modernize` which simplified many loops and replaced many `0` to `nullptr`. Additionally, we added `override` and `default` at many places.
We removed all places where we had `ifdefs` to use `auto_ptr` if no modern smart pointer is available.
Because of these changes there is no easy way to compile Elektra without C++11 support, except by avoiding the C++ parts all together.
The C++ `KeySet` now also supports a `get` to retrieve whole containers at once. By specializing `KeySetType↩ Wrapper` you can support your own containers. Currently only `map<string, T>` is supported (T is any type supported by `Key::get`).
If you haven't removed it from your flags already, there is no use anymore to set `ENABLE_CXX11`.

## 410.12 Documentation

The documentation was improved vastly. Most thanks to Kurt Micheli who did a lot of editing and fixed many places throughout the documentation Also thanks to Michael Zehender who added two paragraphs in the main README.↩ md.
Key names of applications should be called `/sw/org/app/#0/current`, where `current` is the default profile (non given). `org` and `app` is supposed to not contain `/` and be completely lowercase. Key names are documented here. See also here. The main reason for long paths is the provided flexibility in the future (e.g. to use profiles

and have a compatible path for new major versions of configuration). By using bookmarks, users should not be confronted by it too often.

- many man pages were improved
- many typos were fixed, thanks to Pino Toscano!
- Fix documentation for kdb list, thanks to Christian Berrer
- Compilation variants are explained better, thanks to Peter Nirschl for pointing out what was missing
- document ronn as dependency, thanks to Michael Zehender
- fix broken links, thanks to Daniel Bugl

Thanks to Kurt Micheli, developers are now warned during compilation on broken links in Markdown. The output format is the same as for gcc. Additionally, the Markdown documentation of plugins now get a proper title in the pdf and html output of doxygen.

## 410.13 Qt-Gui 0.0.10

Raffael Pancheri again updated qt-gui with many nice improvements:

- update existing nodes in case the config was changed outside the gui
- safely update tree
- add update signal to metadata
- fix crash when closing the GUI
- move Actions in separate file to make main.qml less clustered
- plugins do not output messages at startup
- `BackendBuilder` is now used, which automatically takes cares of the correct ordering of plugins
- Qt 5.4 compatibility is still ensured
- C++11 is now used in Qt-Gui, too

## 410.14 Packaging and Build System

Elektra 0.8.14 now in Debian with qt-gui, man pages, thanks to Pino Toscano `read more here`
Thanks to Gustavo Alvarez for updating and splitting the packages on Arch Linux!
Thanks to `Harald Geyer`, Elektra is now packaged for OpenWRT. We fixed a number of cross-compilation issues and now officially support building against musl libc, with one minor limitation: RPATH works differently on musl so you need to install all plugins directly in /usr/lib/ or set LD_LIBRARY_PATH. Report any bugs in `Harald's OpenWRT packaging issue tracker`.

- export errors/symbols are now called `elektra-export-symbols` and `elektra-export-symbols` and can be installed using `INSTALL_BUILD_TOOLS` (by default off). This is needed for cross-compilation. Thanks to Harald Geyer for reporting.
- some header files are renamed because they endlessly included themselves if the header files were found in wrong order. Thanks to Harald Geyer for reporting.
- fixed compilation when built on more than 20 cores with >= -j15, thanks to Gustavo Alvarez for reporting and Manuel Mausz for analyzing
- lua 5.1 now works too (except for iterators), thanks to Harald Geyer for reporting. thanks to Manuel Mausz for adding a new FindLua.cmake
- pdf builds do not fail due to half written files, reported by René Schwaiger and fixed by Kurt Micheli

Read about `other packages here`.

## 410.15 Fixes and Improvements

- 3 way merge now properly deals with binary data, thanks to Felix Berlakovich
- getenv: fix wrapping on powerpc, thanks to Pino Toscano
- markdownlinkconverter: fix char/int mismatch, thanks to Pino Toscano
- wresolver: use KDB_MAX_PATH_LENGTH instead of PATH_MAX, thanks to Pino Toscano
- Cleaning up #ifdefs that break statements, thanks to Romero Malaquias
- Daniel Bugl tested the INI plugin
- cmake list_filter was broken because of different behavior in cmake_parse_arguments, thanks to Christian Berrer for reporting
- g++5.3 is now supported
- gtest does not link against pthread if not needed
- test cases that are built with BUILD_SHARED also successfully work
- kdb list works when libs are in same path as plugins, thanks to Harald Geyer for reporting
- fix macOS issues, thanks to Peter Nirschl, René Schwaiger and Mihael Pranjic
- fix resolver-baseflag documentation, thanks to Harald Geyer for reporting
- do not create wrong directories called （ and ） in source, thanks to René Schwaiger
- fix CMake for systems where iconv is not part of libc, thanks to Michael Zehender and Peter Kümmel (for FindIconv.cmake)
- fix segfault in libgetenv if root keys are present
- lua: fix Key:tostring(), thanks to Manuel Mausz
- add list of supported bindings, thanks to Manuel Mausz

## 410.16 Get It!

You can download the release from here and now also here on GitHub

- name: elektra-0.8.15.tar.gz
- size: 2338297
- md5sum: 33ec1e5982fb7fbd8893bf7b579b80f0
- sha1: 6b1fdd5aa5aaad6ba377b4bb5ef437e0c85319ff
- sha256: 6a406986cecb8d4a44485ced118ee803bc039b0824b72298e123b4dd47eb0b22
- sha512: 86a408dd546b33e3b437f92f415de7aee6a235189f9eab0762b3f44ab4c453ee369a53de10a9f5b0df1b446460b12c57c

This release tarball now is also available signed by me using gpg
already built API documentation can be found here

## 410.17 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the Mailing List the issue tracker on GitHub or by mail elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Markus

# Chapter 411

# 0.8.16 Release

- guid: 9c9247ee-ee9c-4f4a-a68e-76959def9b82
- author: Markus Raab
- pubDate: Fri, 29 Apr 2016 12:45:39 +0200
- shortDesc: adds stability improvements, configuration profiles & new plugins

In case you do not yet know about it, here is an abstract about Elektra:
Elektra serves as a universal and secure framework to access configuration parameters in a global, hierarchical key database. Elektra provides a mature, consistent and easily comprehensible API. Its modularity effectively avoids code duplication across applications and tools regarding configuration tasks. Elektra abstracts from cross-platform-related issues and allows applications to be aware of other applications' configurations, leveraging easy application integration.
Elektra consists of three parts:

1. *LibElektra* is a modular configuration access toolkit to construct and integrate applications into a global, hierarchical key database. The building blocks are:
    - language bindings (inclusive high-level interfaces)
    - GenElektra, the code generator for type-safe bindings
    - plugins for configuration access behavior and validation
2. *SpecElektra* is a configuration specification language that is easy to use and self-contained in the same key database (i.e. written in any of the configuration file formats Elektra supports).
3. Tools on top of LibElektra for administrators, such as CLI tools and GUIs.

See `https://libelektra.org`
The same text as follows is also available `here as html` and `here on GitHub`

## 411.1 Highlights

- Elektra now allows applications to support multiple profiles with a plugin, thus *without code modifications* in Elektra applications. That means a user can select multiple configuration files to use, even if the application has no explicit support for it. It completes the cascading feature (level `$HOME` before /etc), to allows us also to select different configuration for the same level.
- Resolver can now better handle conflicts that happen when files are removed and others that happen within a single time tick (resolution of your clock) and also better handles NFS and older file systems
- Default storage and resolver can be changed by symlink. So no need to recompile Elektra to change the default storage from INI to dump. INI now works quite reliable as default plugin and already used by default by its author Thomas Waser.

## 411.2 Other important features

- shell plugin allows you to execute shell commands on every KDB access and curlget plugin allows you to download configuration files from a URL during KDB access.
- Improvements in sync/merge of qt-gui with important fix (Usage of 0.8.15 qt-gui is discouraged)
- Add plugin for dpkg database (read-only)
- Assignment for conditionals using `assign/condition`.
- Support for multiple and nested statements
- Support for `condition/validsuffix` which allows you to suffix numbers with signs such as `%` or `$`. It does not check if the suffixes are identical.
- kdb mount now uses topological sorting to always find a dependency solution if there is one, many effort was put in that ordering is as requested, thanks to Thomas Waser for the topological sorting implementation
- Frontend generated by GenElektra now also can reload its values with taking the correct context into account.
- Source is now automatically formatted and formatting is checked on build server
- More flexible CMake syntax for PLUGINS

## 411.3 Plugins

Many new or vastly improved plugins are waiting to be explored.

### 411.3.1 curlget

The plugin curlget fetches a configuration file from a remote host before the configuration is being accessed:

```
kdb mount -R noresolver /tmp/curltest.ini system/curltest ini curlget
       url="https://raw.githubusercontent.com/ElektraInitiative/libelektra/master/src/plugins/ini/ini/plainini"
kdb ls system/curltest  # every get access will redownload the file
```

Thanks to Thomas Waser!

### 411.3.2 INI

The INI plugin is still under heavy development and was again nearly rewritten:

- fixed key is below hacks
- fixed ordering
- custom delimiter
- use meta array for comments
- rewritten ordering
- best effort order
- fixed array support

Thanks to Thomas Waser!

### 411.3.3 shell

This plugin allows you to executes shell commandos after kdbGet, kdbSet and kdbError (failing kdbSet):

```
kdb mount /tmp/test.ini system/shelltest ini array= shell 'execute/set=echo set » /tmp/log,execute/get=echo
       get » /tmp/log'
kdb set system/shelltest
cat /tmp/log
```

Thanks to Thomas Waser!

### 411.3.4 validation

The validation plugin is not new, but got many new features. It allows you to match values by a regex and set your own error messages in case a validation did not match.
Up to now, the regex was given as is to regcomp, which means that if the regex is contained *anywhere* in the value, the value is accepted.
Often this is not what we want, thus Thomas Waser added special support for icase, word and line validation. Additionally, flags allow you now to ignore the case or invert the match. This can be changed for every individual value or for the whole mountpoint.
Additionally, `kdb vset` validation was updated to use the new metadata and correctly match against the whole value.
Thanks to Thomas Waser!

### 411.3.5 hosts

Only minor improvements were necessary for the host plugin but it is quite matured already. The contract was changed so that ipv6 addresses for ipv4 addresses will be rejected:

```
# kdb mount --with-recommends /etc/hosts system/hosts hosts
# kdb set system/hosts/ipv4/localhost ::1
The command set failed...
Reason: localhost value: ::1 message: Address family not supported
# kdb set system/hosts/ipv6/localhost ::1
```

You can also comfortably and safely edit the hosts file with: `kdb editor system/hosts hosts`, then you have the functionality `visudo` for the hosts file.

### 411.3.6 rename

Again not a new plugin, but the plugin was greatly improved and many test cases were added.
Now you can set upper/lowercase individually for both sides:

1. What applications see.
2. What the configuration file contains.

For example, if you always want the keys in the configuration file upper case, but for your application lower case you would use:

```
kdb mount caseconversion.ini /rename ini rename get/case=tolower,set/case=toupper
kdb set user/rename/section/key valu
cat ~/.config/caseconversion.ini
#> [SECTION]
#> KEY = value
```

Thanks to Thomas Waser!

### 411.3.7 Resolver

Resolving by ~ as home directory now also for system and spec namespaces, thanks to Thomas Waser.
Files keep their previous owner, useful when root edits configuration files of others, thanks to Thomas Waser.
The resolver has many improvements to better detect conflicts.
The lock is now extended longer after the commit and already requested in the temporary file.
The warnings were improved when `getcwd` fails.
Resolver now can correctly handle conflicts with empty files. It can also better cope with frequent commits of the same binary. Elektra already reached some limits filesystems have.

## 411.4 Bindings

### 411.4.1 Java

Marvin Mall improved the Java binding, fixed the appending of keysets, added lots of documentation, and many unit tests.

### 411.4.2 C++

Some kind of misusage of vaargs is now detected at compile-time instead of crashing at runtime.

### 411.4.3 Generated C++

Value now supports convenience activations. Values can be used to activate context, no more layers are needed. Topological sorting makes sure that values are activated in the correct order, loops are not allowed anymore.
The `bool operator<` is now correctly inline (allows to use it in more than one translation unit)

## 411.5 Documentation

René Schwaiger<sanssecours> reworked most of the documentation and fixed countless spelling mistakes and other problems.

- Peter Nirschl updated the status of the crypto-plugin and fixed a typo
- Daniel Bugl wrote a cascading tutorial
- Daniel Bugl fixed all broken links
- René Schwaiger also drew a new logo with SVG. It is already used on GitHub as avatar for the organization.
- make all é use the same code point 233.

## 411.6 Testing

- Tests work if the build path contains spaces
- Tests: Fix problems locating memory checker
- remove obsolete TestScript.cmake

Thanks to René Schwaiger

## 411.7 Maintainer

By default now ALL plugins except EXPERIMENTAL are included. Plugins will be automatically excluded if dependencies are missing.
The PLUGINS syntax was vastly improved. Now many categories can be intermixed freely and also categories can be used for exclusion.
E.g. to include all plugins without deps, that provide storage (except yajl) and are maintained, but not include all plugins that are experimental, you would use:

```
-DPLUGINS="NODEP;STORAGE;-yajl;MAINTAINED;-EXPERIMENTAL"
```

Details see `/doc/COMPILE.md`.

### 411.7.1 Renamed files:

/usr/include/elektra/merging/kdbmerge.hpp -> /usr/include/elektra/merging/mergingkdb.hpp
/etc/profile.d/kdb -> /etc/profile.d/kdb.sh
(So that it works on arch linux, thanks to Gabriel Rauter)

### 411.7.2 removed files:

- /usr/lib/elektra/libelektra-crypto.so

was only necessary because of limitations of the build system and is now removed. It never had actual functionality, but was only a stub without a crypto provider selected.

### 411.7.3 new files:

- `/usr/include/kdbease.h`
- `/usr/lib/elektra4/libelektra-curlget.so*`
- `/usr/lib/elektra4/libelektra-dpkg.so*`
- `/usr/lib/elektra4/libelektra-profile.so*`
- `/usr/lib/elektra4/libelektra-resolver_fm_hpu_b.so`
- `/usr/lib/elektra4/libelektra-shell.so*`

more new files with ALL or EXPERIMENTAL:

- /usr/lib/elektra/libelektra-semlock.so

new tests all in folder /usr/lib/elektra/tool_exec: testcpp_contextual_update testkdb_conflict test_keyname testmod_curlget testmod_dpkg testmod_jni testmod_profile testmod_semlock testmod_shell testtool_mergingkdb
Following Plugins are excluded on specific platforms:

- mathcheck on Intel compiler (reason: failing test cases)
- simpleini on non-glibc systems (reason: not portable printf extension)

### 411.7.4 new symlinks:

- /usr/lib/elektra4/libelektra-storage.so
- /usr/lib/elektra4/libelektra-resolver.so

### 411.7.5 new releases

The first release of the libraries libelektratools-full, libelektratools and libelektragetenv. They now have SOVERSION 0.

## 411.8 Development

You do not need to format the source manually anymore. Make sure that you run scripts/reformat-source before creating a PR.
`clang-tidy` helps you to add blocks to have better maintainable code.
Felix Berlakovich improved the performance of the augeas plugin and also contributed a script to benchmark different host plugin. His thesis can be downloaded from `here`. It contains benchmarks and discussions about augeas.
The CMake function `add_plugin` was completely rewritten. Now you do not have to register your plugin at multiple points but instead information of README.md is parsed to correctly register the plugin to categories as stated by `infos/status` and `infos/provides`.
The code generator for errors also yields macros. This avoids usage of the IDs, which can be problematic if multiple pullrequests are prepared at once.

## 411.9 Compatibility

This might be the last release supporting wheezy, because it gets more and more time-intensive to find workarounds for the old compiler. The C++11 regex do not work at all.

### 411.9.1 Binary Compatibility Test

When you execute the test cases of 0.8.15 against Elektra 0.8.16 following test cases fail. None of them effect the API.

- test_splitget test_splitset .. Internal restructuring
- testmod_crypto .. not included by default now
- testmod_ini .. section handling changed, line 178: `nosectionkey contained no comment`
- testmod_rename .. internal API elektraKeyCreateNewName changed
- testmod_resolver .. internal data structure now contains more members to remember uid and gid
- testmod_template .. not present by default
- testtool_backend testtool_backendbuilder testtool_backendparser
- testtool_specreader .. changes in KDB tool before release
- check_kdb_internal_check .. experimental plugins are now excluded

### 411.9.2 Added API

in libease René Schwaiger added:

```
extern char const * elektraKeyGetRelativeName(Key const * cur, Key const * parentKey);
```

in libmeta Thomas Waser added (partly based on ideas/code from Felix Berlakovich):

```
extern void elektraMetaArrayAdd(Key *, char const *, char const *);
extern KeySet * elektraMetaArrayToKS(Key *, char const *);
extern char * elektraMetaArrayToString(Key *, char const *, char const *);
extern int elektraSortTopology(KeySet *, Key * *);
```

## 411.10 Tools

### 411.10.1 Qt-gui

Raffael Pancheri fixed an important issue which broke the synchronization because an key related to Elektra's internal version information was missing.
Felix Berlakovich updated the qt-gui so that it uses a newly written sync-method added in libtools.
Gabriel Rauter added a desktop file that uses the new svg logo from René Schwaiger.

## 411.11 Portability

- Peter Nirschl fixed code in the resolver that uses EBADMSG which was not available in BSD.
- Peter Nirschl improved detection of librt
- Felix Berlakovich fixed searching of FindSystemd
- MinGW64 resolver now handles conflicts correctly and does not ignore them anymore and now also is able to create empty files (but still not directories)

### 411.11.1 macOS

A lot of effort was invested to all test cases also run on macOS:

- .template syntax
- linking errors
- fix regex in conditionals plugins

Thanks to René Schwaiger

## 411.12 Bugs

- print null-environment correctly with `kdb getenv`
- keyIs(Direct)Below didn't work with cascading keys
- fix elektraKeyGetRelativeName (needed by ni) for cascading keys and move it to libease, thanks to René Schwaiger
- make nickel tests show correct test name, thanks to René Schwaiger
- glib: replace cursor_t with gssize so that GElektra-4.0.gir builds with gobject-introspection later than 1.47, thanks to Manuel Mausz
- fixed out-of-bounds bug in timeofday plugin
- elektraMetaArrayToKS correctly adds parent key, thanks to Thomas Waser
- kdb-shell: Do not abort ksOutput on binary data.
- some rework for global hooks, still not stable

## 411.13 Get It!

You can download the release from here and now also here on GitHub

- name: elektra-0.8.16.tar.gz
- size: 2405443
- md5sum: ef0c138b4a4fda017aa8bb6f812671ce
- sha1: c6a6f9c26addd5fcc274cea635de02ef680cfb1a
- sha256: 3cf0624eb027e533192ca9d612618df3d38ec3674c9cd20474f04ff269fad77e
- sha512: b225e61379907365a423ea75ec7138e5257bb78c526bb05a1ec21f66a52eb4bad9e6f1eb23209d700670b21b8616649

This release tarball now is also available signed by me using gpg
already built API documentation can be found here

## 411.14 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the Mailing List the issue tracker on GitHub or by mail elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Markus

# Chapter 412

# 0.8.17 Release

- guid: e6153a39-c4bd-41c3-bc86-785d451eb6c5
- author: Markus Raab
- pubDate: Tue, 14 Jun 2016 08:32:44 +0200
- shortDesc: adds several improvements & survey about configuration

## 412.1 Survey

First off: We created a survey questionnaire to gather more knowledge about the relevance of configuration systems. If you are involved in the development of free and open source software (FLOSS) you are the person we are looking for.
It would be a great help if you take this survey. It will be available till 18.07.2016 (anywhere on earth).
For every thoroughly and not anonymously finished survey € 40 cent will be donated to one of the following organizations of your choice:

- LimeSurvey (LimeService, kindly hosts this survey)
- SPI (General Donation: 0 A.D., LibreOffice, Debian, ArchLinux, . . . )
- FSFE
- GNOME
- KDE
- Mozilla (Firefox)
- Wikimedia Foundation (Wikipedia)

## 412.2 Why should I use Elektra?

The three main points relevant for most people are:

1. Even though Elektra provides a global keydatabase configuration files stay human read- and writable which allows us to integrate unmodified software.
2. Flexible adoption on how the configuration is accessed via plugins: you can run arbitrary code, e.g. do a `git commit` or log/notify when configuration files are changed.
3. Elektra allows you to specify configuration values:
    - use the value of other configuration values (symbolic links)
    - calculate the values based on other configuration values
    - validation configuration files

- generate code based on it
- and much more

Read more about Why using Elektra, which also contains since this release unique features, further reasons and limitations.
For a small demo see here

## 412.3 Highlights

- Qt-Gui reworked mounting and native icons
- Full macOS Support, Build Server improvements and new beginner friendly tasks
- allows us to mount csv, json and xml (and other common provider names) without needing to know plugin names
- colored output for kdb tools
- Experimental GSettings support

## 412.4 Beginner friendly tasks

In this release starting developing Elektra gets easier:

- ELEKTRA_DEBUG adds run-time checks and makes stack traces as if Elektra would not use plugins
- CMakeLists.txt for plugins got simplified, in most cases it should be not more than calling a single function, even if unit tests and test data are present
- We prepared beginner friendly tasks for you.

For details about ELEKTRA_DEBUG and CMake, see individual points below.

## 412.5 Find-Tools

There is now a fine collection of external scripts which can executed by kdb + <script>. The new script kdb find-tools provides full text search over the metadata as provided by the scripts.

- kdb find-tools -b BRIEF to search for a short text.
- kdb find-tools -a AUTHOR to search for an author.
- kdb find-tools -d DATE to search for a creation date.
- kdb find-tools -e EXECUTE to search for a type.

Developers should now add MetaData for their scripts..
Thanks to Kurt Micheli!

## 412.6 macOS Support

Because of its POSIX support one might think it would be trivial to support macOS. Unfortunately there were many small issues, especially in the regular expression handling and the filesystem.
Nevertheless we finally fully support macOS and the newly added travis build server makes sure it will stay this way.
A huge thanks to Manuel Mausz and Mihael Pranjić for fixing the issues and setting up travis:

- jni plugin now can load Elektra (avoids using .so) thanks to Mihael Pranjić
- initial creation of travis.yml thanks to Manuel Mausz
- Add all 3 different Xcode setups and some macOS fixes thanks to Mihael Pranjić

## 412.7 jenkins

Now (nearly) every build job can be triggered from Pull Requests. For example:

- jenkins build `libelektra` please

For a full list see `here`.
Thanks to Mihael Pranjić for the setup!

## 412.8 Fixes

- fix inconsistency with one excluded compilation variant, thanks to Harald Geyer for reporting #698
- fix dynamic searching of installed plugins, needed so that kdb list-tools works correctly thanks to Harald Geyer for reporting
- kdbtimer, `include <vector>` as needed by some compilers, a big thanks to Andreas Bombe for the non-maintainer upload in Debian to fix it for upcoming Debian release
- also find yajl header files if installed in non-standard include directories, thanks to Mihael Pranjić
- glib: make sure we use all definitions returned by pkg-config #719, fixes build on FreeBSD now glib bindings need CMake 2.8.12 thanks to Mihael Pranjić for reporting/testing and Manuel Mausz for fixing
- fix INI for macOS (did require some non-portable sorting properties of `qsort`.)
- INI makes INI-specific metadata private by prefixing `ini`.
- `kdb export` also works under MinGW, thanks to Gabriel Rauter

## 412.9 Rework Add Plugin

- prefer to link shared
- add plugin tests when using link shared
- make ADD_TEST simpler (without calling add_plugintest)
- make installation of test data simpler + honor INSTALL_TESTING option
- fix installation of test_data (do not install whole dir)
- introduce cache so that it is enough to pass parameters to `add_plugin*` once
- avoid PLUGIN_DIRECTORY_NAME and change CMAKE_CURRENT_SOURCE_DIR and CMAKE_↩ CURRENT_BINARY_DIR instead
- add_plugin: remove unused option SHARED_SOURCES
- implement a 3rd phase to add test cases: correctly handles dependencies of test cases to bindings
- fix testmod_jni

## 412.10 CMake

for maintainers:

- The CMake variables KDB_DB_SYSTEM and KDB_DB_HOME are now STRING and not PATH.
- BUILD_FULL and BUILD_STATIC are now OFF by default
- building with BUILD_SHARED is now preferred (for all examples, test cases,...)
- ELEKTRA_DEBUG_BUILD and ELEKTRA_VERBOSE_BUILD is not used anymore.

- ENABLE_DEBUG was added: it does not add debug symbols but run-time assertions.
- More CMake variables are marked as advanced.

for developers:

- BUILD_STATIC and BUILD_FULL is now OFF by default (nearly) all unit tests now also work with BUILD_↩ SHARED
- to support shared unit tests, a third phase was added when adding plugins inconsistent adding (across phases) of plugins and unit tests is reported
- in `add_plugin` remove SHARED_SOURCES, and add `ADD_TEST` and `INSTALL_TEST_DATA`.

and fixes:

- adding plugin tests is now much simpler, simply use `ADD_TEST` in `add_plugin`.
- KDB_DB_SYSTEM and KDB_DB_HOME are now STRING and not PATH because of incorrect resolving of ~.
- lua bindings tests: make sure lua executable matches with the lua libraries version thanks to Mihael Pranjić
- lua bindings: do not use hard coded `lua` executable.
- Fix CMake configure when BUILD_DOCUMENTATION is set to OFF thanks to Kurt Micheli

See more about changes to plugin adding in CMake in the `plugin decision`.

## 412.11 Experimental GSettings support

As part of the ongoing work of the bachelor thesis `Integration of Elektra into the GNOME desktop environment` we now have experimental support for Elektra as a GSettings backend on Linux (We will look into getting macOS support on a later date). When installed, applications using GSettings default backend will write to Elektra below the `/sw` key. The GSettings bindings are intended as a preview version so please do not use them in a production system.
To build the GSettings backend you have to explicitly add the binding even if `ALL` is given. e.g. `-DBINDINGS=gsettings -DBINDINGS="ALL;gsettings"`
All needed core functionality of a GSettings backend is already implemented. This includes notification support if you have your `/sw` mounted with the dbus plugin.
Please report any bugs you encounter.
For further information regarding the status of the implementations please refer to the corresponding `README` and `ticket`.

## 412.12 Common Provider Names

Mounting now supports to mount commonly known names even if the name is not a plugin. If more than one plugin is available automatically the best one is selected. The selection process works by annotating different qualities of the plugins, see `infos/status` in the README.md of individual plugins.
E.g. to mount a file using a json plugin (called yajl because of the library's name it build upon)

```
kdb mount file.json json
```

## 412.13 New Cachefilter Plugin

stores filtered keys internally so that they do not get accidentally lost and can be written to the storage again without the user having to remember including them in the writeout
The longer term goal is to add such global plugins per default, so that the usage of the API is easier.
For now you can simply add it using:

```
kdb global-mount cachefilter
```

Thanks to Marvin Mall.

## 412.14 Qt GUI 0.0.12

The Qt GUI receives new features and a better gnome integration. Its version number was updated to 0.0.12 (beta). Major features:

- use native icons (Qt GUI xdg icon theme support rework) thanks to Gabriel Rauter
- update desktop entry org.libelektra.elektra-qt-editor.desktop with new symbolic icon of Elektra's logo so that qt-gui can nicely started from within Gnome thanks to Gabriel Rauter
- Add new layout elements to backend wizard and integrate new BackendBuilder functionality (See Common Provider Names) to qt-gui thanks to Raffael Pancheri

Bug fixes:

- Reset to defaults now reverts back to built-in defaults.
- Make clicks on search icon focus on search textfield.
- save settings when settings dialog is closed.
- fix crash of qt-gui when crypto plugin was enabled (and added /shutdown option to enable previous behavior) thanks to Peter Nirschl
- fix qt-gui fails to synchronize because of readonly plugins thanks to Raffael Pancheri
- Rename desktop file: correct reverse url from org.elektra to org.libelektra.
- Rename elektra-qt to elektra-qt-editor.
- Rename ChooseColorWindow: The ChooseColorWindows will be replaced by a AppearanceSettingsWindow, all references to ChooseColor, choose color have been replaced by AppearanceSettings or choose appearance.

Other improvements:

- Install `elektra-qt-editor` binary so both the desktop files TryExec works and people not starting the gui trough `kdb qt-gui` have a speaking name in their process list.
- Replace occurrences of `Elektra Editor` with `Elektra Qt Editor` so that we use the same name in all places apart from the tools binary.
- Introduce Appearance Settings Window: Appearance Settings Window contains both color settings as well as a switch to disable or enable the system icon theme. For this to work we had to introduce the setting in `guisettings`. We also added a private function in `guisettings` to get and set settings with a boolean value.
- Tree reload on Settings close: We now synch and refresh the tree view on closing of the settings window if any settings have been changed, so changes can be seen imediatly in the tree.
- Add qt5 svg module as dependency: the qt5 svg module is needed so we can display icon themes that provide svg as icon format.
- Add and install symbolic icon with the installation of the Elektra Qt Editor.

Thanks to Gabriel Rauter and Raffael Pancheri for the engagement in improving qt-gui.

## 412.15 Colored kdb tool

A big thanks to Gabriel Rauter for improving the user experience with the kdb tool. On errors and in `kdb info` it was often quite hard to find the relevant text.
Now important parts are highlighted by bold or colorful text. This helps to spot the important information immediately without sacrificing information that would be important for a detailed analysis.
Every tool now has the option `--color` and `-C` which is set to `auto` per default. By writing to:

```
kdb set user/sw/elektra/kdb/#0/color off
```

one can go back to previous behavior.

## 412.16 Documentation

- improve documentation about how to pop a key
- document how to avoid running test cases as root in TESTING.md.
- document guarantees of elektraPluginGetData, thanks to Marvin Mall
- doc mentions that -1 should be returned *always* when an error is set
- many more spelling mistakes were fixed and useless whitespace was removed, thanks to René Schwaiger
- describe preferences when plugins are included/excluded
- improvements in ksCopy, ksPop, kdbGet and kdbSet API description
- added WHY document
- updated plugin decision to include 3rd phase

## 412.17 ELEKTRA_DEBUG build

ENABLE_DEBUG now enables a debug build for Elektra. It has nothing to do with debug symbols, but:

- it enables assertions
- it enables undefinied behavior sanitizer for clang
- plugins will not be closed so that stack traces are more useful (using RTLD_NODELETE)

ENABLE_DEBUG is recommended for every developer, even if you are not modifying Elektra itself. The assertions will give you hints on API misusage.
For example, keyNew was known to be error-prone. ENABLE_DEBUG now will report wrong parameters by an assertion.
The old options ELEKTRA_DEBUG and ELEKTRA_VERBOSE are not available anymore.
Thanks to:

- Thomas Waser for pointing to RTLD_NODELETE
- Gabriel Rauter for fixing qt-gui with -DENABLE_DEBUG=ON

The constants plugin was updated to provide cmake/ENABLE_LOGGER cmake/ENABLE_DEBUG and will no longer provide cmake/ELEKTRA_DEBUG_BUILD cmake/ELEKTRA_VERBOSE_BUILD

## 412.18 Other

- Gabriel Rauter is now listed in AUTHORS.md
- constants plugin: configure_file now uses current binary directory, not cluttering the main build directory.
- fix ssize_t for VS2015, thanks to Gabriel Rauter
- gtest: fix linking when using arch systemd-nspawn, thanks to Marvin Mall
- LD_LIBRARY_PATH is added to lua and python bindings needed for macOS, thanks to Mihael Pranjić
- Fix external unit test for Ubuntu 15.04 by putting files before the flags, thanks to Marvin Mall
- symbols in Ni_ namespace are now in elektraNi_
- add more ipv4 and ipv6 test cases for IP adress validation checker
- crypto-plugin avoid usage of hard coded error numbers, thanks to Peter Nirschl
- do not use number for resolver position

- to fix a compiler warning in macOS, we made the printf format specifier of time_t more portable, thanks to René Schwaiger
- many preparations for global plugins and mmap
- in the constants plugin `cmake/BUILTIN_PLUGIN_FOLDER`, `BUILTIN_DATA_FOLDER` and `BUILTIN_EXEC_FOLDER` were added.
- doxygen is only run once during build, thanks to René Schwaiger
- add script configure-home to build Elektra that it will resolve all pathes to home-directories
- add script metaini-to-c that converts METADATA.ini to C code, thanks to Thomas Waser
- add note that default values must be present for code generation, thanks to Martin Schleiss
- avoid `seq` as it is not available in some `*BSD`, thanks to Mihael Pranjić
- make jni testmod check consistent to others

## 412.19 Compatibility

As always, the ABI and API is fully forward- and backward-compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile every program without errors (API). This time you can even compile programs against 0.8.17 and run with 0.8.16.
For the qt-gui the svg module is added as dependency.
New and missing files in the installation:

- `elektra-qt-editor` is installed in the path (needed for TryExec in Desktop file)
- `libelektrasettings.so` will be installed if `gsettings` binding is enabled
- `libelektra-cachefilter.so` is the new cachefilter plugin
- `tool_exec/testmod_cachefilter` is its unit test
- `tool_exec/find-tools` is a new python script to find other tools
- `appdata/org.libelektra.elektra-qt-editor.appdata.xml`
- `icons/hicolor/scalable/apps/elektra-symbolic.svg`
- `share/man/man1/kdb-find-tools.1`

Renamed files:

- `applications/org.elektra.elektra-qt.desktop` got renamed to `applications/org.libelektra.elektra-qt-editor.desktop`.

Removed files:

- Some of the installed "test data" actually was source code from Elektra. Test data from the following plugins is affected: `hosts`, `ini`, `lineendings`,

Temporarily removed files:

- `testmod_lua`, `testmod_python` and `testmod_python2` do not work in a shared build and are temporarily disabled if `BUILD_SHARED` is enabled. Also their test data is affected.

## 412.20 Get It!

You can download the release from here and also here on GitHub

- name: elektra-0.8.17.tar.gz
- size: 2459542
- md5sum: e53efdb9a5e0852c58b21280b1e6c07d
- sha1: a1abcd4ac5aabfc60c34da98a02c4636e4634b5c
- sha256: a6a41afb0160feef84f7d1e0d199da26022ff8cb52ed455a0d306b589838d8f5

This release tarball now is also available signed by me using gpg
already built API documentation can be found here

## 412.21 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the Mailing List the issue tracker on GitHub or by email elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Markus

# Chapter 413

# 0.8.18 Release

- guid: 190576e0-9fef-486e-b8da-c4e75be08329
- author: Markus Raab
- pubDate: Fri, 16 Sep 2016 23:31:27 +0200
- shortDesc: adds intercept open, crypto plugins & improved git-resolver

## 413.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration parameters in a global, hierarchical key database. For a small demo see here:

## 413.2 Highlights

- Intercept open syscalls which allows Elektra to dynamically generate config files from Elektra's database
- Experimental version of cryptographic plugins
- A new zsh completion file (next to the bash completion file)
- Gitresolver allows to directly read and write config files from Git instead of files present in the file system.
- Survey completed successfully (and debts paid), we are now preparing raw data.

### 413.2.1 Crypto Plugin

Gpg is now used to decrypt a master password, which is used by the individual crypto backends. So all necessary parts for encryption of decryption of individual keys is present.
Furthermore, a new `botan` backend was implemented.
`See here`
Thanks to Peter Nirschl.

### 413.2.2 Open Interception

When Elektra directly modifies config files which are on the disc, and applications read the config files without Elektra, Elektra has no control over the access, e.g. we cannot dynamically calculate values. To avoid this, we wrote a library that intercepts the `open`-call.
Together with the `mozprefs` plugin, we got control over the configuration of Firefox and can dynamically change config values with all possibilities Elektra provides.
For easy setup, we implemented the script `configure-firefox`.
`See here`
Thanks to Thomas Waser.

### 413.2.3 Resolver

Resolvers in Elektra are the code that are responsible to determine where content should be read from and stored to. They are independent of the actual configuration file syntax.
The `gitresolver` allows you to get/store config data in git.
The `blockresolver` allows Elektra to take control of parts of the configuration file. This is useful for config files such as vim or zsh, which contain program code. The plugin allows you to split config files with special markers into parts containing code and others controlled by Elektra.

### 413.2.4 zsh completion

Added zsh completion file, and a script (`kdb install-sh-completion`) that installs bash+zsh completion when the default installation places do not work (e.g. for macOS).
Thanks to Sebastian Bachmann.

## 413.3 Documentation

- fix `kdb-import` man page, thanks to Kurt Micheli
- mark `keyIsSystem`/`keyIsUser` as internal
- fix doxygen reference to example
- better document that `global-mount` or `gmount` will overwrite previously mounted global plugins
- fix spelling mistake, thanks to René Schwaiger
- Wrote tutorial how to use Elektra-python bindings, thanks to Ulrike Schäfer

## 413.4 Quality

- shell recorder test cases now run during `make test`, thanks to Kurt Micheli and René Schwaiger (Warning: might remove present keys when it conflicts with their mountpoints)
- find-tools now is pep and pyflakes happy, thanks to Kurt Micheli
- fix bashism, thanks to Thomas Waser and Kurt Micheli
- better error message for conditionals plugin, thanks to Thomas Waser
- better error message for augeas plugin, thanks to Felix Berlakovich
- Many compilation warnings fixed, thanks to Gabriel Rauter, Thomas Waser
- GSettings: fix double free, thanks to Gabriel Rauter
- Fix external links and implement an external link checker, thanks to Kurt Micheli
- Fix openwrt/musl warnings with wrong printf format, thanks to Thomas Waser
- Fix NODEP metadata, allows you to build all plugins that do not have dependencies.

## 413.5 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).

### 413.5.1 Libtools

Libtools got a new major version (SOVERSION 0 -> 1):

- backend/plugin configs are now validated by plugins (needed by gpg plugin, which checks if wrong key IDs are supplied during mount)
- resolveRecommends was never implemented and was now removed

### 413.5.2 Plugins

The plugins conditionals and mathcheck are incompatible in some cases because of changes in syntax.

### 413.5.3 Proposal

New API: `keyRel2` which differs from `keyRel` by allowing you to specify which relation should be checked.

## 413.6 Development

- GitHub descriptions+workflow (displayed by github when creating PRs and issues)
- new trigger phases for GitHub, see `doc/Git` thanks to Mihael Pranjić
- valgrind suppressions are great again, thanks to Peter Nirschl
- Plugins get a new namespace `internal` which can be used for metadata that is not relevant for other plugins.
- kdberrors.h is only generated once, which allows us to use other build systems, thanks to René Schwaiger
- `INCLUDE_SYSTEM_DIRECTORIES` in add_plugin allows you to add an include path where warnings are suppressed (useful for boost).
- `infos/provides` now allows multiple entries

## 413.7 Packaging

- Plugin-provider `CRYPTO` can be used to enable/disable all crypto plugin variants (not enabled by default because its experimental).
- Config option `ENABLE_OPTIMIZATIONS`, enable by default: trade more memory for speed (can be turned off on embedded systems)
- `INSTALL_SYSTEM_FILES` is now off by default on macOS.
- bash-completion is installed to where pkg-config tells us, thanks to Gabriel Rauter (fallback is now `/usr/share/bash-completion/completions/kdb`) was `/etc/bash_completion.d/kdb` (removed)
- zsh is now installed to `/usr/share/zsh/vendor-completions/_kdb` (except for Darwin, where `/usr/local/share/zsh/site-functions` is used)
- removed `/etc/profile.d/kdb.sh`: the script `elektraenv.sh` was removed (and is no longer installed), superseded by `elektrify-getenv`
- added scripts install-sh-completion configure-firefox elektrify-open
- added plugins libelektra-blockresolver.so libelektra-boolean.so libelektra-crypto_botan.so libelektra-crypto_↩ openssl.so libelektra-desktop.so libelektra-mozprefs.so libelektra-passwd.so
- added tests testmod_blockresolver testmod_boolean testmod_crypto_botan testmod_crypto gcrypt testmod↩ _crypto_openssl testmod_mozprefs testmod_passwd test_opmphm_vheap test_opmphm_vstack
- added test data blockresolver mozprefs passwd

## 413.8 Other

- Conditionals and mathcheck plugins got support to specify relative keys, thanks to Thomas Waser
- `kdb` command-list: commands are written in bold
- GSettings backend can be build standalone, thanks to Gabriel Rauter
- first data structures for order preserving minimal perfect hash map, thanks to Kurt Micheli
- added a new passwd plugin, thanks to Thomas Waser
- boolean plugin to normalize boolean values, thanks to Thomas Waser
- `desktop` plugin to detect which desktop currently is running (supports kde, gnome, tde, unity or any other XDG conformant desktop)
- `doc/paper` contains some info for `joss`

## 413.9 Get It!

You can download the release from `here` and also `here on GitHub`

- name: elektra-0.8.18.tar.gz
- size: 2582183
- md5sum: 62fe0fbf9ee57ffaa58a982f602f596a
- sha1: 743484e16b102a00cd58956a49f0c558939d56a8
- sha256: 9ee65895ba5cba6736c13c264637664c1410b25f4aaaeac8f1f83712ff93d53b

This release tarball now is also available `signed by me using gpg`
already built API documentation can be found `here`

## 413.10 Stay tuned!

Subscribe to the `RSS feed` to always get the release notifications.
For any questions and comments, please contact the `Mailing List` the issue tracker `on GitHub` or by email `elektra@markus-raab.org`.
`Permalink to this NEWS entry`
For more information, see `https://libelektra.org`
Best regards, Markus

# Chapter 414

# 0.8.19 Release

- guid: 8e05231a-4f3d-488b-8dc2-5f0d5c474c39
- author: Markus Raab
- pubDate: Tue, 22 Nov 2016 22:04:59 +0100
- shortDesc: adds more tutorials, ruby bindings & cleanup of core

## 414.1 What is Elektra?

Elektra solves a non-trivial issue: how to abstract configuration in a way that software can be integrated and reconfiguration can be automated. Elektra solves this problem in a holistic way. Read `why Elektra` for an explanation of why such a solution is necessary. It can be seen as a `virtual file system` for configuration files.

## 414.2 Highlights

- more tutorials and getting started guides
- new Ruby bindings
- cleanup of core (only 124K for main library on Debian/amd64)

### 414.2.1 More Tutorials

Elektra already has an open and welcoming environment, with many interesting discussions. It is our interest that we keep it that way. To make this a bit more formal we added a `code of conduct`.
But without good introductions, it is easy to get lost in such a large initiative like Elektra. Thus we focused on writing great tutorials for this release!

- We wrote an `overview readme`
- We wrote new tutorials about `mounting` and `validation` (thanks to Christoph Weber)
- We wrote a readme to shell recorder transpiler which allows us to execute tutorials and verify that the examples in them work. (thanks to Thomas Waser)
- `Lua` and `Python` plugins got tutorials and better explanations! (Thanks to Marvin Mall)
- The `doxygen` documentation now also uses links to directories, thanks to Kurt Micheli!

Thanks to Armin Wurzinger for pointing to areas of improvement. A big thanks to Marvin Mall, Kurt Micheli, Christoph Weber and Thomas Waser!
If you like the tutorials, we would love to read from you. Please feel free to `start a discussion or ask a question`. We also added a `FAQ` and updated `CONTRIBUTING`

### 414.2.2 Ruby Bindings

We now provide Ruby bindings for Elektra. The bindings are based on the C++ bindings and are generated by SWIG. A strong focus was put on a good integration with standard Ruby features and conventions, such as naming conventions, predicates, key and metadata iteration...
A `short introduction` shows some basic usage scenarios. More detailed examples can be found in the `examples directory`.
A big thanks to Bernhard Denner!

### 414.2.3 Cleanup of Core

Following methods were hidden (`static`) or removed:

- `mount*` methods
- `trie*` methods
- `backend*`
- `split*`
- `keyGetParentNameSize`
- `keyGetParentName`

These are dozens of methods and it was required to adapt the unit tests to work with the hidden methods.
A big thanks to Kurt Micheli!

## 414.3 Usability

- Improved many error messages
  - spelling
  - be more friendly to the user
  - capitalization
  - mention `sudo !!`
- `kdb set`: do not print what was not done
- `kdb editor` handles non-modified files (will not do anything)
- Be more chatty about what `kdb` does, can be disabled with `-q` or `/sw/elektra/kdb/#0/current/quiet`.
- Furthermore, `-v` now tells even more details (e.g. `kdb-import` outputs the key about to import)

## 414.4 Plugins

### 414.4.1 New

- `c plugin` generates C code that represents configuration. This is useful for unit tests or if you need to have hard coded fallback configuration in your C application.
- `base64 plugin` allows you to encode binary data. This is especially handy in combination with the `crypto plugin` to avoid problems with non-printable characters in configuration files. (Thanks to Peter Nirschl)
- `fcrypt plugin` allows you to fully encrypt configuration files. They are only decrypted when applications access them. (Thanks to Peter Nirschl)
- required plugin rejects every key that is not required by an application.
- simple spec lang allows you to define metadata for enum and required in a more compact way.

### 414.4.2 Major Enhancements

- simpleini got a configurable format in which it will read and write configuration files. For example, one can use format=% -> % to have key -> value.
- enum got support for multi-enums, i.e., multiple separated values within one value. The error reporting was improved, too. (Thanks to Thomas Waser)
- glob accepts a list of named flags instead of an integer value and aborts matching after first hit. (Thanks to Felix Berlakovich)
- hosts now only accepts ipv4 and ipv6 keys. (Thanks to Felix Berlakovich)

## 414.5 Development

In the perpetual effort to improve software quality, we made several improvements: (This information is mainly intended for Elektra's developers.)

- A new logger encourages developers to write more comments (ELEKTRA_LOG)
- ELEKTRA_ASSERT prints better messages on failure and does not need && trick.
- get rid of previous VERBOSE macro at many places.
- Many assertions were added in the low-level helpers (memory management)
- Using the assertions we fixed some undefined behavior. (Thanks to Thomas Waser)
- added new configure-debian-debug and configure-debian-log helper scripts
- The build server now checks if builds with active logger and debugging work correctly.
- Improved Coding Style in crypto_botan (thanks to Peter Nirschl)
- add external-links.txt to outputs (The file is generated in the build directory and contains all external-links. To validate them, use ./scripts/link-checker) (Thanks to Kurt Micheli)
- markdownlinkconverter handles directories correctly (using stat). (Thanks to Kurt Micheli)
- Fixed compiler warning caused by libxml2 (different behavior since 2.9.4), thanks to René Schwaiger
- added often used links in main README
- Improve documentation about failing test cases and what to do about it.
- added decisions about plugin_variants and array. (Thanks to Marvin Mall)
- Rename to metadata, metakey, mountpoint (Thanks to Peter Nirschl)
- std::ios_base::showbase can be used to output metadata when streaming keys (C++)
- New infos/status: readonly, writeonly, limited (Thanks to Marvin Mall)
- The tool update-infos-status orders infos/status and allows devs to easily add/rem entries. (Thanks to Kurt Micheli)
- Automatic setting of infos/status: nodoc, nodep, unittest, memleak, configurable (Thanks to Kurt Micheli)
- Improve create_lib_symlink, add PLUGIN argument and make it useful also for other library symlinks.
- New Markdown style applied to most Markdown files. (Thanks to Marvin Mall)
- Tracer is now disabled, even for ENABLE_DEBUG. (Thanks to Marvin Mall)
- Updated SECURITY document
- Macro naming convention ELEKTRA_, added kdbmacros.h
- ENABLE_DEBUG also works with clang and ENABLE_ASAN now allows devs to additionally enable sanitizers. Thanks to Gabriel Rauter.

## 414.6 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
It is now possible to enquiry which plugins provide a specific format. This needed changes in libtools, which got a new major revision. Changes in the plugin's contract are fully compatible: You can now use `storage/ini` instead of `storage ini` in `infos/provides` which gives you the information that `ini` is a storage format (and not anything else the plugin might provide). For compatibility reasons, the build system still adds `storage ini` even if only `storage/ini` is specified.
That means that `kdb mount file.json /examples/json json` still will find `json` plugins even if they are not called `json` but `yajl`.
Another breaking change in `libtools` is that `appendNamespace` was renamed to `prependNamespace`.
Error messages changed a bit, so if you tried to parse them, make sure to make the `e` of error case-insensitive (`[eE]`).
In the C++ binding, `rewindMeta` is now `const` and some methods to check if a key is in a namespace were added.
The intercept libraries were moved to a `common folder`. They can now be included or excluded like other `BINDINGS`. For consistency reasons the libraries were also renamed (`libelektraintercept-fs.so` and `libelektraintercept-env.so.0`), but symlinks allow you to link against their old names (`lib/libelektraintercept.so` and `lib/libelektragetenv.so.0`).

## 414.7 Package Maintainers

This information is intended for package maintainers.

- GI Bindings were removed from `BINDINGS=ALL`. It is recommended to use `SWIG` bindings instead, which will be added with `ALL`.
- Intercept libraries are part of `BINDINGS`. They will be added on glibc systems where `BINDINGS=ALL` is used.
- Documentation in textfiles is now installed, `TARGET_DOCUMENTATION_TEXT_FOLDER` was added for that purpose. The files are:
  - `BIGPICTURE.md`, `GOALS.md`, `LICENSE.md`, `METADATA.ini`, `SECURITY.md`, `AUTHORS`, `CONTRACT.ini`, `NEWS.md`, and `WHY.md`

Other new files are:

- Plugins: `libelektra-base64.so`, `libelektra-c.so`, `libelektra-fcrypt.so` `libelektra-required.so`, `libelektra-simplespeclang.so` (only in `EXPERIMENTAL`, not added by default, but with `ALL`)
- `site_ruby/_kdb.so` (ruby binding, only in `ALL`)
- `testcpp_keyio`, `testkdb_error`, `testmod_base64`, `testmod_fcrypt` (test binaries in `TARGET_TOOL_EXEC_FOLDER`)

Changed files are:

- `libelektraintercept-env.so` (renamed from `libelektragetenv.so.`, but still available as symlink)
- `libelektraintercept-fs.so` (renamed from `libelektraintercept.so`, but still available as symlink)
- version upgrade: `libelektratools.so.2`

## 414.8 Portability

Elektra should work on every system that has CMake and a C/C++ compiler.
For this release we increased portability to better work with macOS, CentOS 7, and OpenSuse 42.

- macOS:
  - Travis build server now also build qt-gui
  - Support for xcode8 added (xcode6 still supported)
- fix lua != 5.2 issues (wrong output), update docu
- remove hard dependency to `pkg-config`
- remove hard dependency to version 3 of `cmake` (most parts still work with version 2)
- make search for swig 2 visible
- fix plugin names and mounting on OpenSuse 42.1

A big thanks to Kai-Uwe Behrmann, Mihael Pranjić and Sebastian Bachmann.

## 414.9 Fixed Issues

- simpleini: use correct error number when open file fails
- yajl: improve error message on non-utf8 text. (Thanks to Christoph Weber)
- drop multiple / from ~ paths (Thanks to Thomas Waser)
- fix failing test cases with `ENABLE_DEBUG` #988 (Thanks to Thomas Waser)
- csvstorage: files in source are rewritten #987 (Thanks to Thomas Waser)
- fix RTLD_NODELETE for OpenBSD (Thanks to Thomas Waser)
- better handle adding/deleting of read-only (info) plugins.
- fix behavior of multiple plugins setting errors (first error wins, later errors are transformed to warnings) (Thanks to Thomas Waser)
- fix resolver logic for missing files
- regex string in conditionals (Thanks to Thomas Waser)
- use `KDB` environment variable in shell tests and fix counting of tests for `kdb run_all`.
- output to `stderr` for `elektrify-*` scripts
- make `desktop plugin` mountable
- avoid CMake warnings in `make uninstall` (avoid `@`)
- fix quoting in ini plugin (Thanks to Thomas Waser)
- fix plugin names and mounting with plugin pre/postfixes (Thanks to Kai-Uwe Behrmann)
- mount-openicc: rename to openicc.json (Thanks to Kai-Uwe Behrmann)

## 414.10 Get It!

You can download the release from here and also here on GitHub

- name: elektra-0.8.19.tar.gz
- size: 2681639
- md5sum: 6669e765c834e259fb7570f126b85d7e
- sha1: 82cefe4cea58d6e6b0a99ddbda24d1b57e98d93a
- sha256: cc14f09539aa95623e884f28e8be7bd67c37550d25e08288108a54fd294fd2a8

This release tarball now is also available signed by me using gpg
already built API documentation can be found here

## 414.11 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the Mailing List the issue tracker on GitHub or by email elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Markus

# Chapter 415

# Website Release

- guid: 102b84a3-c41e-485c-8fe2-f12a24b3fbfd
- author: Marvin Mall
- pubDate: Thu, 22 Dec 2016 17:46:19 +0100
- shortDesc: introduces new Elektra website with snippet sharing

## 415.1 Highlight

1. Release of new Elektra website with an integrated service for sharing of configuration snippets.
2. The website also supports conversion between different configuration formats.
3. Website structures documentation and news sections in a new way.

## 415.2 Introduction

With Elektra developing into a more and more reliable as well as popular system to manage system configurations, the demand for a better public appearance increases as well. For this reason, we are happy to be able to announce the release of our new `website`!
The new website does not only give us a chance to better present ourselves to the open world, it also enables us to structure our project documentation better. We hope that this will make it easier for our users to get started with Elektra and all of its awesome features!
Besides the documentation, the website does also include a database that can be used to share, search, download and convert configuration snippets in various formats. We hope that this tool helps developers and administrators, but also anyone else to simplify their configuration processes when they have to look for a specific configuration snippet. Btw. with snippet we mean that you can also share parts of configuration files that you find particular useful! But sharing of snippets does not only help other users, it can help yourself as well because you can search for them easier. You also have access to the snippets in various formats at any time, allowing you to use them across multiple system by mounting them with the `curlget` resolver!

## 415.3 The Website

The website was written by Marvin Mall in the course of his `bachelor thesis` as part of the frontend he developed for his snippet sharing service. His main goals were to create a proper appearance for Elektra, but also to create a platform that promotes his service. We think that this worked out quite well by connecting the website with the service the way it was done.

### 415.3.1 Documentation

An important aspect of the new website was to make existing documentation more transparent and structured. A lot of documentation files have been changed to achieve this goal and an equal amount of effort was put into writing

a system that decouples the documentation structure on the website from the structure used within the Elektra repository.
The tutorials section was partially reworked to make the first steps together with Elektra easier for our users. Clearly the effort put into the tutorials is worth it. Thanks to Erik Schnetter for the valuable feedback where improvements are needed and Christoph Weber for (re)writing the tutorials!
We should note – as always in software – that the structure on the website is not final yet and will definitely develop over time, especially the bindings and libraries sections will get some more attention.
If you are interested in the techniques we use to structure our files, you can have a look at the website readme. The website is already the fourth view of our Markdown pages! The others are man pages, doxygen, and github.

### 415.3.2 Homepage & News

Besides the documentation we also wanted a place to properly present ourselves and our news around Elektra. For this reason we created a new home page which shall give an overview of what Elektra is and can do. Additionally to that, we also added a news section to keep you better up-to-date!
We hope that you enjoy our new appearance as much as we do!

### 415.3.3 Snippet Sharing

Another important part of the website and also without doubt the part that took most effort to create, is the service that allows for sharing of configuration snippets. It is run by a REST service fully built with the help of `CppCMS` on basis of Elektra as data store. All data concerning snippets and user accounts is stored in Elektra's key database (of course with password being properly hashed).
The service allows you to paste configuration snippets in various (supported) formats and to tag, describe and name them. This in return allows you to search snippets by keywords and to download them – even in other formats than the format used for uploading.
Clearly the service is meant to be driven by its users. Therefore we ask you to share your own configuration snippets, maybe they can be of help, e.g., be a time saver for someone else!
Snippets shared with the service are `BSD licensed`. The snippets can also be downloaded directly as bundle from a separate `GitHub repository`. As soon as a snippet is added, changed or deleted on the website, a job that updates the repository is triggered. So you can expect the repository to be always up-to-date.

### 415.3.4 NoScript

The website is fully written with the help of AngularJS and is therefore heavily based on JavaScript. This should be no issue though as the website does only use resources that can be found in the official Elektra repository:

1. So in case you cannot or do not want to use JavaScript, you can find all resources also `here`.
2. If you are only worried about executed untrusted JavaScript, you can study and improve the Web Frontend, which builds the website. Based on this, we hope you disable `NoScript` for our page so that you are able to share snippets!

## 415.4 Domains

All Elektra Domains directly hosted by us are now only served by `https`. The former `http` sites are only redirects to `https`. This might cause trouble with some software, e.g., update `/etc/apt/sources.list`:

```
deb     [trusted=yes] https://build.libelektra.org/debian/ wheezy main
deb-src [trusted=yes] https://build.libelektra.org/debian/ wheezy main
```

The build Server is no longer reachable at port 8080, but now only directly at `https://build.libelektra.org/`.
While most `libelektra.org` now point to the new website, you can still directly go to `GitHub` and also to the `bug tracker`.
The old Wordpress installation was shut down because of security concerns.

## 415.5 Feedback

At this point there is not much more to say about the new website except for: Feel free to explore it!

We greatly appreciate all feedback, be it for the website, the snippet sharing service or other parts of the Elektra Initiative. We always have an open ear for suggestions and we also like to help with technical issues, simply leave us a note on GitHub!

## 415.6 Stay tuned!

Subscribe to the reimplemented RSS feed to always get the release notifications.
For any questions and comments, please contact the mailing list, use the issue tracker on GitHub or write an email to elektra@markus-raab.org. For issues or feedback concerning the website, you can also contact us at website@libelektra.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Marvin & Markus

# Chapter 416

# Elektrify LCDproc

- guid: d52657b5-60da-4a21-9679-f8aacf6d6b72
- author: Markus Raab
- pubDate: Sat, 18 Mar 2017 20:43:03 +0100
- shortDesc: Elektrify LCDproc

  LCDproc is a piece of open source software that displays real-time system information from your `Linux/*BSD` box on a LCD.

LCDproc's `website` is a bit outdated but LCDproc itself is now well-maintained on `GitHub` and had a `release recently`.
Like in many projects, it invented its own configuration access and INI parser which did not evolve with the needs of the project. As a consequence inconsistencies and code duplication spread over the source. For example, the LCDproc configuration access does not support values that represent display's size (such as `20x4`). Thus every LCDproc's module has its own parsing code for such values.
Some days ago (16.03.2017), we met with Harald Geyer and discussed the current situation. We decided that we will elektrify LCDproc and remove all the configuration access and parsing code within LCDproc. To **elektrify** an application means to change the application so that it uses LibElektra afterwards.

## 416.1 Goals

We formulated three goals:

1. We (Elektra Initiative, mainly Thomas Waser) remove as much code as possible from LCDproc's code base.
2. Users of LCDproc should be able to use `LCDd.conf` as they use it now.
3. We avoid the current duplications of configuration specifications. (Currently in `LCDd.conf`, the docbook, in code checking the limits, and the default parameter in the code.)

Nice-to-have is:

- Safe updates: `make install` should not break the current `LCDd.conf`. (As it is already done in Debian.)
- The robustness of LCDproc on misconfiguration should be improved (Thomas Waser writes the specification).
- Automatic reloading of the daemon when using Elektra to update configuration. (Manual `SIGHUP`-signals will work in any case.)
- Have physical units with metric prefix like `500ms` for `0.5s` (seconds).

Possible limitations are:

- We break support for systems that only have very old compilers.

- The `-c` option to specify a different configuration file is against the abstraction Elektra should deliver. We might create a `wrapper script` that emulates `-c` via mountpoints.
- LCDproc will depend on the yet-to-be-released `0.8.20`. We will delay the 0.8.20 release until all parts for LCDproc are tested and ready.

In any case, the `advertised benefits of Elektra` will automatically apply (incomplete list):

- Global key database: you can connect other configuration files with the LCDproc's configuration and validation.
- Allows users to easily modify the specifications, for example to have different command-line options, or support for environment variables.
- `Profile support`: Having multiple complete configuration settings you can easily choose from. (thanks to Thomas Waser)
- Introspectibility: you can check with `kdb`-tool which configuration settings LCDproc will receive.
- Other configuration file formats can be used instead, e.g. JSON or XML. (Only if wanted as personal preference, by default INI will be used to remain compatibility with `LCDd.conf`.)
- Easy migration paths to use other configuration file formats such as YAML as default in future. (thanks to René Schwaiger)
- Elektra's tool can be used to configure LCDconf, including:
  - `kdb set` to modify individual settings within scripts and validation.
  - `kdb editor`, which spawns your favorite editor but validates the configuration file before writing it out.
  - `kdb qt-gui`, the Qt GUI (thanks to Raffael Pancheri).
  - The web UI of Elektra (thanks to Daniel Bugl).
- Elektra's website can be used to share LCDproc's configuration files, and you can use the `curlget plugin` to mount files from the website. (thanks to Marvin Mall)
- You can use Shell, Python, or Lua to write small scripts that are executed on configuration access.
- Elektra allows you to directly read and write from git using the `Git plugin`. (Even if `LCDd.conf` is not checked out, thanks to Thomas Waser)
- We extensively `test` Elektra with modern techniques such as fuzzing.

## 416.2 Validation

Instead of `if`s within the code, we will use SpecElektra for validation. `SpecElektra` is a configuration specification language that allows us to describe which configuration is valid.
This specification will be installed as part of LCDproc to `/usr/share/elektra/specifications`. It uses the `metadata` as defined by the plugins to validate configuration changed via Elektra.
For broken installations or when executing LCDproc from the source repository, there is also a built-in specification. The specification will only be used if the installed one cannot be found.
Thus the configuration validation specification is a normal configuration file also integrated in Elektra, also the specification can be introspected: useful for system administrators who want to know about valid entries, but also for tools like our newly developed web-UI by Daniel Bugl.
The web-UI automatically restricts the interface so that only valid entries can be entered. For example, if you should enter a boolean, only a check box is presented to you.

- For more information about validation, see the `tutorial`

## 416.3 Code Generation

We (mainly Dominik Hofer) are currently developing a high-level API, whose first user will be LCDproc. In this API, we will make sure during compilation, that configuration access is done correctly.
This is especially useful when configuration settings get renamed. Then all places where out-dated configuration settings are used will fail to compile.
In particular using code generation developers do not need to use strings to refer to configuration settings and they get easy-to-use enums consistent with the configuration specification.
Furthermore, code generation will make sure that a specification (and default configuration settings) will be found even if no `/etc` or no `/usr` is found.
We also found that we cannot use code generation everywhere. In generic access code, code generation obviously is limited.

- For more information about code generation, see the tutorial

## 416.4 Risks

Understandably, users might be concerned that such a change will not work or create problems in the future. Here we will discuss some of the concerns.

> Elektra might be discontinued.

From history perspective, Elektra received steady development since 2004. Elektra is a FLOSS project and welcomes everyone to join. In the last years several people did, with an increasing number per year. Currently following people are working on substantial new features in Elektra (sorted by first name):

- Armin Wurzinger: Quality Improvements
- Bernhard Denner: Puppet Module
- Daniel Bugl: WebUI
- Dominik Hofer: the high-level API
- Kurt Micheli: Order Preserving Minimal Hash Map
- Markus Raab: Maintainer
- Michael Zehender: Quality Improvements
- Mihael Pranjić: mmap plugin
- Peter Nirschl: crypto plugin
- René Schwaiger: YAML plugin
- Sebastian Bachmann: Shell Completion
- Thomas Waht: Notification
- Thomas Waser: Validation and Transformations of Configuration
- Vanessa Kos: Misconfiguration Bug Database

Obviously, there are many more casual contributors.
Elektra has a large set of automated tests and only a small amount of technical dept. Elektra has no required external dependencies except libc. So without internal changes, only minimal maintenance cost is required.

> Elektra is unfinished.

Technically this is true: we did not reach 1.0. We are, however, on track to reach this goal within this summer. Now is the best time to join because we can provide more support and are more flexible for changes and wishes.
Some time ago we asked in a survey in which direction Elektra should develop. Most open issues are (in)direct responses from this wanted direction.
Some plugins are experimental or proof-of-concept, but they are clearly marked as such.

It seems a bit to me like xkcd: Standards.

Elektra does not invent a new configuration file format nor new standards where to store configuration files but abstracts over these issues.

Elektra not being available in my distribution.

For the following Linux Distributions Elektra 0.8 packages are available:

- Gentoo
- Arch Linux
- Debian
- Ubuntu
- OpenSuse
- PLDLinux
- Linux Mint
- OpenWRT

See INSTALL for the complete and up-to-date list.

If Elektra does not take off and achieve world dominance, will we be worse off than before?

Making sure that projects will not be worse off is what we did the last years: Not only offer an API and wait for world dominance but to offer an implementation that can compete with any configuration library out there. We are not completely there yet (there are some details where specific other libraries are better than Elektra in specific points) but these are not points that the current configuration system of LCDproc supports (not even close). And the libraries that can compete with Elektra have a completely different level on which dependencies they have: Elektra is the only one only requiring libc.

Do we retain the old way of configuring things, i.e. manually editing an ini file in /etc?

Absolutely, you can think of libelektra as a small library in C that reads a configuration file and returns a data structure if you do not use any of its advanced features.

Do we retain the old way reloading/restarting the system service?

Elektra does not interfere with restarting. It is a passive library. It provides some techniques for reloading but they are optional (but we recommend that you keep the in-memory and persistent configuration in sync via notification). For more information, see the FAQ.

## 416.5 Win-Win

We can both profit from it:

1. For LCDproc it will be a simplification of code while getting many more tools.
2. For Elektra it will improve its adoption and packaging.

For oyranos it already worked well on both sides, see our discussions in the issue tracker, for example #1134.
If you also maintain an free or open source (FLOSS) project with an out-dated configuration system, please contact us.
Obviously, we cannot fully port every FLOSS project ourselves, instead we will handle requests on a first come, first serve basis. Earlier projects will also have an higher impact on the feature set of Elektra, thus you will less likely need to implement your own plugin.

## 416.6 See Also

- Progress can be viewed here
- We discussed about alternatives to Elektra here.

Best regards, Markus

# Chapter 417

# 0.8.20 Release

- guid: 547d48e2-c044-4a8e-9d32-ca6b6fb914d9
- author: Markus Raab
- pubDate: Thu, 31 Oct 2017 23:08:07 +0200
- shortDesc: New Website, puppet-libelektra, New Plugins

## 417.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit https://libelektra.org.

## 417.2 Highlights

This is by far the largest release in Elektra's history. In 2813 commits, 19 authors changed 1714 files with 92462 insertions(+) and 21532 deletions(-). The highlights are:

- libelektra.org: new website and puppet-libelektra
- plugin+bindings for Haskell and Ruby
- improved shell completion
- new plugins: yamlcpp, camel, mini, date, file, range, multifile, xerces, ipaddr

### 417.2.1 libelektra.org

Unfortunately this release was delayed. The reason for the delay is that our community server ( build server, web site,...) was compromised and we needed to reinstall everything from scratch.
We took advantage of the situation, and reinstalled everything properly managed by puppet-libelektra. With puppet-libelektra, you can directly set keys, specifications (validations), and even mount new configuration files from within Puppet.
Our community server is now completely managed by libelektra.
Thanks to Bernhard Denner, for rescuing us from the difficult situation, especially for the sprint shortly before the release.
As already already announced in December 2016 we completely reimplemented our website. Now all our websites are available via https. This release is the first one that includes the source code of the website and its snippet sharing functionality.
The backend for this snippet sharing website uses Elektra itself: both for its configuration and for the configuration snippets.
Thanks again to Marvin Mall for the awesome website.

#### 417.2.1.1 Haskell and Ruby

The Ruby binding, created by Bernhard Denner, was greatly improved and now includes libtools bindings. It is the first binding that goes beyond Elektra's main API. Bernhard Denner also added many `examples` that demonstrate how you can take advantage of the Ruby bindings.
Armin Wurzinger created a new binding for the functional language Haskell. He also added support for Haskell plugins. Due to generic CMake and C Code, plugins can be written exclusively in Haskell, without any glue code. Several Haskell examples already exist. The Haskell support is currently experimental.

### 417.2.2 Shell Completion

René Schwaiger added completion support for `Fish` in this release. We also extended our support for other shells: The new tool `kdb complete` suggests how to complete an Elektra path. It considers mountpoints and also takes bookmarks into account. Thanks to Armin Wurzinger for creating this useful utility. Our Zsh and fish completions already take advantage of `kdb complete`. Thanks to Sebastian Bachmann for taking the time to update the `zsh` completions.

### 417.2.3 New Plugins

See `plugin overview` to get an overview of the ever-growing number of plugins.
The `yamlcpp plugin` and camel plugin add first support for YAML.
The `mini plugin` is yet another minimal INI plugin.
Thanks to René Schwaiger.
The `date plugin` supports validation of dates according to three standards:

- `RFC2822`
- `ISO8601`
- `POSIX`

The `multifile plugin` allows us to integrate many configuration files via globbing with a single mount command. It supports `.d` configuration directories as often used today.
The `file plugin` interprets the content of a file as configuration value.
The `ipaddr plugin` adds support for IP address validation on systems that do not support `getaddrinfo`.
Thanks to Thomas Waser for creating these useful plugins.
The `xerces plugin` supplants the `xmltool plugin` and allows us to use XML files not following a specific schemata. Attributes are mapped to Elektra's metadata, multiple keys with the same names are mapped to arrays.
Thanks to Armin Wurzinger.

## 417.3 Documentation

The documentation was greatly improved within this release.

- Added "Hello, Elektra" and logging tutorial, thanks to René Schwaiger
- extended `FAQ`
- Christoph Weber (@krit0n) improved some tutorials
- options are passed to PDFLaTeX compiler, thanks to René Schwaiger
- small fixes, thanks to Dominik Hofer
- fix many spelling mistakes, use American english, fix formatting, fix+add links, unify title style, fix code blocks, add titles and fix the PDF manual a big thanks to René Schwaiger

## 417.4 Features

We added even more functionality, which could not make it to the highlights:

- DBUS support for qt-gui (listening to configuration changes): qt-gui gets a viewer-mode where configuration settings are immediately updated via DBus notifications, thanks to Raffael Pancheri With the new qt-gui and newer qt releases (~5.7) the qtquick experience is much smoother, for example, the tree view does not collapse on syncs anymore.
- Armin Wurzinger greatly improved the `JNA binding`. The build system now uses Maven to build them. Armin also added Doxygen documentation and a `script` to test the JNA binding using `Randoop`.
- The improved `curlget plugin`, is now able to upload configuration files, thanks to Thomas Waser and Peter Nirschl (CMake fixes).
- New command `kdb rmmeta`, thanks to Bernhard Denner
- `crypto plugin` and `fcrypt plugin`
  - The configuration option `gpg/key` was renamed to `encrypt/key`
  - The plugins now make sure that you configured them properly by validating key IDs
  - thanks to Peter Nirschl
- `fcrypt plugin`:
  - The plugin now list available GPG keys when config is missing
  - You can now specify signatures via the configuration option `sign/key`
  - New text mode, enabled by default (disable by setting `fcrypt/textmode` to `0`)
  - New option `fcrypt/tmpdir` allows you to specify the output directory of `gpg`
  - If you want to learn how to use the plugin please check out our new `ASCII cast`
  - thanks to Peter Nirschl
- Thomas Waser added useful scripts:
  - `mount-list-all-files` to list all mounted files.
  - `mountpoint-info` to provide more info about mountpoints.
  - `stash` to stash away Elektra's configuration, to be restored using `restore`.
  - `backup` to backup Elektra's configuration.
  - `restore` to restore a backup or stash.
  - `check-env-dep` allows users to check if environment has influence on configuration settings.
  - `change-resolver-symlink` allows users to change the default resolver.
  - `change-storage-symlink` allows users to change the default storage.
- limit min/max depth for `kdb ls` (`-mM`), thanks to Armin Wurzinger.
- conditionals: allow multiple assigns and conditions
- base64 also works as filter for binary data (not just encrypted data), thanks to René Schwaiger
- `csvstorage plugin`: compatibility with RFC 4180, thanks to Thomas Waser
- `gitresolver plugin`: improvements and update of libgit version, thanks to Thomas Waser
- `curlget plugin`: also allow uploading of configuration files, thanks to Thomas Waser

## 417.5 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
We added `explicit` to some C++ constructors in libtools and internally moved some typedefs. ModulesPlugin↩ Database now has protected members (instead of private). This might break code in special cases, but should not affect binary compatibility. As always we tested for binary compatibility. This time we had to revert some changes to keep libelektra-tools ABI compatible.
Furthermore:

- scripts now work on macOS (readlink and sed), thanks to Armin Wurzinger and René Schwaiger
- generalized error #60, "invalid line encountered"
- added new errors #164 - #187
- added private headerfiles `kdbglobal.h`, `kdbinvoke.h`

## 417.6 Notes for Package Maintainer

These notes are of interest for people maintaining packages of Elektra:

- LICENSE.md now contains only the BSD 3-Clause License (without any additional non-license text)
- AUTHORS renamed to AUTHORS.md
- NEWS.md is now a generated file containing all news concatenated
- CMake 2.8.8 is no longer supported, CMake 3.0 is now needed
- fix macOS `RPATH`, remove old policies, thanks to René Schwaiger
- new option `BUILD_DOCSET` to build DocSet, thanks to René Schwaiger
- new option `ENABLE_OPTIMIZATIONS` for `OPMPHM` preparation work, thanks to Kurt Micheli For this release, please keep `ENABLE_OPTIMIZATIONS` turned off. Currently the flag increases memory usage, without being faster.
- add `TARGET_TOOL_DATA_FOLDER` for installation of tool data

The following files are new:

- Libs: `libelektra-utility.so` ,`libelektra4j-0.8.20.pom.xml`, `libelektra-invoke` (needed by plugins: curlget, gitresolver, dini, blockresolver, multifile)
- Plugins: `libelektra-camel.so`, `libelektra-date.so`, `libelektra-file.so`, `libelektra-ipaddr.so`, `libelektra-mini.so`, `libelektra-multifile.so`, `libelektra-range.so`, `libelektra-xerces.so`, `libelektra-yamlcpp.so`
- Tools: `backup`, `mount-list-all-files`, `mountpoint-info`, `restore`, `stash`, `update-snippet-reposito`
- Tests: `change-resolver-symlink`, `change-storage-symlink`, `check-env-dep`, `check_bashisms`, `check_doc`, `check_meta`, `testmod_camel`, `testmod_crypto_openssl`, `testmod_date`, `testmod_file`, `testmod_ipaddr`, `testmod_jni`, `testmod_mini`, `testmod_range`, `testmod_simpleini`, `testmod_xerces`, `testmod_yamlcpp`, `testtool_plugindatabase`, `test_utility`

The following files were removed: `testmod_curlget`, `testmod_dpkg`, `testmod_profile`, `testmod_shell`, `testmod_spec`, `test_opmphm_vheap`, `test_opmphm_vstack`
The following files were renamed: `libelektra-1.jar` → `libelektra4j-0.8.19.jar`
In the Debian branch of the `git repo`, we now build upon the work of Pino Toscano. The branch allows building Debian packages of the release for Debian Stretch and Jessie.
Thanks to Pino Toscano for the high-quality packages.

## 417.7 Notes for Elektra's Developers

These notes are of interest for people developing Elektra:

- Added macros, thanks to René Schwaiger:
  - `ELEKTRA_NOT_NULL` is an assertion against null pointers
  - `ELEKTRA_MALLOC_ERROR` sets an error when allocation failed
  - `ELEKTRA_STRINGIFY` to quote a macro value
  - `ELEKTRA_PLUGIN_STATUS_ERROR`, `ELEKTRA_PLUGIN_STATUS_SUCCESS`, `ELEKTRA_PLUGIN_STATUS_NO_UPDATE` for return values of plugins.
- `ELEKTRA_STRINGIFY` used throughout, thanks to René Schwaiger
- use `(void)` instead of `()` (added `-Wstrict-prototypes`)
- new positions for global plugins, thanks to Mihael Pranjic
- Kurt Micheli added `generateKeySet` to randomly generate large key sets
- add vagrant and docker support, thanks to Christoph Weber (@krit0n)
- improve support for CLion, NetBeans and `oclint`
- portability improvements for logger, thanks to René Schwaiger
- metadata consistency check within source repo, thanks to Thomas Waser
- `ELEKTRA_PLUGIN_EXPORT` accepts macro as argument
- fallthroughs in switch statements are now marked with `FALLTHROUGH`
- introduce `print_result` to unify test output, thanks to René Schwaiger
- export `validateKey` as preparation for type plugin

## 417.8 Other

Various other changes happened in the code repository:

- kdb: errors are more colorful, add infos to report issues, catch signals for `kdb` tools to print errors on crashes, use `$EDITOR` if `sensible-editor` and `editor` is not found. René Schwaiger fixed preposition and format of the messages.
- added Spanish translation for qt-gui thanks to AdanGQ (@pixelead0)
- augeas plugin: error messages improved, export genconf (for WebUI to list all lenses)
- improvements for CentOS and Debian Packages, thanks to Sebastian Bachmann
- Travis improvements, fixes, and many build variants added, thanks to Mihael Pranjic and René Schwaiger
- `ronn` now always uses UTF-8 as encoded and is no longer required as essential dependency to get man pages, thanks to René Schwaiger
- GitHub now recognizes that we have a BSD licence, thanks to René Schwaiger
- uninstallation Script also uninstalls directories and Python files, thanks to René Schwaiger
- Kurt Micheli created a benchmark tool to generate large KeySets
- added/reformatted use cases, thanks to Daniel Bugl
- Thomas Wahringer prepared for a new theme on the website
- Arfon Smith updated metadata for Elektra's journal entry

## 417.9 Quality

In this release we had a focus on quality improvements:

- fixed all remaining ASAN problems, thanks to René Schwaiger and Armin Wurzinger (some tests are excluded when compiled with ASAN)
- fix many compilation warnings, thanks to René Schwaiger and Armin Wurzinger
- fixed many potential out-of-bound errors, thanks to René Schwaiger
- fixed warnings of Clang's static analyzers, thanks to René Schwaiger
- fixed cppcheck warnings, thanks to Armin Wurzinger
- fixed strict prototypes, thanks to Armin Wurzinger
- fixed and use scan-build (clang)
- fixed potential memory leaks on errors
- added assertions
- generate Java API tests with randoop which revealed bugs in jna bindings that were fixed, thanks to Armin Wurzinger
- Numerous fixes in the shell recorder, which does regression tests on Elektra's tutorials and READMEs, thanks to René Schwaiger and Thomas Waser

## 417.10 Fixes

Many problems were resolved with the following fixes:

- `kdb file`: never print errors, thanks to René Schwaiger
- plugin mathcheck: fixed regex #1094, thanks to Thomas Waser
- dbus: properly do unref and document how to integrate DBus, thanks to Kai-Uwe Behrmann
- dbus accepts announce=once which is used for `kdb mount-openicc` It protects against message floods in large configuration files, thanks to Kai-Uwe Behrmann for reporting
- plugin desktop: fix crash if `DESKTOP_SESSION` is missing
- shell-recorder: many fixes and improvements, thanks to Thomas Waser and René Schwaiger
- fix getopt positional parameters, thanks to Armin Wurzinger
- resolver: avoid silent errors of fchown/fchmod
- plugin fcrypt: fixes in file name handling to make leaks less likely (still needs tmpfs to be secure!), thanks to Peter Nirschl
- plugin jni: fix segfaults on errors, plugin is nevertheless tagged as experimental due to other problems
- plugin type: reject integers if garbage follows
- `kdb`: fix memleak when listing plugins
- many spelling fixes and fix typo in source of qt-gui: thanks to klemens (ka7)
- dpkg, fix file leakage, thanks to Armin Wurzinger
- plugin line: only skip parentKey if present
- plugin resolver: avoid failure after commit for files that cannot be removed

- plugin simpleini: handle more errors, make format parameter more robust thanks to Bernhard Denner
- plugin crypto: fix compilation errors for openssl versions on Debian 9, thanks to Peter Nirschl
- `kdb mv`: fail without keys also in recurse mode
- fix bashism, thanks to Armin Wurzinger
- qtgui: fix crash on unhandled exception on binary values, thanks to Raffael Pancheri

## 417.11 Outlook

We are currently working on following topics:

- Migration of LCDproc, OpenIcc, machinekit, ... to Elektra.
- A reimplementation of Elektra's core API in Rust (next to implementation in C).
- A user interface which generates restricted input fields based on the specification.
- YAML as configuration file format (next to INI, XML, JSON, TCL, ...).
- An mmap persistent cache.
- Improvements for the specification language.
- New APIs to be directly used by applications.
- An order-preserving minimal hash for O(1) lookup and iteration.
- Mainloop migration for notifications (currently only DBus, to be extended to Redis, ZeroMq).
- Improvements on the Website and snippet sharing to also handle misconfiguration.

## 417.12 Get It!

You can download the release from `here` or `GitHub`
The `hashsums are:`

- name: elektra-0.8.20.tar.gz
- size: 4740032
- md5sum: 0e906f1a1677a8bfb31d144e1eaeb3cf
- sha1: 5e33c49ae6e3b890c9267288fb9f321289910eb5
- sha256: e9cbc796e175685ccb6221f1dd5ea5c43832f545c40557c32b764ff5d567b312

The release tarball is also available signed by me using gpg from `here` or `GitHub`
Already built API documentation can be found `online` or `GitHub`.

## 417.13 Stay tuned!

Subscribe to the `RSS feed` to always get the release notifications.
For any questions and comments, please contact the issue tracker `on GitHub` or me by email using `elektra@markus-raab.org`.
`Permalink to this NEWS entry`
For more information, see `https://libelektra.org`
Best regards, Markus

# Chapter 418

# 0.8.21 Release

We are proud to release Elektra 0.8.21.

- guid: 7f5de1b1-6086-47a6-9922-cac08c898ae7
- author: Markus Raab
- pubDate: Fri, 22 Dec 2017 09:24:02 +0100
- shortDesc: FOSDEM, New Book, Maturing of Plugins

## 418.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit https://libelektra.org.
The news can be read rendered at our web server.

## 418.2 Highlights

In this release 8 authors created 307 commits and changed 217 files (5227 insertions, 1914 deletions). The highlights of this release are:

- Fosdem Talk about Elektra was accepted
- CC-licensed book about Elektra published
- Maturing of plugins
- Elektra with encryption
- Preparation for switch to INI as default storage

### 418.2.1 Fosdem Talk about Elektra in Main Track

We are happy to announce that there will be a talk about Elektra in one of the main tracks of Fosdem 2018:

- Title: Configuration Revolution
- Subtitle: Why it Needed 13 Years and How it Will be Done
- Day: Saturday 2018-02-03
- Start time: 15:00
- Duration: 50 min
- Room: K.1.105 (La Fontaine)

And a second talk in the Config Management DevRoom:

- Title: Breaking with conventional Configuration File Editing
- Subtitle: Puppet with a Key/Value API in a User Study
- Day: Sunday 2018-02-04
- Start time: 12:30
- Duration: 25 min
- Room: UA2.114 (Baudoux)

See you in Brussels at 3 and 4 February 2018!
I will also be present in the Config Management Camp directly after Fosdem in Gent.

### 418.2.2 CC-licenced Book About Vision of Elektra Published

I am proud to release a book with the title "Context-aware Configuration" describing:

- the last 13 years of Elektra (focus on last 4 years with the questionnaire survey and code analysis),
- the current state of Elektra, and
- the long-term goals of Elektra (context-aware configuration).

The Fosdem talk will cover some highlights from the book.
A huge thanks to everyone involved in the questionnaire survey, without you we would not have been able to collect all the information that led to the requirements for Elektra.
The LaTeX sources are available here and the compiled book can be downloaded from here.

### 418.2.3 Maturing of Plugins

- The new Directory Value plugin supports storage plugins such as YAJL and YAML CPP. It adds extra leaf values for directories (keys with children) that store the data of their parents. This way plugins that normally are only able to store values in leaf keys are able to support arbitrary key sets.
- The YAML CPP plugin reads and writes YAML data using yaml-cpp. The plugin supports arrays, binary data and metadata.
- The Camel plugin stores data as simplified YAML flow lists containing double quoted keys and values. For proper YAML support please use the YAML CPP instead.
- The mINI plugin reads and writes simple property list, separated by equal (=) signs.
- The xerces plugin allows Elektra to read and write XML data. The plugin uses Xerces-C++ for this task. It supports both arrays and metadata.
- The boolean plugin normalizes boolean values such as `0`, `1`, `true` and `false`.
- The crypto plugin and fcrypt plugin are described below.

### 418.2.4 Elektra With Encryption

The plugins `fcrypt` and `crypto` are now considered stable. They are no longer tagged as `experimental`. While `crypto` encrypts individual values within configuration files, `fcrypt` encrypts and/or signs the whole configuration file.
For this release Peter Nirschl prepared a demo showing Elektra's cryptographic abilities:

Thanks to Peter Nirschl for this great work!

### 418.2.5 Switch to INI

We plan to switch to INI as default storage instead of Elektra's infamous internal dump format.
As preparation work we implemented the `dini` plugin which transparently converts all `dump` files to `ini` files on any write attempt. Furthermore, we fixed most of the INI bugs which blocked INI to be the default storage.
Due to this progress we will likely switch to INI as default starting with the next release. If you want to, you can switch now by compiling Elektra with:
`-DKDB_DEFAULT_STORAGE=dini`
Or simply switch for your installation with:
`sudo kdb change-default-storage dini`
If you are already using `ini` as default, changing to `dini` will:

- add some overhead because `dini` always checks if a file uses the `dump` format, unless the `dump` plugin is not installed.
- add support for binary values using the `binary` plugin

  NOTE: INI (dini) was not completely ready for 0.8.21 thus we kept `dump` as default. `dini` is currently an experimental plugin.

## 418.3 Other New Features

We added even more functionality, which could not make it to the highlights:

- `kdb rm` now supports `-f` to ignore non-existing keys
- use `%` as profile name to disable reading from any profile
- The new function `elektraArrayDecName`:

```
int elektraArrayDecName (Key * key);
```

decreases the index of an array element by one. It can be used to reverse the effect of `elektraArrayIncName`, thanks to René Schwaiger

## 418.4 Documentation

We improved the documentation in the following ways:

- We renamed our beginner friendly issues to "good first issue" as recommended by GitHub.
- In many parts of the documentation we already switched to American spelling thanks to René Schwaiger
- Added more `automatic spelling corrections` thanks to René Schwaiger
- Fixed many spelling mistakes thanks to René Schwaiger
- We extended the ReadMe of the `jni` plugin. The ReadMe now also contains information about the Java prerequisites of the `jni` plugin on Debian Stretch.
- Improved notes about testing thanks to Thomas Wahringer
- qt-gui: give hints which package to install
- The build phrases `jenkins build all please` and `jenkins build doc please` were `documented` thanks to René Schwaiger
- Documentation for libelektra-invoke was added

## 418.5 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
All unit tests of 0.8.20 run successfully with Elektra 0.8.21. There are, however, some additions and changes in rarely used interfaces:

- added `elektraArrayDecName` and `elektraArrayValidateName` in libease
- fixed `kdbinvoke.h` interface: make structure private and complete API
- fixed `xmlns` and `xsi:schemaLocation` to be `https://www.libelektra.org`
- the private header file `kdbopmphm.h` got nearly rewritten

## 418.6 Notes for Maintainer

These notes are of interest for people maintaining packages of Elektra:

- We added the following files in this release:
  - `libelektra-dini.so`
  - `libelektra-directoryvalue.so`
  - `testmod_directoryvalue`
- The following plugins are not marked as experimental anymore:
  - `camel`
  - `crypto`
  - `mini`
  - `xerces`
  - `yamlcpp`
- The binding `intercept-fs` is now marked more clearly as experimental
- The `jni` plugin is again experimental because it does not work with some Java systems. For the `lua` plugin there are also problems with some Lua systems.

## 418.7 Notes for Elektra's Developers

These notes are of interest for people developing Elektra:

- From now on release notes are written as part of PRs
- Elektra Initiative is spelled as two words
- At some more places we switched to use the logger, thanks to René Schwaiger
- Shell Recorder got many improvements, see below in Testing. Please use it.
- The plugin's template now adds all placements within backends by default (must be removed accordingly).
- We now warn if plugins do not have any placement.
- Please prefer -log and -debug builds
- The build server now understands the build phrase `jenkins build all please` thanks to René Schwaiger. Please use it carefully, since it puts our `build server` under heavy load.
- Markdown Shell Recorder Syntax recommended when reporting bugs.
- Elektra's `Dockerfile` was improved and simplified, thanks to Thomas Wahringer.

- Add more Explanations how to do Fuzz Testing
- Started documenting disabled tests in doc/todo/TESTING
- You now can use tests/icheck.suppression to disable already checked API changes.
- The (hopefully) last Sourceforge references were removed and a redirection page was added, thanks to @the-Arioch for reporting.

## 418.8 Testing

- AFL unveiled some crashes in INI code
- fix OCLint problems, thanks to René Schwaiger
- fix ASAN problems, thanks to René Schwaiger
- disabled non-working tests
- Shell recorder
- Benchmark optionally also works with OpenMP, thanks to Kurt Micheli
- The Shell Recorder now uses kdb-static or kdb-full if kdb is not available (BUILD_SHARED=OFF)

## 418.9 Fixes

Many problems were resolved with the following fixes:

- fix use of dbus_connection_unref(NULL) API thanks to Kai-Uwe Behrmann
- Properly include headers for std::bind thanks to Nick Sarnie
- qt-gui: assure active focus on appearance selection window thanks to Raffael Pancheri
- René Schwaiger repaired the boolean plugin:
  - wrong metadata was used
  - plugin configuration was missing
  - documentation was missing
  - logging code was added
- René Schwaiger repaired many problems different build agents had
- kdb info -l does not open KDB anymore.
- change-resolver-symlink and change-storage-symlink now correctly use @TARGET_↩ PLUGIN_FOLDER@
- date plugin will be removed on attempts to compile it with gcc 4.7, thanks to René Schwaiger
- C plugin: storage/c metadata added
- fix disabling documentation in CMake, thanks to Kurt Micheli
- Simplify elektraArrayValidateName, thanks to René Schwaiger

## 418.10 Outlook

The Order Preserving Minimal Perfect Hash Map (OPMPHM) is ready to extend ksLookup. The implementation of the randomized Las Vegas hash map algorithm is in a final stage and the heuristic functions that ensure time and space optimality are backed up by benchmarks. Thanks to Kurt Micheli, the next release will include the OPMPHM!

## 418.11 Get It!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.8.21.tar.gz
- size: 4712043
- md5sum: d627a01a0249fde46e80042c848d4521
- sha1: a7659a7bb1b2388d03cdf0084160de612e5c4511
- sha256: 51892570f18d1667d0da4d0908a091e41b41c20db9835765677109a3d150cd26

The release tarball is also available signed by me using GnuPG from here or GitHub
Already built API documentation can be found online or GitHub.

## 418.12 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the issue tracker on GitHub or me by email using elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Markus Raab for the Elektra Initiative

# Chapter 419

# 0.8.22 Release

- guid: 4884a54f-996a-4564-a138-38a70166fed7
- author: Markus Raab
- pubDate: Tue, 27 Feb 2018 19:35:58 +0100
- shortDesc: Logo, INI, Lookup

We are proud to release Elektra 0.8.22! In 429 commits, 8 authors changed 548 files with 60369 insertions(+), 6783 deletions(-).

## 419.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
For a small demo see here:

You can also read the news `on our website`
You can read the `FOSDEM interview` and watch the `FOSDEM main talk` given recently.
Elektra is now an official part of `Homebrew`.

## 419.2 Highlights

- New Logo and Website Theme
- INI plugin greatly improved
- Notifications API and Bindings for Asynchronous I/O
- Plugin Processes
- Lookup with the Order Preserving Minimal Perfect Hash Map

### 419.2.1 New Logo and Website Theme

We are proud to present our new logo. It has a new shape and cooler colors.
Thanks to Philipp Frei!
We also gave the website a new look. It has the colors from the logo and new fonts ( `Lato` and `Libre Franklin`) that improve readability and add to a unique look. The restructured start page contributes to the new look as well.
We also updated asciinema-player to 2.6.0.
Thanks to Thomas Wahringer.
We also fixed security issues in the Website due to an old version of jquery, thanks to Marvin Mall.
Thanks to Bernhard Denner for keeping our infrastructure running.

### 419.2.2 INI plugin greatly improved

- dini is no longer experimental anymore and adds the binary plugin.
- We added a crash test for the INI plugin that feeds the plugin with problematic input data we determined using AFL
- We fixed various small bugs that could potentially cause the INI plugin to crash and fixes the problems as reported by AFL
- The INI plugin now converts a section to a normal key-value pair if you store a value inside it. This has the advantage that you will not lose data unexpectedly anymore.
- The INI plugin should now read most key-value pairs containing delimiter characters (=) properly.

Thanks to René Schwaiger!
Nevertheless, we did not switch to INI as default format. This has some reasons:

- In many workflows, dump is the better choice: e.g. with kdb editor you can edit any parts of Elektra with any format (e.g. INI) more safely. (With the INI plugin in some of these situations metadata is wrongly shown.)
- The code base of INI is not fully tested and partly not well understood.
- We plan to switch to a newly written parser (most likely YAML) and want to avoid that users have two migrations. The migration from dump is easier (especially compared with INI) because the dump format is recognisable without ambiguity. (Thus the INI file parses nearly any file, it is hard to detect that a file is not INI)

But for those who want to switch, the migration will be smooth: The dini plugin makes sure that old dump files are still being read. Only when writing out configuration files, configuration files are converted to INI. To change to INI during compilation, simply use:
```
-DKDB_DEFAULT_STORAGE=dini
```
Or simply switch for your installation with:
```
sudo kdb change-default-storage dini
```
You can also mount INI (or dini) as root:
```
sudo kdb mount default.ini / dini
```

### 419.2.3 Notification API and Bindings for Asynchronous I/O

This release contains an experimental implementation of Elektra's notification feature. This feature enables applications to get updates when configuration is changed at run-time. For more details see the preview tutorial at doc/tutorials/notifications.md
The Notification API is implemented by a new library called elektra-notification. To use the library you need the new internalnotification plugin. Since the plugin is experimental it needs to be enabled when building Elektra from source (e.g. by passing -DPLUGINS="ALL;-EXPERIMENTAL;internalnotification" to cmake).
New bindings for asynchronous I/O called "I/O bindings" also have been added. These bindings allow Elektra's plugins and other parts to perform *asynchronous* operations. I/O bindings are opt-in for application developers. New features of Elektra that take advantage of I/O bindings will have fallbacks where viable. These fallbacks will use synchronous I/O thus keeping the status quo.
This release includes an experimental I/O binding for uv. The interface for I/O bindings is currently experimental and might change in the future.
Elektra's notification feature is not complete yet. So-called "transport plugins" will be responsible for sending and receiving notifications using different protocols or libraries (like ZeroMQ or D-Bus). These plugins will make use of the new I/O bindings. We plan to introduce the first transport plugins with the next release of Elektra.

### 419.2.4 Plugin Processes

A new library called pluginprocess has been added. This library contains functions that aid in executing plugins in a separate process. This child process is forked from Elektra's main process each time such plugin is used and gets closed again afterwards. It uses a simple communication protocol based on a KeySet that gets serialized through a pipe via the dump plugin to orchestrate the processes.

Such a behavior is useful for plugins which cause memory leaks to be isolated in an own process. Furthermore this is useful for runtimes or libraries that cannot be reinitialized in the same process after they have been used.

### 419.2.5 Lookup with the Order Preserving Minimal Perfect Hash Map

The `ksLookup (...)` has a new search algorithm, that acts as an alternative to the binary search. The Order Preserving Minimal Perfect Hash Map (OPMPHM) is a non-dynamic, randomized hash map and is very effective for mostly static configurations. The OPMPHM can be enabled for a search by passing the in `kdbproposal.h` defined option `KDB_O_OPMPHM` to the lookup. Be aware that if the KeySet changes often using the OPMPHM might not be a good idea, read more about the `OPMPHM`.
When you are not sure if the OPMPHM will speed up you searches, wait for the next release, that one will include a hybrid search algorithm that combines the best properties of both search algorithms.
To disable OPMPHM (e.g. on systems with tight memory constraints), you can pass `-DENABLE_↩ OPTIMIZATIONS=OFF`

## 419.3 Other New Features

We added even more functionality, which could not make it to the highlights:

- The Web UI was greatly improved, thanks to Daniel Bugl The version of clusterd was increased from 1.0.0 to 1.1.0.
- Elektra is now part of the official `Homebrew repository`. We still provide a `tap`, if you want to install Elektra together with plugins or bindings that require additional libraries.
- The building and linking of the Haskell bindings and Haskell plugins has been `greatly improved`.
- The invoke library can now `report errors` upon opening/closing a plugin, thanks to Armin Wurzinger.
- The `YAML CPP plugin` does not require `Boost` anymore, if you installed `yaml-cpp 0.6`.
- Improved colored output in `kdb` tool.
- We added two build jobs: docker and haskell. Thanks to Bernhard Denner and Armin Wurzinger.
- `YAML CPP` does not write binary data to a config file, if you forget to load the `Base64 plugin`.
- `YAML CPP` now encodes empty binary keys as NULL values (~), and also adds the metakey `binary` for such values automatically.

## 419.4 Documentation

We improved the documentation in the following ways:

- We have `documented how you can setup a build node for Jenkins using a Docker container` We also provide an example Dockerfile based on Debian Stretch for that purpose, thanks to Armin Wurzinger.
- Document how `rlwrap` might be used for `kdb shell`.
- Fixed documentation in `hosts` plugin.
- Greatly improved the `license documentation` in `debian/copyright` in the `debian` branch, thanks to Thomas Wahringer.
- Various fixes in doc/METADATA.ini

## 419.5 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
We executed old test cases against the latest Elektra version and all tests passed.
In `kdbinvoke.h` we changed the API so that `elektraInvokeOpen` and `elektraInvokeClose` can yield error messages.
We added:

- the public headerfiles `kdbnotification.h`, `kdbio.h`, `kdbiotest.h`.
- the private headerfiles `kdbnotificationplugin.h`, `kdbioprivate.h`.

## 419.6 Portability

- Fix bash shebang of bash scripts, thanks to Jakub Jirutka
- Remove unportable unneeded asm, thanks to Timo Teräs and Jakub Jirutka
- Fixed syntax in shell recorder, thanks to René Schwaiger
- Used `mktemp` in `check_distribution.sh` to allow parallel run of test cases, thanks to Armin Wurzinger.

## 419.7 Notes for Maintainer

These notes are of interest for people maintaining packages of Elektra:

- `dini` is no longer experimental.
- CMake: `BINDINGS` now behaves like `PLUGINS` By default now all MAINTAINED bindings except EXPERIMENTAL and DEPRECATED are included. For more details see `/doc/COMPILE.md`. To include both intercept bindings, you now need to write `INTERCEPT`, to only include getenv interception `intercept_env`. `intercept` in lower-case does not work anymore.

The following files are new:

- Libs: `libelektra-notification.so`, `libelektra-io.so`, `libelektra-io-uv.so`, `libelektra-pluginprocess.so`
- Headers: `kdbgetenv.h`, `kdbio.h`, `kdbpluginprocess.h`
- Plugins: `base64/Base64.pdf`
- Binaries: `getenv`, `test_context`, `test_fork`, `test_getenv`, `test_ks_opmphm`, `test_opmphm`
- Other: `elektra-io.pc`

The following files were removed:

- Tests: `testmod_semlock`

## 419.8 Notes for Elektra's Developers

These notes are of interest for people developing Elektra:

- We now use `clang-reformat-5.0` for formatting. Please upgrade your clang.
- Build Agent `ryzen v2` was added to speed up `jenkins build all please`, thanks to Armin Wurzinger.

- Travis maintenance (Qt 5 and other problems), thanks to René Schwaiger.
- BINDINGS was greatly improved and the CMake functions were simplified. Bindings now also have a README.md with metadata. A big thanks to Thomas Wahringer.
- Decision: Logging with ELEKTRA_LOG is only for C/C++.
- Including kdberrors.h in a C++ files now also works, if you do not add the statement

```
using namespace ckdb;
```

before you import kdberrors.h.

- The CMake code for Elektra's Qt-GUI now detects the location of Qt 5 automatically if you installed Qt via Homebrew.
- All Shell Recorder tests should not now correctly restore your old configuration after you execute them, thanks to René Schwaiger.
- The Base64 plugin does not encode empty binary values in meta mode anymore. This update allows plugins such as YAML CPP to handle empty keys properly.

## 419.9 Fixes

Many problems were resolved with the following fixes:

- kdb global-umount also removed other mountpoints
- We fixed internal inconsistency in the CMake code of the Augeas plugin
- We fixed the haskell bindings and plugins on Debian Stretch and added a new build server job to test that in the future.
- Cleanup in list plugin, thanks to Thomas Wahringer
- The Shell Recorder now exports and imports the root of a mountpoint instead of the mountpoint itself. This change makes sure that Shell Recorder test do not change the configuration directly below the specified mountpoint.

## 419.10 Get It!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.8.22.tar.gz
- size: 5885118
- md5sum: 5cbd9e33daf51f6f33791ff3f99342a6
- sha1: 4954ff16cfe7dc69e45e6a748556262b8fb1a9fa
- sha256: 962598315619d5dff3a575d742720f076dc4ba3702bd01609bfb7a6ddb5d759f

The release tarball is also available signed by me using GnuPG from here or GitHub
Already built API documentation can be found online or GitHub.

## 419.11 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the issue tracker on GitHub or me by email using elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 420

# 0.8.23 Release

- guid: 9a9ab08b-9ca0-4242-b617-5a8b21ea42a0
- author: Markus Raab
- pubDate: Sun, 13 May 2018 08:57:15 +0200
- shortDesc: Notification, Web UI, Build System

We are proud to release Elektra 0.8.23.
In 717 commits 11 authors changed 835 files with 31144 insertions(+), 21773 deletions(-).

## 420.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a specified, global, hierarchical key database. For more information, visit `https://libelektra.org`.
For a small demo see here:

You can also read the news `on our website`

## 420.2 Highlights

- Notification: New transport plugin
- Web UI greatly improved
- Overhaul of Build System and Daily Stretch Repository

### 420.2.1 Notification: New transport plugin

To keep persistent configuration settings in sync with the configuration settings of applications, notifications are needed. For notifications it is important that they do not block the execution of the applications. In Elektra we achieve this using transport plugins.
Elektra's notification feature has received its first transport plugin pair: D-Bus. Transport plugins provide a link between applications using Elektra. These plugins send and receive notifications when a key is modified within the key database. The existing `dbus` plugin has been modified to use an asynchronous I/O binding for sending messages, if available. The new `dbusrecv` plugin is responsible for receiving messages sent from the `dbus` plugin and other sources with the same `message format`.
For more details see the `notification tutorial` or the `example applications`.
Thanks to Thomas Wahringer.

### 420.2.2 Web UI greatly improved

The goal of the Web UI is to provide safe and unified access to all configuration settings of a system. Different to other UIs, it generates its interface according specifications as found in Elektra.

For example, if a configuration setting only has a number of choices, you get exactly these choices within the user interface.
To get outstanding usability, Web UI now provides:

- undo functionality
- visibility functionality to hide irrelevant configuration settings
- built-in validation for many types of configuration settings
- support for arrays
- descriptions of configuration settings embedded in the user interface

Furthermore:

- The Web-UI now is able to install itself via CMake.
- The API was updated for `elektrad` and `webd` (former clusterd).

`Read here to get started`.

Note that new version of the Web UI requires Elektra 0.8.23 or later.

Thanks to Daniel Bugl.

### 420.2.3 Overhaul of Build System and Daily Stretch Repository

We started to overhaul our build system to improve build times and responsiveness. It focuses heavily on containerisation to improve hardware utilization.
If you are interested in `#devops` have a look at our `Jenkinsfile`.
Daily builds Debian packages for Stretch are available again in our `stretch repository`. Add it to your `sources.list`:

```
deb     [trusted=yes] https://debian-stretch-repo.libelektra.org/ stretch main
deb-src [trusted=yes] https://debian-stretch-repo.libelektra.org/ stretch main
```

Thanks to Lukas Winkler.

## 420.3 Other New Features

We added even more functionality, which could not make it to the highlights:

- A new experimental `I/O binding for glib` has been added. It can be used to integrate the notification feature with applications based on glib.
- The Order Preserving Minimal Perfect Hash Map (OPMPHM), used to speed up the lookups, got optimized and a benchmark was added, thanks to Kurt Micheli
- We added a script that calculates the complexity of configuration settings based on their specification, thanks to Anton Höß l
- `kdb ls` now has `-0` option to allow key names with newlines (needed for Web UI)
- The `csvstorage` now can treat selected columns to be part of the key. Error messages were improved. thanks to Thomas Waser

## 420.4 Other News

- We added a tutorial about securing the integrity and confidentiality of configuration values, thanks to Peter Nirschl
- Peter Nirschl finished his `thesis` ( `signature`). It includes a benchmark of different cryptographic providers.
- Markus Raab gave a `talk` at Linuxwochen Wien (in German). For similar talks in English, please refer to the `FOSDEM talks`.
- We replaced the word "project" to "initiative" in the `Code of Conduct` (project has per definition an end date).

## 420.5 Documentation

We improved the documentation in the following ways:

- FAQ was extended by `Why do I need Elektra if I already use configuration management tools?`
- Documentation about the recommended `environment` for test runs were added
- uniformly add `.` at end of short help
- Logo for Doc Set was added and logo for favicon was updated, thanks to René Schwaiger
- template of design decisions was updated to use the words problem (instead of issue) and rationale (instead of argument).
- `METADATA.ini`:
  - added visibility (as used in Web UI)
  - added type (only check/type existed)
  - plenty of metadata is now used by Web UI
- update documentation for type plugin that `check/type/min` and `check/type/max` are deprecated
- Fixed various spelling mistakes, thanks to René Schwaiger
- Document limitations of resolver (kdbOpen fails if no home directory found) and json/yaml plugins (intermixing of array and objects not detected, which is possible in Elektra but impossible in JSON)
- Required environment to run tests is documented.
- A decision about `deferred plugin calls` has been made and implemented, thanks to Thomas Wahringer.

## 420.6 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
We removed:

- the not used error code `12` from `kdb mv` from docu

We changed:

- `kdb get`, `kdb mv` and `kdb cp` now use error code `11` if keys are not found
- cascading keys as arguments to `kdb cp` and `kdb mv` now fail instead of doing something unexpected, thanks to René Schwaiger for reporting

Shell scripts:

- cp and mv no longer accept cascading keys.

## 420.7 Notes for Maintainer

These notes are of interest for people maintaining packages of Elektra:

- Documentation is updated that `CMake3` is required. thanks to Lukas Winkler for reporting.
- To run all tests successfully, the `spec` and `list` plugin is required. So if `ENABLE_TESTING` is checked, CMake checks the presence of a storage, a resolver, the list and the spec plugin, thanks to René Schwaiger

- Tests no longer clear environment or reset locales. This fixes lua and dbus problems but might cause problems if TMPDIR is set, thanks to Lukas Winkler
- This will be the last release supporting Debian Wheezy (LTS support will stop in May). Directly after the release, Jessie (oldstable) with gcc 4.8.4 will be the oldest supported platform.

We added:

- the private headerfiles `kdbnotificationinternal.h`, `kdbioplugin.h`.
- the headerfiles `kdbio_glib.h` and `kdbio_uv.h`
- the plugin `libelektra-dbusrecv.so`
- the scripts `build-web`, `run-elektrad`, and `run-web`
- the test case `testmod_dbusrecv`
- the constant `ENABLE_ASAN` in the constants plugin
- several man pages such as: `kdb-run-elektrad.1` and `kdb-run-web.1`

We removed:

- `Base64.pdf` is not installed anymore
- doxygen-generated man pages such as: `doc_docker_jenkinsnode_README_md.3elektra`, `doc_docker_README_md.3elektra`, and `doc_vagrant_README_md.3elektra`

## 420.8 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Error page that is shown if no JavaScript is enabled now more clearly says that the Website only renders content from the repo and only contains free JavaScript.
- The `FAQ` is now more visible (added to "getting started").
- The `Code of Conduct` was added.

## 420.9 Notes for Elektra's Developers

These notes are of interest for people developing Elektra:

- `. run_env` is a script to be sourced from the build directory. It sets environment variables, so that Elektra from the build directory is used (instead of the installed one).
- All current versions of Clang-Format (6.0+) and the outdated Clang-Format 5 will now produce exactly the same output for the whole codebase, thanks to René Schwaiger.
- To make enums nicely formatting, make sure at least one member is documented.
- You can now add a `Markdown Shell Recorder` test for a plugin via the CMake function `add_plugin` by adding `TEST_README`. Furthermore `TEST_REQUIRED_PLUGINS` allows us to specify which additional plugins are required, thanks to René Schwaiger
- `const` was added to exceptions in catch blocks thanks to René Schwaiger
- We now mention to read `doc/DESIGN.md` in the contributing guidelines.
- The CMake functions
  - `add_plugin`

  - add_msr_test
  - add_msr_test_plugin, and the new
  - add_shell_recorder_test

  now allow you to specify a list of required plugins for Shell Recorder and Markdown Shell Recorder tests.

- The Markdown Shell Recorder now compares the whole output of stderr with the text following the directive STDERR:, thanks to René Schwaiger
- You can now leave the text following the directive STDERR: in a Markdown Shell Recorder test empty:

```
true # Print nothing to `stderr`
# STDERR:
```

. The Markdown Shell Recorder will then check if the command printed nothing to the standard error output.

- The Shell Recorder now also prints the content of the protocol file if a test was unsuccessful or you used the command switch -p, and always cleans up the protocol, thanks to René Schwaiger
- We added an Markdown Shell Recorder test for the Constants plugin.
- The Markdown Shell Recorder now prints the path of the test file. thanks to René Schwaiger
- The Haskell binding now explicitly requires GHC installed with a minimum version of 8.0.0 during cmake thanks to René Schwaiger and Lukas Winkler
- If any of the tests in make run_memcheck fail valgrind will now set an exit-code which will get picked up by make, thanks to Lukas Winkler
- We introduced Git reference repositories to save I/O on our build system, thanks to Lukas Winkler
- Set LD_LIBRARY_PATH in all tests removing the need to specify it for running ctest, thanks to Lukas Winkler
- Provide the RUN_SERIAL property to all tests that can not be run in parallel, thanks to Lukas Winkler
- Speeding up your test runs via ctest -j is now possible, thanks to Lukas Winkler
- We now disable the Xerces plugin if you use GCC with enabled ASAN to build Elektra. This update makes sure that you do not build the plugin with compilation settings that are known to cause problems.
- Documentation and debugging capabilities of Markdown Shell Recorder were improved.
- We added style guidelines for CMake code to doc/CODING.md.

## 420.10 Fixes

Many problems were resolved with the following fixes:

- YAML CPP now also saves key values directly below a mountpoint correctly, thanks to René Schwaiger
- If you use a minimal configuration ( dump, resolver, list, and spec), all test of the test suite now finish successfully again, thanks to René Schwaiger
- small refactoring in kdb-test
- The Haskell plugin failed to build if the Haskell bindings were not included explicitly by name.
- Fix invalid handling of keynames in the spec plugin.
- The Shell Recorder counts the number of executed tests properly again.
- CMake now fails if the required plugins list or spec (on non- MinGW platforms) are missing from the current build configuration.

- The `Lua`, `Python 2`, `Python`, and `Ruby` plugins now require SWIG bindings for the corresponding programming language, thanks to René Schwaiger
- The `type checker` now also honors `type` next to `check/type`
- Fix various compiler warnings
- The detection of Botan, Libgcrypt, LibGit2 and OpenSSL now also works properly, if we treat warnings as errors (compiler switch `-Werror`), thanks to René Schwaiger
- The `multifile plugin` now passes the child config to the storage plugins too and also handles symlinks correctly, thanks to Thomas Waser

## 420.11 Workshop

`Elektra Initiative` are the people behind Elektra. Our goal is to build up expertise with configuration settings and improve the situation in the FLOSS landscape. To learn more about the needs of configuration-wise non-trivial FLOSS applications, we have workshops. After a successful workshop with the LCDproc's maintainer, the next Workshop will be with people from KDE.
We will use the opportunity of `Akademy` being in Vienna. We already got positive feedback from kconfig maintainers (David Faure and Aleix Pol).
If you are interested, you can sign up. We are looking forward to an informative, interactive and interesting workshop!

## 420.12 Outlook

We are currently working on following topics:

- Klemens Böswirth: `elektrifying LCDproc` After some setbacks (the two original developers who wanted to work on LCDproc resigned because of job duties) LCDproc development restarted now successfully. The new plan is to have more intermediate stages. In particular the first integration will be a minimal invasive integration without high-level API.
- Armin Wurzinger: type system for Elektra's specification language
- Anton Hössl: Puppet
- Daniel Bugl: Web UI
- Hani Torabi Makhsos: reduce community entry barriers
- Kurt Micheli: order preserving minimal perfect hash map
- Lukas Winkler: continous integration
- Mihael Pranjić: mmap plugin (i.e. avoiding reparsing of configuration files)
- Peter Nirschl: integrity and confidentiality
- René Schwaiger: parsing techniques
- Thomas Wahringer: notification techniques
- Thomas Waser: Linux distribution based on Elektra
- Ulrike Schaefer: generate shell completion files from Elektra's specification
- Vanessa Kos: misconfiguration bug database
- We created a proof of concept for a Chef resource and an Ansible module successfully setting Elektra's keys. They are not yet published. If you are interested on this preliminary work, please contact us.

## 420.13 Get It!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.8.23.tar.gz
- size: 5870069
- md5sum: 0a065ed381a59b6213bd46fd3c82ba83
- sha1: 0727b420ff721e654b0ba6ab1d0c78e5e2341d26
- sha256: f1d3cd4888ba3ef47c1327cbddf21dff7be289f94217f12e5e93105273ca6c48

The release tarball is also available signed by Markus Raab using GnuPG from here or GitHub
Already built API documentation can be found online or GitHub.

## 420.14 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the issue tracker on GitHub or Markus Raab by email using elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 421

# 0.8.24 Release

We are proud to release Elektra 0.8.24.

- guid: 889b700d-9eac-4eff-9a3d-f6fb15c3d9da
- author: Markus Raab
- pubDate: Sat, 18 Aug 2018 18:13:40 +0200
- shortDesc: Elektra Web, Notifications, Type System

## 421.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
For a small demo see here:

You can also read the news `on our website`

## 421.2 Highlights

- Elektra Web
- Notifications
- KDE Workshop
- Type System Prototype
- Chef Cookbook

### 421.2.1 Elektra Web 1.6

The new release of Elektra Web features many UX improvements from the usability test!

Try it out now on: `http://webdemo.libelektra.org/`

#### 421.2.1.1 1.5 changelog:

- search completely reworked - it does not act as a filter on already opened keys anymore, and instead searches the whole key database - feedback from the search was also greatly improved (pulsating while searching, glowing blue when done)
- added "abort" buttons to dialogs to revert actions
- added "create array" button to easily create arrays

- removed confirmation dialog before deletion (undo can be used instead)
- created a docker image: `elektra/web`
- implemented auto-deployment of webdemo.libelektra.org
- small fixes:
  - updated visibility levels
  - removed "done" button in main view
  - fixed issues with the opener click area
  - remove metakeys when they are set to the default value or empty/0
  - improved keyboard support
  - fixed many small issues ( `#2037`)

#### 421.2.1.2 1.6 changelog:

- fixed bugs related to arrays ( `#2103`)
- improved performance of search for many results
- added 404 page for invalid instance ids
- implement drag & copy by holding the Ctrl or Alt key
- add button to show error details
- allow deleting all keys in a namespace

Thanks to Daniel Bugl.

### 421.2.2 Notifications

Elektra's notification feature which allows applications to keep persistent configuration settings in sync with the key database and other applications was greatly improved with this release:

- The `notification API` now supports more types and has improved support for callbacks.
- With the addition of the `zeromqsend` and `zeromqrecv` plugins together with the `hub-zeromq` tool we have an alternative to the D-Bus transport plugins ( `dbus` and `dbusrecv`).
- The new asynchronous I/O binding for `ev` is the third I/O binding - so notifications can be used in applications using `glib`, `uv` or `ev`. If your application uses a different library please check out the `"How to create your own I/O binding" section` in the `notification tutorial`.
- Notifications can be used to reload KDB after Elektra's configuration (e.g. mountpoints or globally mounted plugins) has changed. We added a `how-to to the notification tutorial` that explains the required steps and the `"notificationReload"` example with the complete code.

More details can be `found in this news`. Check out the updated `notification tutorial` and notification examples ( `polling`, `async` and `reload`.

### 421.2.3 KDE Workshop

At `Akademy 2018` we had a successful `Config Workshop`.
We generally agreed that misconfiguration is important and the situation in FLOSS needs to improve. We discussed how Elektra can be used in KDE and came up with the idea that KConfig could be moved to a Elektra plugin. Then KConfig could be patched to use Elektra instead. This would lead to the situation that KDE users would have the same user experience with the advantages of Elektra, like:

- Elektra Web and qt-gui to safely modify all settings

- modification of settings via configuration management tools
- switch to other configuration file formats (e.g., XML or YAML for plasma)
- provide notification with main-loop integration
- plugin system for LDAP support and similar features

For more information see the Slides

### 421.2.4 Type System Prototype

Elektra supports specifying the semantics of keys via metakeys in the spec namespace. An example is the metakey check/range which can be used to specify that a key only holds numbers in a given range. Another metakey is check/enum which only allows specific keywords to be the content of a key. Up to now these semantics are only being checked at runtime. Therefore a type system was developed to be able to check configuration specifications statically. As an example, it would detect when one accidentally adds both a range and an enum check if their possible contents are not compatible with each other.
The type system is available as a plugin that gets mounted along with a configuration specification into the spec namespace. Furthermore we include a set of type definitions for commonly used metakeys such as check/range, check/enum, check/validation, fallback or override.
For more details see the typechecker readme
Thanks to Armin Wurzinger.

### 421.2.5 Chef Cookbook

Next to the Puppet Resource Type we now also prepared a Chef Cookbook which allows us to use Elektra from within Chef.
For example, to set mount a configuration file, you can use:

```
kdbmount 'system/hosts' do
        file '/etc/hosts'
        plugins 'hosts'
        action :create
end
```

And to add an hosts entry, you can use:

```
kdbset '/hosts/ipv4/showthatitworks' do
        namespace 'system'
        value '127.0.0.33'
        action :create
end
```

> Note that currently kdb is invoked and Elektra needs to be installed for managed systems.

Thanks to Michael Zronek and Vanessa Kos.

## 421.3 Plugins

### 421.3.1 CCode

- We fixed various warnings in the source code reported by OCLint. _(René Schwaiger)_
- The plugin now also encodes and decodes key names in addition to key values. _(René Schwaiger)_

### 421.3.2 CPP Template

- We added a new template for C++ based plugins. To create a plugin based on this template, please use the command

```
scripts/copy-template -p pluginname
```

, where pluginname specifies the name of your new plugin. _(René Schwaiger)_

### 421.3.3 Crypto

- The crypto plugin now uses Elektra's libinvoke and the base64 plugin in order to encode and decode Base64 strings. This improvement reduces code duplication between the two plugins. _(Peter Nirschl)_

### 421.3.4 CSVStorage

- Changed behavior of export to validate the structure of exported keys only. _(Thomas Waser)_

### 421.3.5 Directory Value

- We rewrote the plugin using C++. _(René Schwaiger)_
- `Directory Value` now also supports nested arrays. _(René Schwaiger)_
- The plugin now also adds leafs for a key, if its value is null or the empty string. _(René Schwaiger)_

### 421.3.6 fcrypt

- The `fcrypt` plugin will consider the environment variable `TMPDIR` in order to detect its temporary directory. See [#1973] _(Peter Nirschl)_

### 421.3.7 fstab

- The `fstab` plugin now passes tests on musl builds. _(Lukas Winkler)_

### 421.3.8 Haskell

- An issue when building Haskell plugins with a cached sandbox is fixed in case a Haskell library bundled with Elektra gets changed. _(Armin Wurzinger)_
- The script that generates the list of Haskell dependencies now also works on ghc8.0.1 and older cabal versions. Furthermore one can specify the build directory as a parameter if it is not located within the source directory. _(Armin Wurzinger)_

### 421.3.9 Interpreter Plugins

- The plugins Ruby, Python and Jni can now also be mounted as global plugin.
- Fix crashes in global Python plugin by using pluginprocess. Python plugin can now shutdown properly again. _(Markus Raab and Armin Wurzinger)_

### 421.3.10 JNI

- We now disable the plugin for the `BUILD_STATIC` or `BUILD_FULL` build variants, since otherwise the plugin breaks the `kdb` tool. _(René Schwaiger)_
- We disabled the internal check (`testscr_check_kdb_internal_check`) for the plugin, since it always fails. _(René Schwaiger)_

### 421.3.11 HexNumber

- The plugin `hexnumber` has been added. It can be used to convert hexadecimal values into decimal when read, and back to hexadecimal when written. _(Klemens Böswirth)_

### 421.3.12 List

- The `list plugin` now allows us to pass common configuration for all plugins by using keys below the "config/" setting. The updated plugin documentation contains more information and an example. _(Thomas Wahringer)_
- The `list plugin` which is responsible for global mounting had a bug which prevented globally mounted plugins from being configurable. _(Thomas Wahringer)_

### 421.3.13 mINI

- We fixed a memory leak in the `mINI plugin` by requiring the plugin `ccode` instead of the "provider" `code`. _(René Schwaiger)_
- Removed unused header files. _(René Schwaiger)_

### 421.3.14 network

- Fixed an error in network plugin that prevented it from working on non-glibc platforms. _(Lukas Winkler)_

### 421.3.15 Type

- We extended the `Markdown Shell Recorder` example inside the `README of the plugin`. _(René Schwaiger)_

### 421.3.16 Regex Dispatcher

- The plugin `regexdispatcher` has been added. It calculates regex representations for commonly used specification keywords to be used with the `typechecker`. Currently the keywords `check/range`, `check/enum` and `default` are supported. _(Armin Wurzinger)_

### 421.3.17 Typechecker

- The plugin `typechecker`, used to validate configuration specifications for Elektra statically, has been improved under the hood. It now supports a more concise and efficient type checking process including a greatly improved type inference scheme that should make generated specification files and thus generated errors to be easier to understand. An example of such error message is shown in the `README`. _(Armin Wurzinger)_

### 421.3.18 Tcl

- The `tcl` plugin does not fail anymore, if its configuration file does not exist and you try to retrieve the plugin contract. _(René Schwaiger)_
- The plugin now uses relative key names. This update addresses issue `#51`. _(René Schwaiger)_

### 421.3.19 YAJL

- The `YAJL Plugin` now uses the internal logger functionality instead of `printf` statements. _(René Schwaiger)_
- We fixed a problem with negative values reported by the `UndefinedBehaviorSanitizer`. _(René Schwaiger)_

### 421.3.20 YAML CPP

- The plugin does not save empty intermediate keys anymore. The example below shows the old and the new behavior of the plugin:

```
# Mount plugin
kdb mount config.yaml /tests/yamlcpp yamlcpp
# Store single key-value pair
kdb set /tests/yamlcpp/level1/level2/level3 value
# Old behavior
kdb ls /tests/yamlcpp
#> user/tests/yamlcpp/level1
#> user/tests/yamlcpp/level1/level2
#> user/tests/yamlcpp/level1/level2/level3
# New behavior
kdb ls /tests/yamlcpp
#> user/tests/yamlcpp/level1/level2/level3
```

. _(René Schwaiger)_

- YAML CPP now requires at least yaml-cpp 0.6, since the current MSR test for the plugin triggers two bugs:
  - https://github.com/jbeder/yaml-cpp/issues/247
  - https://github.com/jbeder/yaml-cpp/issues/289

  in earlier versions of the yaml-cpp library. _(René Schwaiger)_
- The plugin does now support arrays containing empty fields. _(René Schwaiger)_
- YAML CPP now also adds array metadata for arrays containing arrays. _(René Schwaiger)_
- The plugin now also supports empty arrays:

```
kdb mount test.yaml user/tests/yamlcpp yamlcpp
kdb setmeta user/tests/yamlcpp/array array ""
kdb export user/tests/yamlcpp/array yamlcpp
#> []
```

- YAML CPP now handles null values containing metadata properly:

```
kdb mount test.yaml user/tests/yamlcpp yamlcpp
kdb set user/tests/yamlcpp/null
kdb setmeta user/tests/yamlcpp/null comment 'Null Key'
kdb export user/tests/yamlcpp/null yamlcpp
#> !<!elektra/meta>
#> - ~
#> - comment: Null Key
```

### 421.3.21 YAML Smith

- YAML Smith is a plugin that converts Elektra's KeySet data structure to a textual representation in the YAML serialization format. The plugin is currently in a **very early stage of development**. Please be advised, that it is quite likely that the plugin will produce incorrect or even invalid YAML data, especially if your KeySet contains special characters.

### 421.3.22 Yan LR

- The experimental Yan LR plugin uses a parser, generated by ANTLR to read basic YAML data. The plugin only converts YAML data to Elektra's KeySet data structure. If you want to write data in the YAML format please take a look at the YAML Smith plugin. _(René Schwaiger)_

### 421.3.23 ZeroMQ transport plugins

- New notification transport plugins for ZeroMQ were added. The new "zeromqsend" and "zeromqrecv" plugins use ZMQ_PUB and ZMQ_SUB sockets to send and receive notifications. The plugins can be used instead or along with the "dbus" and "dbusrecv" transport plugins. Check out the plugin documentation for more information. _(Thomas Wahringer)_

### 421.3.24 Misc

- The logging plugins "syslog", "journald" and "logchange" now have a new option called "get" which can be enabled to log which configuration settings are loaded by applications. The new option can be used for logging application behavior when using notifications. _(Thomas Wahringer)_
- Do not exclude simpleini silently on non-glibc systems but output a message like for other plugins _(↩ Markus Raab)_
- We updated the infos/status clause of the following plugins:
  - boolean,
  - constants,
  - csvstorage,
  - hexnumber,

  - internalnotification,
  - ruby,
  - simpleini,
  - uname, and
  - xerces

. _(René Schwaiger)_

## 421.4 Libraries

### 421.4.1 General

- Replaced strdup with elektraStrDup (for C99 compatibility). _(Markus Raab)_
- You can now remove the basename of a key via the C++ API by calling key.delBaseName(). _(René Schwaiger)_
- The function elektraArrayGetNextKey now uses NULL instead of the empty string as init value for the returned key. _(René Schwaiger)_

### 421.4.2 pluginprocess

- The library pluginprocess that is used to execute plugins run inside own processes has been improved. This is useful as some plugins like Haskell-based plugins or python can only be started once inside a single process, while libelektra may call a plugin several times. The library now uses an improved communication protocol that separates between pluginprocess-related data and keysets passed to plugins. This avoids any possible name clashes between keys used by a plugin and keys used by pluginprocess. The documentation of the plugin has been improved as well, some mistakes were corrected and it should be more clear how to store plugin data besides pluginprocess's data structure. Tests have been added to the library to ensure its correct functionality. _(Armin Wurzinger)_
- Anonymous pipes are now used instead of named pipes for the communication as anonymous pipes get terminated by the OS in case a child process dies before writing back data to the parent. Currently the parent process will freeze otherwise attempting to read from the child. _(Armin Wurzinger)_

## 421.5 Bindings

- A new I/O binding for ev has been added. It can be used to integrate the notification feature with applications based on ev main loops. _(Thomas Wahringer)_

## 421.6 Notifications

- The notification API was extended. The API now supports more types: int, unsigned int, long, unsigned long, long long, unsinged long long, float and double. It also supports all of Elektra's kdb_*_t types defined in kdbtypes.h. Also contexts for callbacks were added and elektraNotificationRegisterCallbackSameOrBelow() allows for notifications for the registered key or below. _(Thomas Wahringer)_

## 421.7 Tools

- The new tool kdb find lists keys of the database matching a certain regular expression. _(Markus Raab)_
- You can now build the Qt-GUI using Qt 5.11. _(René Schwaiger)_

## 421.8 Scripts

- The script `check_formatting.sh` now also checks the formatting of CMake code if you installed `sponge` and `cmake-format`. _(René Schwaiger)_
- The script `check_formatting.sh` now no longer writes to stdout if clang-format5.0 can not be found. _(Lukas Winkler)_
- The script `check_bashisms.sh` should now work correctly again, if the system uses the GNU version `find`. _(René Schwaiger)_
- The script `reformat-cmake` now checks if `cmake-format` works before it reformats CMake files. Thank you to Klemens Böswirth for the `detailed description of the problem`. _(René Schwaiger)_
- `scripts/build/run_icheck` now no longer leaves the base directory of the project when checking if the ABI changed. _(Lukas Winkler)_
- The completion for `fish` now also suggest the `info/` meta attributes of the `file plugin`. _(René Schwaiger)_

### 421.8.1 Copy Template

- The script `copy-template` is now location independent. It will always create a new plugin in `src/plugins`. _(René Schwaiger)_
- The command now also supports the new `template for C++ based plugins`. Please use the command line switch `-p` to create a new plugin based on `cpptemplate`.

## 421.9 Documentation

- We improved the formatting of our compilation guide. _(René Schwaiger)_
- We fixed various minor spelling mistakes in the documentation. _(René Schwaiger)_
- The man pages for `kdb change-resolver-symlink` and `kdb change-storage-symlink` referenced the wrong command. _(Lukas Winkler, René Schwaiger)_
- We added documentation for our build system in `BUILDSERVER.md`. _(Lukas Winkler)_
- The documentation for `kdb` and `kdb set` now mentions the `--` argument that stops processing of command line switches. This is useful for setting negative values among other things. _(Klemens Böswirth)_
- We added a new tutorial about the jna binding. The tutorial shows how to use the java library to interact with kdb _(Michael Zronek)_
- GitHub now detects the license of the repository correctly again. _(René Schwaiger)_
- We added a tutorial describing Elektra's `array data type`. _(René Schwaiger)_

## 421.10 Tests

### 421.10.1 (Markdown) Shell Recorder

- We added new `Markdown Shell Recorder` tests for the
  - `ccode`,
  - `file`,
  - `iconv`,
  - `ni`,
  - `rename`, and

  - uname plugin. _(René Schwaiger)_

- (Markdown) Shell Recorder tests now save test data below /tests (see issue #1887). _(René Schwaiger)_
- The Markdown Shell Recorder checks kdb set commands to ensure we only add tests that store data below /tests. _(René Schwaiger)_
- The Markdown Shell Recorder now supports indented code blocks. _(René Schwaiger)_
- The Markdown Shell Recorder now also tests if a command prints nothing to stdout if you add the check #>. _(René Schwaiger)_
- We fixed some problems in the Markdown Shell Recorder test of kdb ls. _(René Schwaiger)_
- The Shell Recorder now does not interpret - in checks as option character any more. _(René Schwaiger)_
- The add_plugin helper now respects ENABLE_KDB_TESTING when adding Markdown Shell Recorder tests. _(Lukas Winkler)_
- The Markdown Shell Recorder test for kdb find now removes the configuration file at the end of the test. _(René Schwaiger)_
- The Shell Recorder now properly unmounts any additional mountpoints created during a test. _(René Schwaiger)_
- We removed the broken auto unmounting feature from the Markdown Shell Recorder. _(René Schwaiger)_
- The Markdown Shell Recorder does not require a bash compatible shell anymore. _(René Schwaiger)_

### 421.10.2 General

- Plugins added with the flag SHARED_ONLY no longer get tested in the script check_kdb_internal_↩ check.sh if executed with kdb-full or kdb-static. _(Armin Wurzinger)_
- Add compare_regex_to_line_files which allows to compare a file made of regex patterns to be compared with a text file line by line. _(Lukas Winkler)_
- The OPMPHM has a new test case _(Kurt Micheli)_
- Do not execute fcrypt and crypto unit tests if the gpg binary is not available. _(Peter Nirschl)_
- Resolved an issue where tests did not cleanup properly after they ran. This was especially noticeable for gpg tests as the gpg-agents that were spawned did not get cleaned up afterwards. _(Lukas Winkler)_
- We disabled the general plugin test (testkdb_allplugins) for the semlock plugin, since the test reported memory leaks on the latest version of Debian Unstable. _(René Schwaiger)_
- The CFramework macro compare_keyset now supports the comparison of two empty key sets. _↩ (René Schwaiger)_
- The C++ version of the macro exit_if_fail now really exits the test progamm if the test fails. _(René Schwaiger)_
- The C++ testing framework now supports the macro compare_keyset that checks if two key sets are equal. _(René Schwaiger)_

## 421.11 Build

As written in the previous release notes:

- Debian Wheezy is not supported anymore.
- Jessie (oldstable) with gcc 4.8.4 is now the oldest supported platform.

Another important change is:

- We now import the current version of Google Test as external project at configuration time using DownloadProject. If you want to use a local installation of Google Test instead, please set the value of GTEST_ROOT to the path of you local copy of the Google Test framework. _(René Schwaiger)_
- The CMake variable GTEST_ROOT now respects the environment variable GTEST_ROOT if it is set. _(Lukas Winkler)_

### 421.11.1 CMake

- The build system no longer installs Haskell dependencies from hackage by itself, instead this has to be done beforehand like it is the case with all other dependencies. The main reason is that the build servers shouldn't compile the dependencies over and over again, only if something changes. _(Armin Wurzinger)_
- Plugins can be specified to be only built for BUILD_SHARED builds, but to be excluded from any BUILD_FULL or BUILD_STATIC builds using the new optional argument ONLY_SHARED for our CMake macro add_plugin. This way BUILD_SHARED can be combined with the other options without excluding such plugins. The CMake messages about plugin inclusion have been updated to indicate this behavior. This behavior has been applied for the Haskell plugins- and bindings and JNI plugin as they currently don't support full or static builds. _(Armin Wurzinger)_
- The build system does not install Google Test anymore if you install Elektra. _(René Schwaiger)_
- We disabled the test testlib_notification on ASAN enabled builds, since Clang reports that the test leaks memory. _(René Schwaiger)_
- Disable Markdown Shell Recorder test validation.md for ASAN builds. It leaks memory and thus fails the test during spec mount. _(Lukas Winkler)_
- Haskell plugins and bindings are now correctly excluded when using BUILD_FULL or BUILD_STATIC as this is currently unsupported. Another issue when building Haskell plugins with a cached sandbox is fixed as well. _(Armin Wurzinger)_
- Fix compilation with BUILD_TESTING=OFF when spec or list plugins are not selected.
- Set coverage prefix to PROJECT_SOURCE_DIR, resulting in easier readable coverage reports. _(Lukas Winkler)_
- The functions add_plugintest and add_plugin now also support adding a C++ test instead of a C test. _(René Schwaiger)_
- The function add_plugintest now also supports setting environment variables for C/C++ based tests. _(René Schwaiger)_
- The build system now automatically detects Homebrew's OpenSSL version on macOS. _(René Schwaiger)_
- We improved the automatic detection of Libgcrypt and OpenSSL. _(René Schwaiger)_
- Resolved an issue where CMake did not properly set test feature macros to detect and use libc functionality. _(Lukas Winkler)_
- Improve the detection of ftw.h, if the current build use the compiler switch -Werror. _(René Schwaiger)_
- We now ignore warnings about

  - zero size arrays (Clang),
  - variadic macros (Clang, GCC),
  - conversions to non-pointer type (GCC), and
  - attribute warnings (GCC),

  caused by code generated via `SWIG` in the Ruby binding and plugin. _(René Schwaiger)_

### 421.11.2 Docker

- `clang-5.0` is now used for clang tests by the build system _(Lukas Winkler)_
- An additional build job on Ubuntu:xenial has been added _(Lukas Winkler)_
- `withDockerEnv` Jenkinsfile helper now no longer provides stages automatically. _(Lukas Winkler)_
- `Google Test` is installed in Docker images used by the build system. _(Lukas Winkler)_

## 421.12 Infrastructure

### 421.12.1 Jenkins

- A build job checks if PRs modify the release notes. _(Markus Raab)_
- Several improvements to the build system have been implemented _(Lukas Winkler)_:
  - Better Docker image handling.
  - Abort of previously queued but unfinished runs on new commits.
  - Document how to locally replicate the Docker environment used for tests.
- The Jenkins build server now also compiles and tests Elektra with enabled address sanitizer. _(Lukas Winkler)_
- Add `STATIC` and `FULL` linked builds. _(Lukas Winkler)_
- Ported GCC ASAN build job to new build system _(René Schwaiger + Lukas Winkler)_
- Docker artifacts are now cleaned up in our daily build job. _(Lukas Winkler)_
- `clang` tests have been ported to the new build system _(Lukas Winkler et al)_
- `icheck` build server job has been ported to our new build system. _(Lukas Winkler)_
- Port `elektra-gcc-configure-debian-optimizations` to new build system. _(Lukas Winkler)_
- Port `elektra-gcc-configure-mingw-w64` to new build system. _(Lukas Winkler)_
- Port `debian-multiconfig-gcc-stable` to new build system. _(Lukas Winkler)_
- Port `elektra-ini-mergerequests` to new build system. _(Lukas Winkler)_
- Port `elektra-gcc-configure-debian-nokdbtest` to new build system. _(Lukas Winkler)_
- Port `elektra-gcc-configure-xdg`to new build system. _(Lukas Winkler)_
- Port `elektra-gcc-i386` to new build system. _(Lukas Winkler)_
- Port `elektra-gcc-configure-debian-musl` to new build system. _(Lukas Winkler)_
- Docker Registry is cleaned up by our daily buildserver task. _(Lukas Winkler)_
- Remove `elektra-gcc-configure-debian-nokdbtest` test. Instead we are now removing write permissions of Elektra's paths to detect if we write to the filesystem even though tests are not tagged as such. _(Lukas Winkler)_

- Remove `elektra-gcc-configure-debian-withspace` test. We now test for compatibility of spaced build paths during normal tests. _(Lukas Winkler)_
- Check for source formatting during early test stages. _(Lukas Winkler)_
- Remove the amount of spawned tests via not running a full multiconfig setup for the `PLUGINS=NODEP` config. They did not provide any additional coverage. Instead we added a new test checking if `PLUGINS=NODEP` builds in an minimal Docker image. _(Lukas Winkler)_
- Speed up coverage data upload. _(Lukas Winkler)_
- Fix an issue where file archiving did not happen because of suppressed shell expansion _(Lukas Winkler)_
- Setup mailing for jenkins _(Lukas Winkler)_
  - send mail to `build@libelektra.org` when `master` fails _(Lukas Winkler)_
  - parse change list into mail _(Lukas Winkler)_
  - do not send mails if pipeline run was aborted _(Lukas Winkler)_

### 421.12.2 Travis

- Travis now uses the latest version of GCC and Clang to translate Elektra on Linux. _(René Schwaiger)_
- Our Travis build job now
  - builds all (applicable) bindings by default again, and
  - checks the formatting of CMake code via `cmake-format`. _(René Schwaiger)_
- Some cache issues on the Travis build job for cached Haskell sandboxes have been resolved. _(Armin Wurzinger)_
- Travis caches downloaded Homebrew packages to improve the reliability of macOS build jobs. _(René Schwaiger)_
- Travis is now using Xcode 9.4.1 on macOS 10.13 for most macOS build jobs. _(Mihael Pranjić)_
- We added a unique name to each build job, so you can see quickly which configuration caused problems. _(René Schwaiger)_
- We now specify custom binding, plugin and tool configuration for jobs via the environment variables:
  - `BINDINGS`,
  - `PLUGINS`, and
  - `TOOLS`

  . We also added environment variables for the build configuration options `BUILD_FULL`, `COMMON_FLAGS`, `ENABLE_ASAN` and the command used to test the build (`TEST_COMMAND`). _(René Schwaiger)_
- The ASAN build jobs `green apple Clang ASAN` and `penguin GCC ASAN` now only build the `kdb` tool and the `cpp` binding. This update ensures, that we do not hit the `job timeout for public repositories` that often. _(René Schwaiger)_
- We now use the latest version of Ruby (`2.5.1`) to build and test the Ruby binding/plugin. _(René Schwaiger)_

## 421.13 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
Following changes were made:

- The C++ API was extended with `delBaseName()`. This does not affect ABI compatibility, also C++ programs compiled against 0.8.24 and using `delBaseName()` will work with Elektra 0.8.23 or older.
- `kdbtypes.h` now comes with support for C99 types.
- We added the private headerfiles `kdbnotificationinternal.h`, `kdbioplugin.h`. _(Thomas Wahringer)_
- The I/O binding header files have been moved a new directory called `kdbio`. For example, instead of including `elektra/kdbio_ev.h` users of the binding now include `elektra/kdbio/ev.h`. _(Thomas Wahringer)_
- The plugin directoryvalue has changed its behavior, see above.
- The plugin list changed its configuration, see above.
- The plugin yamlcpp now gets excluded with too old versions of yamlcpp (Debian Stretch is affected).

The new plugins are:

- hexnumber
- yamlsmith
- zeromqrecv
- zeromqsend

The new tool is: kdb-find

## 421.14 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date.

## 421.15 Outlook

We are currently working on following topics:

- The hybrid search algorithm for the Key search `ksLookup (...)` is now in preparation. The preparation includes a new KeySet flag `KS_FLAG_NAME_CHANGE`, this flag will be used by the hybrid search. The hybrid search combines the best properties of the binary search and the `OPMPHM`. The hybrid search uses a modified branch predictor to predicts KeySet changes and decides if binary search or OPMPHM would be faster. _(Kurt Micheli)_

## 421.16 Statistics

Following persons made in total 1734 commits:

| Commits | Author |
|---|---|
| 1 | Mihael Pranjic `mpranj@limun.org` |
| 2 | Thomas Waser `thomas.waser@libelektra.org` |
| 7 | Michael Zronek `michael.zronek@gmail.com` |
| 12 | Kurt Micheli `e1026558@student.tuwien.ac.at` |

| Commits | Author |
|---|---|
| 17 | Peter Nirschl peter.nirschl@gmail.com |
| 21 | Klemens Böswirth k.boeswirth+git@gmail.com |
| 197 | Markus Raab elektra@markus-raab.org |
| 102 | Thomas Wahringer thomas.wahringer@libelektra.org |
| 117 | Daniel Bugl me@omnidan.net |
| 265 | Lukas Winkler derwinlu+git@gmail.com |
| 249 | Armin Wurzinger e1528532@student.tuwien.ac.at |
| 744 | René Schwaiger sanssecours@me.com |

In total there were 792 files changed with 27677 insertions(+) and 39176 deletions(-).

## 421.17 Get It!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.8.24.tar.gz
- size: 6130464
- md5sum: 2e3def7b905f94e1f9f7fa0fe4743189
- sha1: ff2a9b2d3a5e20a456e272a47fe9fd79ad410428
- sha256: 454763dd00e95e774a907b26eb59b139cfc59e733692b3cfe37735486d6c4d1d

The release tarball is also available signed by Markus Raab using GnuPG from here or on GitHub
Already built API documentation can be found online or GitHub.

## 421.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
For any questions and comments, please contact the issue tracker on GitHub or Markus Raab by email using elektra@markus-raab.org.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 422

# 0.8.25 Release

We are proud to present Elektra 0.8.25.

## 422.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
For a small demo see here:

You can also read the news `on our website`

## 422.2 Metadata

- guid: 472392e0-cc4f-4826-a0a9-2764d90c5f89
- author: Markus Raab
- pubDate: Sun, 18 Nov 2018 21:24:34 +0100
- shortDesc: faster storage and lookup

## 422.3 Highlight

This release is dedicated to drastically improve the speed of Elektra. Two non-trivial features facilitate most of the improvement:

- mmap storage for very fast retrieval of configuration
- Hybrid Search Algorithm for `ksLookup (...)` for very fast access of configuration

### 422.3.1 mmap storage

We added a new, binary and fast storage plugin called `mmapstorage`. It leverages the `mmap()` syscall and supports full Elektra semantics. We provide two compile variants: `mmapstorage` and `mmapstorage_crc`. The `mmapstorage_crc` variant enables CRC32 checksums for critical data, while the `mmapstorage` variant omits the checksum for maximum performance.
We ran a synthetic benchmark with 257 iterations using 40k keys in a keyset, and compared the performance to the `dump` storage plugin.
Median write time in microseconds:

| Plugin | Time |
|---|---|
| `dump` | 71079 |
| `mmapstorage` | 2964 |
| `mmapstorage_crc` | 7644 |

Median read time in microseconds:

| Plugin | Time |
|---|---|
| dump | 82737 |
| mmapstorage | 1145 |
| mmapstorage_crc | 5744 |

In our benchmark, the mmapstorage plugin writes more than 23x faster, and reads more than 72x faster than the dump storage plugin.
For this release the plugin is marked as experimental, even though it is already used as default storage plugin in a build job on our build server.
Thanks to Mihael Pranjić for this improvement.

### 422.3.2 Hybrid Search Algorithm for <tt>ksLookup (...)</tt>

The hybrid search algorithm is now implemented, this concludes the extension of the ksLookup (...) search with the order preserving minimal perfect hash map (OPMPHM). The hybrid search combines the best properties of the binary search and the OPMPHM. The hybrid search decides dynamically which search algorithm to use.
Because of the automatic decision, usually nothing needs to be done by API users to take advantage of this improvement. Advanced API user, however, can overrule the hybrid search by passing KDB_O_OPMPHM or KDB_O_BINSEARCH to ksLookup (...). The constants are defined in kdbproposal.h. For low-memory systems the building of the hash map can be disabled altogether at build-time by disabling the CMake variable ENABLE_OPTIMIZATIONS (by default enabled now).
The implemented randomized OPMPHM algorithm is in 99.5% of the measured random cases optimal. However the randomization property of the algorithm leaves an uncertainty.
The results made with random cases had shown that the hybrid search is, except for small keyset sizes, almost always faster compared to the standalone binary search. The performance increase strongly depended on the measured hardware. In the random cases where the hybrid search is faster, on average ~8.53% to ~20.92% of time was saved. The implemented hybrid search works only above a keyset size of 599 to exclude the small keyset sizes.
Thanks to Kurt Micheli for this improvement.

## 422.4 Plugins

The following section lists news about the plugins we updated in this release.

### 422.4.1 Directory Value

We improved the performance of the directoryvalue plugin. _(René Schwaiger)_ This plugin is used for configuration file formats that do not support that directories contain values, like it is the case in JSON. A program manipulating a 13 MB JSON file which first did not succeed within 10 hours is now finished in 44 seconds.

### 422.4.2 Process

There is a new, experimental plugin called process. This plugin utilizes the pluginprocess library in order to execute arbitrary other plugins in an own process, acting as a proxy itself. Therefore it is not required to explicitly change a plugin's implementation if it shall be executed in an own process. This plugin is not completely finished yet, as currently there is no way for it to mimic the proxied plugin's contract in Elektra. It can be used with simple plugins like dump however, check the limitations in the readme for more details. _(Armin Wurzinger)_

### 422.4.3 FSTab

The detection of the mntent functions now also works correctly, if you use the compiler switch -Werror. _(René Schwaiger)_

### 422.4.4 passwd

We fixed an issue with the passwd plugin not properly setting compile flags. This resolves a problem with undefined functions when building with musl. _(Lukas Winkler)_

### 422.4.5 gpgme

The experimental `gpgme plugin` was brought into existence to provide cryptographic functions using GnuGP via the `libgpgme` library. _(Peter Nirschl)_

### 422.4.6 network

The `network` plugin now also allows for non-numerical hosts (i.e. "localhost") to be set and tries to resolve it via DNS. _(Michael Zronek)_

### 422.4.7 YAMBi

This new plugin parses a subset of YAML using a parser generated by `Bison`. _(René Schwaiger)_

### 422.4.8 YAML CPP

The build system now disables the plugin automatically, if you use a GCC compiler (`6.x` or earlier) and enable the option `ENABLE_ASAN`. We updated the behavior, since otherwise the plugin will report memory leaks at runtime. _(René Schwaiger)_

### 422.4.9 Yan LR

- The plugin does not modify the (original) parent key. As a consequence, setting values at the root of a mountpoint:

```
sudo kdb mount config.yaml user/tests/yambi yambi
kdb set user/tests/yanlr 'Mount Point Value'
kdb get user/tests/yanlr
#> Mount Point Value
```

now works correctly. _(René Schwaiger)_

- We now use C++ code to test the plugin. _(René Schwaiger)_
- The CMake code of the plugin now also recognizes `antlr` as ANTLR executable, if `antlr4` is not available. _(René Schwaiger)_
- The build system now disables the unit test for the plugin, if you use GCC (`6.x` or earlier) to translate Elektra. We introduced this behavior, since the code generated by ANTLR (`YAML.h`) seems to contain a double free that causes a segmentation fault on systems that use the GNU C library. _(René Schwaiger)_
- The build system now disables the plugin automatically, if you use a GCC compiler (`6.x` or earlier) and enable the option `ENABLE_ASAN`. _(René Schwaiger)_

### 422.4.10 YAwn

This new plugin parses a subset of YAML using the Earley Parser library YAEP. _(René Schwaiger)_

### 422.4.11 Reference

This new plugin can be used to validate that the value of a key is a reference to another key. _(Klemens Böswirth)_

## 422.5 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 422.5.1 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
This is the last release for which we have built Jessie packages:

```
deb     [trusted=yes] https://debian-stable.libelektra.org/elektra-stable/ jessie main
deb-src [trusted=yes] https://debian-stable.libelektra.org/elektra-stable/ jessie main
```

Obviously, we will continue to update the stretch package:

```
deb     [trusted=yes] https://debian-stretch-repo.libelektra.org/ stretch main
deb-src [trusted=yes] https://debian-stretch-repo.libelektra.org/ stretch main
```

### 422.5.2 Infos for Package Maintainers

Following plugins got added:

- libelektra-gpgme.so
- libelektra-mmapstorage_crc.so
- libelektra-mmapstorage.so
- libelektra-process.so
- libelektra-reference.so
- libelektra-yambi.so

A new library got added (should be packaged privately for now):

- libelektra-globbing.so

### 422.5.3 Core

Optimize elektraKsFilter to not duplicate keys _(Markus Raab)_

### 422.5.4 Globbing

A new library which can be used to match keys against globbing patterns was introduced. _(Klemens Böswirth)_
The API is still experimental, so it should not be used externally for now.

### 422.5.5 Ease

`libease` provides the function `elektraArrayValidateBaseNameString`, which can be used to validate that a given string is an Elektra array name. _(Klemens Böswirth)_

## 422.6 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 422.6.1 Ruby

Do not use private Elektra headers for Ruby bindings as preparation for a Ruby `libelektra` gem. _(Bernhard Denner)_

## 422.7 Tools

- Added benchmarks for storage plugins. The currently benchmarked plugins are `dump` and `mmapstorage`. _(Mihael Pranjić)_
- Avoid raw pointers in KDB tools. _(Markus Raab)_

- Improved error text of KDB tool `cp`. _(Markus Raab)_
- Document hidden feature of KDB tool `mount`. _(Markus Raab)_
- Add more tags to the KDB tools to be used with `kdb find-tools`. _(Markus Raab)_

## 422.8 Scripts

- We now require `clang-format` 6.0 for formatting C and C++ code. _(René Schwaiger)_
- The command `reformat-source` now displays information about the installed version of `clang-format`, if it is unable to locate a supported version of the tool. _(René Schwaiger)_
- We now also check the POSIX compatibility of our scripts with `shfmt`. _(René Schwaiger)_
- The new command `reformat-shfmt` reformats Shell scripts using the tool `shfmt`. _(René Schwaiger)_

## 422.9 Documentation

- We fixed some minor spelling mistakes in the documentation. _(René Schwaiger)_
- Improved the plugins documentation. _(Michael Zronek)_
- The ReadMe now includes two badges that show the latest released version of Elektra and the status of the Travis build. _(René Schwaiger)_
- Fixed documenation error on Ruby plugin Readme. _(Bernhard Denner)_
- Go into more detail in `BUILDSERVER.md`. _(Lukas Winkler)_

## 422.10 Tests

- Fix potential parallel execution of maven tests, which write to KDB. _(Markus Raab)_
- The unit test for the `dbus plugin` does not leak memory anymore, if it fails on macOS. _(Thomas Wahringer)_
- The tests `testkdb_allplugins` and `testscr_check_kdb_internal_check` do not test a plugin on an ASAN enabled build anymore, if you specify the status tag `memleak` in the plugin contract. _(René Schwaiger)_
- The `CFramework` macro `compare_key` now also checks if the meta values of keys are equal. _(René Schwaiger)_
- The test `testscr_check_bashisms` does not print warnings about skipped files anymore. _(René Schwaiger)_
- We added a `Markdown Shell Recorder` test for the `shell plugin`. _(René Schwaiger)_
- Added many storage plugin tests. Most tests use the keyset, key name and value APIs. Currently, the tests are only active for `dump` and `mmapstorage`. _(Mihael Pranjić)_
- The test `testcpp_contextual_basic` now compiles without warnings, if we use Clang 7 as compiler. _(René Schwaiger)_
- crypto, fcrypt and gpgme properly shut down the gpg-agent after the unit test is done. See #1973 . _(Peter Nirschl)_
- minor refactoring of the unit tests for crypto, fcrypt, gpgme: moved shared code to separate module in order to avoid code duplication. _(Peter Nirschl)_

- The CMake targets for plugin tests (`testmod_[plugin]`) now depend on the respective CMake targets for the plugins themselves (`elektra-[plugin]`). _(Klemens Böswirth)_
- Fixed bug in CMake plugin tests, if only `BUILD_FULL` but not `BUILD_SHARED` is used. _(Klemens Böswirth)_
- The test `testscr_check_formatting` now also checks the formatting of Shell code. _(René Schwaiger)_
- We pumped version numbers in XML-test files. _(Markus Raab)_
- We fixed a crash in the unit test of the `JNA` binding. _(René Schwaiger)_
- The command `kdb run_all` now only prints the output of tests that failed. To print the full output of all test, please use the option `-v`. _(René Schwaiger)_
- The `Shell Recorder` does not use the non-POSIX grep option `--text` any more. _(René Schwaiger)_
- The test suite now uses `Google Test 1.8.1`. _(René Schwaiger)_

## 422.11 Build

### 422.11.1 CMake

- We improved the detection of Python 2 and Python 3 in the CMake code of the Python bindings/plugins. _(René Schwaiger)_
- We restructured the code of the CMake module we use to detect Haskell tools . _(René Schwaiger)_
- Building the Haskell binding should now work again. _(René Schwaiger)_
- The CMake configuration step now displays less debug messages about found libraries. _(René Schwaiger)_
- Provide a wrapper around `check_symbol_exists` that handles issues with `-Werror -Wpedantic`. _(Lukas Winkler)_
- The argument `INCLUDE_SYSTEM_DIRECTORIES` of the function `add_plugin` now supports multiple include directories. _(René Schwaiger)_
- We reformatted all CMake source files with cmake-format 0.4.3. _(René Schwaiger)_
- Generating coverage data (`ENABLE_COVERAGE=ON`) should now also work on macOS. _(René Schwaiger)_
- You can use the new target `run_checkshell` to run all shell checks (`testscr_check.*`). _(René Schwaiger)_
- The new target `run_nocheckshell` runs all tests except for shell checks. _(René Schwaiger)_
- The target `run_all` now runs tests that do not modify the key database in parallel. _(René Schwaiger)_
- Fix CMake inclusion logic for GLib/Gi _(Markus Raab)_

### 422.11.2 Docker

- The Docker image for Debian stretch now contains all (optional) dependencies for Elektra. _(René Schwaiger)_

- The docker images used by our build system are now available to download from our systems without authentication. Try it out and list available images via `docker run --rm anoxis/registry-cli -r https://hub-public.libelektra.org`. You can search for images using `--images-like`, for example: `docker run --rm anoxis/registry-cli -r https://hub-public.libelektra.org --images-like alpine`. Afterwards pull your desired image as you would do from any public registry, for example: `docker pull hub-public.libelektra.org/build-elektra-alpine:201811-37597a34fed4988639cdaf4d6a2c54754d09918586f53389e4fc` _(Lukas Winkler)_

### 422.11.3 Vagrant

- Added Vagrantfile for Ubuntu artful 32-bit. _(Mihael Pranjić)_

## 422.12 Infrastructure

### 422.12.1 Jenkins

- We enabled tests that write to the hard disk on the build job `alpine`. _(René Schwaiger)_
- The build jobs now print less non-relevant output. _(René Schwaiger)_
- Enable `-Werror` in `debian-stable-full`. _(Lukas Winkler)_
- We added the compiler switch `-Werror` to the build jobs:
  - `alpine`,
  - `debian-stable-full-i386`,
  - `debian-stable-full-mmap-asan`,
  - `debian-stable-full-mmap`,
  - `debian-stable-full-optimizations-off`,
  - `debian-stable-full-xdg`,
  - `debian-stable-minimal`,
  - `debian-stable-multiconf`,
  - `debian-unstable-clang-asan`,
  - `debian-unstable-full-clang`,
  - `debian-unstable-full`,
  - `ubuntu-xenial`, and
  - `debian-stable-asan`. _(René Schwaiger)_
- Build Artifacts for all PR's to detect issues before merging _(Lukas Winkler)_
- Stricter removal of temporary docker images on docker nodes _(Lukas Winkler)_
- Added jenkins build jobs `debian-stable-full-mmap` and `debian-stable-full-mmap-asan` with `mmapstorage` as the default storage. _(Mihael Pranjić)_
- We added basic support for coverage analysis via `Coveralls`. _(René Schwaiger)_

### 422.12.2 Travis

- Travis now also checks the code for memory leaks in the build job `green apple Clang ASAN`. _(René Schwaiger)_
- The Travis build jobs `green apple Clang ASAN` and `penguin GCC ASAN` now only translates a minimal set of plugins, since we had various timeout problems with these jobs before. We explicitly excluded plugins, to make sure that the build jobs still test newly added plugins. _(René Schwaiger)_

- Added travis build job `green apple mmap` on macOS with `mmapstorage` as the default storage. _(Mihael Pranjić)_
- Travis now prints the CMake configuration for each build job. _(René Schwaiger)_
- We now test Elektra using the latest version of Xcode (`10.0`). _(René Schwaiger)_
- We added the build job `green apple Check Shell`, which only runs shell checks such as `testscr_check_oclint`. This update allows us to remove the shell checks from the jobs `green apple MMap` and `green apple Clang`, which sometimes hit the `timeout limit for public repositories` before. _(René Schwaiger)_
- All Travis build jobs now use the compiler switch `-Werror`. _(René Schwaiger)_
- The new job `green apple FULL` and the build job `penguin FULL` build Elektra using the CMake options `BUILD_FULL=ON` and `BUILD_SHARED=OFF`. _(René Schwaiger)_
- The `script` stage of the build jobs print less non-relevant output. Usually the commands in this stage should now only print verbose output if a test fails. _(René Schwaiger)_

## 422.13 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date.

## 422.14 Outlook

We are currently working on following topics:

- Global mmap cache: This feature will enable Elektra to return configuration without parsing configuration files or executing other plugins as long as the configuration files are not changed. _(Mihael Pranjić)_
- Finish high-level API. _(Klemens Böswirth)_
- Validation improvements. _(Michael Zronek)_
- Improve YAML plugins. _(René Schwaiger)_

## 422.15 Statistics

Following authors made this release possible:

| Commits | Author |
|---|---|
| 1 | Thomas Wahringer thomas.wahringer@libelektra.org |
| 2 | Bernhard Denner bernhard.denner@gmail.com |
| 7 | Kurt Micheli e1026558@student.tuwien.ac.at |
| 12 | Michael Zronek michael.zronek@gmail.com |
| 33 | Lukas Winkler derwinlu+git@gmail.com |
| 28 | Klemens Böswirth k.boeswirth+git@gmail.com |
| 30 | Armin Wurzinger e1528532@student.tuwien.ac.at |
| 38 | Peter Nirschl peter.nirschl@gmail.com |
| 100 | Markus Raab markus@libelektra.org |
| 180 | Mihael Pranjic mpranj@limun.org |
| 418 | René Schwaiger sanssecours@me.com |

849 commits, 581 files changed, 18503 insertions(+), 3192 deletions(-)
We welcome new contributors!

## 422.16 Finished Thesis

- Daniel Bugl, see also the web demo.
- Thomas Wahringer.
- Kurt Micheli
- Armin Wurzinger

## 422.17 Get It!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.8.25.tar.gz
- size: 6233918
- md5sum: d5614b2049fb8431a80842a4909b140e
- sha1: c7dfb5fa87284d8f5ba4d4753e0e47a0e362c733
- sha256: 37829256e102e967fe3d58613a036d9fb9b8f9658e20c23fa787eac0bfbb8a79

The release tarball is also available signed by Markus Raab using GnuPG from here or GitHub
Already built API documentation can be found here or GitHub.

## 422.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via the issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 423

# 0.8.26 Release

- guid: 55950e64-fa4e-4eb9-9a3a-2c73d9cd6478
- author: Markus Raab
- pubDate: Tue, 26 Feb 2019 15:31:09 +0100
- shortDesc: high-level API

We are proud to release Elektra 0.8.26 with the new high-level API.

## 423.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit https://libelektra.org.
You can also read the news on our website

## 423.2 High-Level API

The new high-level API provides an easier way for applications to get started with Elektra.
To use Elektra in an application (including proper error handling) you now only need a few self-explanatory lines of code:

```
ElektraError * error = NULL;
Elektra * elektra = elektraOpen ("/sw/org/myapp/#0/current", NULL, &error);
if (elektra == NULL)
{
        printf ("An error occurred: %s", elektraErrorDescription (error));
        elektraErrorReset (&error);
        return -1;
}
// Once you have an instance of `Elektra` you simply call one of the typed `elektraGet*` functions to read a
      value:
kdb_long_t mylong = elektraGetLong (elektra, "mylong");
printf ("got long " ELEKTRA_LONG_F "\n", mylong);
const char * mystring = elektraGetString (elektra, "mystring");
printf ("got string %s\n", mystring);
elektraClose (elektra);
```

To run the application, the configuration should be specified, e.g., for mylong:

```
sudo kdb setmeta /sw/org/myapp/#0/current/mylong type long
sudo kdb setmeta /sw/org/myapp/#0/current/mylong default 5
```

In the getters/setters there is no need to convert types or to specify the base path `/sw/org/myapp/#0/current`, as the high-level API does that for you. The API supports the CORBA types already used by the plugins. The high-level API should also be used in combination with a specification (`spec-mount`). When used this way, the API is designed to be error and crash free while reading values. Writing values, can of course still produce errors.
Another advantage of the new API is, that it will be much easier to write bindings for other languages now, because only a few simple types and functions have to be mapped to provide the full functionality.
Take a look at the README for more information.
Because of the high-level API, we now have the new header files `elektra.h` and a folder `elektra` in Elektra's include directory. Furthermore, we install the library `libelektra-highlevel.so` and the pkgconfig file `elektra-highlevel.pc` for easier detection.

For an example on how to build an application using this API take a look at this.
A big thanks to *Klemens Böswirth* for making this possible. Also big thanks to *Dominik Hofer*, who did all the foundation work for this API.

## 423.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 423.3.1 Augeas

- We changed the default `Augeas` directory prefix for the lenses directory on macOS to the one used by `Homebrew`: `/usr/local`. _(René Schwaiger)_

### 423.3.2 Network

- The `network` plugin also supports port declarations to check if a port number is valid or if the port is available to use. _(Michael Zronek)_
- We added a `Markdown Shell Recorder` test to the `ReadMe of the plugin`. _(René Schwaiger)_

### 423.3.3 YAMBi

- The build system does not print a warning about a deprecated directive any more, if you build the plugin with Bison `3.3` or later. _(René Schwaiger)_
- YAMBi now handles comments at the end of input properly. _(René Schwaiger)_

### 423.3.4 YanLR

- We improved the error reporting capabilities of the plugin. It now stores all of the error message reported by ANTLR and also specifies the line and column number of syntax errors. We also visualize these error messages in a similar way as modern compilers like Clang or GCC. For example, for the following erroneous input:

```
key: - element 1
- element 2 # Incorrect Indentation!
```

the plugin currently prints an error message that looks like this:

```
config.yaml:2:1: mismatched input '- ' expecting end of map
                 - element 2 # Incorrect Indentation!
                 ^^
config.yaml:2:37: extraneous input 'end of map' expecting end of document
                  - element 2 # Incorrect Indentation!
                                                      ^
```

. The inspiration for this feature was taken from the book "The Definitive ANTLR 4 Reference" by Terence Parr. _(René Schwaiger)_

- Yan LR's lexer now
  - handles comment at the end of a YAML document correctly,
  - stores a more human-readable description in tokens (e.g. `end of map` instead of `MAP END`)

  . _(René Schwaiger)_

### 423.3.5 Path

Enhanced the plugin to also check for concrete file or directory permissions such as `rwx`. For example, you can specify that a user can write to a certain directory or file which prevents applications of runtime failures once they try to access the given path (such as a log directory or file). Simply add `check/path/user <user>` and `check/path/mode <modes>` as specification (metadata) and be assured that you can safely set a path value to the key. A more detailed explanation can be found here _(Michael Zronek)_

### 423.3.6 YAwn

- The plugin now handles comments at the end of a file properly. _(René Schwaiger)_
- We improved the syntax error messages of the plugin. _(René Schwaiger)_
- We fixed a memory leak that occurred, if a YAML file contained syntax errors. _(René Schwaiger)_

### 423.3.7 YAy PEG

- The new plugin YAy PEG parses a subset of YAML using a parser based on PEGTL. _(René Schwaiger)_

### 423.3.8 Ruby

- Added some basic unit tests _(Bernhard Denner)_

### 423.3.9 Misc

- We fixed some compiler warnings for the plugins
  - `camel`,
  - `line`,
  - `mini` and
  - `resolver`

  reported on FreeBSD. _(René Schwaiger)_
- The `resolver` plugin and its tests now better support `KDB_DB_SYSTEM` and `KDB_DB_SPEC` paths using ~ to refer to a home directory. _(Klemens Böswirth)_
- If `KDB_DB_SYSTEM` is set to a relative path, it is now treated as relative to `CMAKE_INSTALL_PREFIX`. This ensures that `KDB_DB_SYSTEM` actually points to the same location no matter the current working directory. _(Klemens Böswirth)_

## 423.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 423.4.1 Compatibility

As always, the ABI and API of kdb.h is fully compatible, i.e. programs compiled against an older 0.8 version of Elektra will continue to work (ABI) and you will be able to recompile programs without errors (API).
We have two minor incompatible changes:

- we now support larger array numbers (i.e. the larger numbers are not an error anymore)
- `elektraArrayValidateBaseNameString` returns the offset to the first digit of the array index instead of `1`

For details of the changes see below `Core` and `Libease`.

### 423.4.2 Core

- All plugins in the KDB now get a handle to a global keyset via `elektraPluginGetGlobalKeySet()`, for communication between plugins. See Global KeySet Handle for details. _(Mihael Pranjić)_
- `elektraWriteArrayNumber` now uses `kdb_long_long_t` for array indices to be compatible with the high level API. Similarly the value of `ELEKTRA_MAX_ARRAY_SIZE` was changed to match this. _(Klemens Böswirth)_

### 423.4.3 Libease

- The function `elektraArrayValidateBaseNameString` now returns the offset to the first digit of the array index, if the given string represents an array element containing an index. This update enhances the behavior of the function. Now it not only tells you if a name represents a valid array element, but also the start position of the array index.

```
elektraArrayValidateBaseNameString ("#_10");
//                                   ~~^ Returns '2' (instead of '1')
elektraArrayValidateBaseNameString ("#___1337");
//                                  ~~~~^ Returns '4' (instead of '1')
```

If your program already used `elektraArrayValidateBaseNameString` and you check for a valid array element using the equality operator (`== 1`), then please use (`>= 1`) instead. For example, if you code that looks like this:

```
if (elektraArrayValidateBaseNameString(baseName) == 1) ...;
```

, please update your code to check for a valid array element name like this:

```
if (elektraArrayValidateBaseNameString(baseName) >= 1) ...;
```

. _(René Schwaiger)_

### 423.4.4 Libopts

- This is a new library containing only the function `elektraGetOpts`. This function can be used to parse command line arguments and environment variables and add their values to keys in the proc namespace.

  You can use `opt`, `opt/long` and `env` to specify a short, a long option and an environment variable. For more information take a look at the tutorial and the code documentation of `elektraGetOpts`. _(Klemens Böswirth)_

## 423.5 Tools

- `kdb spec-mount` correctly includes type plugin to validate `type`. _(Markus Raab)_
- `kdb setmeta` reports if it removed a metakey. _(Markus Raab)_
- `system/elektra/version` now has metadata to indicate that it cannot be edited or removed. _(Dominic Jäger)_

## 423.6 Scripts

- The script `reformat-source` now also handles filenames containing spaces correctly. _(René Schwaiger)_
- The script `reformat-markdown` formats `Markdown` files in the repository with `prettier`. _(René Schwaiger)_
- The scripts `reformat-source`, `reformat-cmake`, `reformat-shfmt` and `reformat-markdown` don't format files that are ignored by Git anymore. _(Klemens Böswirth)_

## 423.7 Documentation

- We fixed various spelling mistakes. _(René Schwaiger)_
- The documentation for `elektraMetaArrayToKS` was fixed. It now reflects the fact that the parent key is returned as well. _(Klemens Böswirth)_

## 423.8 Tests

- The tests for the IO bindings and notification plugins now use increased timeout values so that the test suite fails less often on machines with high load. _(René Schwaiger)_

- We update most of the Markdown Shell Recorder tests so they use an explicit namespace (like system or user). This has the advantage that the output of these tests does not change depending on the user that executes them. Before the update these tests used cascading keys. _(René Schwaiger)_
- The Shell Recorder now also works correctly on FreeBSD. _(René Schwaiger)_
- Fix memcheck target to detect memory problems again and enabled parallel testing to speed it up. _(Mihael Pranjić)_
- Fix memleak in pluginprocess tests. _(Mihael Pranjić)_
- The test check-env-dep does not require Bash anymore. _(René Schwaiger)_
- We fixed an incorrect directive in the Markdown Shell Recorder test of the Type Checker plugin. _(René Schwaiger)_
- We added a test that invokes the script fix-spelling to check the documentation for common spelling mistakes. _(René Schwaiger)_
- We added a test that checks the formatting of Markdown files with prettier. _(René Schwaiger)_
- The test testscr_check_formatting now prints instructions that show you how to fix formatting problems. _(René Schwaiger)_

## 423.9 Build

### 423.9.1 CMake

#### 423.9.1.1 Misc

- The plugin name is now provided as compiler definition ELEKTRA_PLUGIN_NAME via CMake. See #1042. _(Peter Nirschl)_
- ELEKTRA_PLUGIN_FUNCTION does not require the module name as parameter any more, instead the ELEKTRA_PLUGIN_NAME compiler definition is being used. See #1042. _(Peter Nirschl)_
- ELEKTRA_README, and ELEKTRA_PLUGIN_EXPORT do not require the module name as parameter any more, instead the ELEKTRA_PLUGIN_NAME compiler definition is being used. See #1042. _(Peter Nirschl)_
- We now specify
  - version number,
  - project description, and
  - homepage URL in the CMake project command. _(René Schwaiger)_
- We fixed the detection of Python for the Python 2 binding on macOS. _(René Schwaiger)_
- You can now use the Ruby binding and plugin without any manual configuration, if you installed Ruby (version 2.5 or later) via Homebrew. _(René Schwaiger)_

#### 423.9.1.2 Find Modules

- The CMake find module FindAugeas.cmake does not print an error message anymore, if it is unable to locate Augeas in the pkg-config search path. _(René Schwaiger)_
- The CMake find module FindLua.cmake does not print an error message anymore, if it is unable to locate a Lua executable. _(René Schwaiger)_
- We added code that makes sure you can compile IO GLIB on macOS, even if pkg-config erroneously reports that GLIB requires linking to the library intl (part of GNU gettext). _(René Schwaiger)_

- We added a CMake find module for GLib. The module makes sure you can compile and link IO GLib on macOS. _(René Schwaiger)_
- The CMake find module FindLibOpenSSL.cmake does not require pkg-config anymore. The updated code also fixes some linker problems on macOS (and probably other operating systems too), where the build system is not able to link to OpenSSL using only the name of the OpenSSL libraries. _(René Schwaiger)_
- We simplified the CMake find module FindLibgcrypt.cmake.The update fixes problems on macOS, where the build system excluded the plugin crypto_gcrypt, although Libgcrypt was installed on the system. _(René Schwaiger)_
- We now use the official CMake find module for iconv. This update fixes linker problems with the iconv and filecheck plugin on FreeBSD 12. _(René Schwaiger)_
- The CMake find module for Botan does not require pkg-config anymore. _(René Schwaiger)_
- The CMake find module for libgit2 now also exports the version number of libgit2. _(René Schwaiger)_
- We added a CMake find module for libuv and fixed a problem on macOS, where the build system was unable to locate the header file of libuv. _(René Schwaiger)_
- We added a CMake find module for ZeroMQ to fix build problems on macOS. _(René Schwaiger)_

### 423.9.2 Docker

- We added
  - ANTLR,
  - ANTLR's C++ runtime,
  - Ninja, and
  - shfmt, to the Dockerfile for Debian sid

  . _(René Schwaiger)_

### 423.9.3 Misc

- We removed the configure script from the top-level directory. CMake is now popular enough so that this helper-script is not needed. _(René Schwaiger)_

## 423.10 Infrastructure

### 423.10.1 Cirrus

- We now use Cirrus CI to build and test Elektra on
  - FreeBSD 11.2 and
  - FreeBSD 12.0

  . Both of these build jobs use -Werror to make sure we do not introduce any code that produces compiler warnings. _(René Schwaiger)_
- The new build job red apple Clang tests Elektra on macOS. _(René Schwaiger)_
- We added the build job red apple Clang ASAN, which uses Clang with enabled Address↩ Sanitizer to test Elektra on macOS. _(René Schwaiger)_
- The new build job red apple FULL compiles and test Elektra using the CMake options BUILD_↩ SHARED=OFF an BUILD_FULL=ON. _(René Schwaiger)_

- We added `red apple MMap`, which tests Elektra using `mmapstorage` as default storage module. _(René Schwaiger)_
- We install and uninstall Elektra in all of the macOS build jobs to make sure that `ElektraUninstall.cmake` removes all of the installed files. _(René Schwaiger)_

### 423.10.2 Jenkins

- We added a badge displaying the current build status to the main `ReadMe`. _(René Schwaiger)_
- The build job `formatting-check` now also checks the formatting of Shell scripts. _(René Schwaiger)_

### 423.10.3 Travis

- We now test Elektra on `Ubuntu Xenial Xerus`. _(René Schwaiger)_
- We removed the build jobs `green apple Clang` and `green apple Check Shell` in favor of the Cirrus build job `red apple Clang`. _(René Schwaiger)_
- We removed the build jobs `green apple Clang ASAN` in favor of the Cirrus build job `red apple Clang ASAN`. _(René Schwaiger)_
- We removed the build jobs `green apple FULL` in favor of the Cirrus build job `red apple FULL`. _(René Schwaiger)_
- We removed the build jobs `green apple MMap` in favor of the Cirrus build job `red apple MMap`. _(René Schwaiger)_

## 423.11 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date.

## 423.12 Outlook

We are currently working on following topics:

- infallible high-level API: creating an API that guarantees you to return configuration values _(Klemens Böswirth)_
- error handling _(Michael Zronek)_
- elektrify LCDproc _(Klemens Böswirth)_ and _(Michael Zronek)_
- YAML as default storage _(René Schwaiger)_
- misconfiguration tracker _(Vanessa Kos)_
- global mmap cache: This feature will enable Elektra to return configuration without parsing configuration files or executing other plugins as long as the configuration files are not changed. _(Mihael Pranjić)_

and since recently:

- automatic shell completion _(Ulrike Schaefer)_
- plugin framework improvements _(Vid Leskovar)_
- 3-way merge _(Dominic Jäger)_
- reducing entry barriers for newcomers _(Hani Torabi Makhsos)_
- bug fixing _(Usama Morad)_ and _(Kurt Micheli)_

## 423.13 Statistics

In this release we created 986 commits in which 802 files were changed, with 21687 insertions(+) and 6912 deletions(-).
Following authors made this release possible:

| Commits | Author |
|---|---|
| 1 | Aybuke Ozdemir aybuke.147@gmail.com |
| 2 | Gabriel Rauter rauter.gabriel@gmail.com |
| 6 | Bernhard Denner bernhard.denner@gmail.com |
| 6 | Dominic Jäger dominic.jaeger@gmail.com |
| 25 | Peter Nirschl peter.nirschl@gmail.com |
| 32 | Mihael Pranjic mpranj@limun.org |
| 66 | Michael Zronek michael.zronek@gmail.com |
| 112 | Markus Raab elektra@markus-raab.org |
| 131 | Klemens Böswirth k.boeswirth+git@gmail.com |
| 141 | Dominik Hofer me@dominikhofer.com |
| 464 | René Schwaiger sanssecours@me.com |

## 423.14 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 423.15 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.8.26.tar.gz
- size: 6395865
- md5sum: 4ef202b5d421cc497ef05221e5309ebc
- sha1: 94f654764bcf49d0ebc7e636f444e24ca6cfeb19
- sha256: 5806cd0b2b1075fe0d5a303649d0bd9365752053e86c684ab7c06e7f369155d3

The release tarball is also available signed by Markus Raab using GnuPG from here or on GitHub
Already built API documentation can be found here or on GitHub.

## 423.16 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 424

# 0.9.0 Release

- guid: e8c753c0-74af-410b-9f66-77c3ce194717
- author: Markus Raab
- pubDate: Tue, 06 Aug 2019 12:09:02 +0200
- shortDesc: Cache, Command-line Options, Error Codes

## 424.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database.
You can also read this document on our website.
We wrote a new article describing our vision from configuration management perspective.
For more information, visit https://libelektra.org.

## 424.2 0.9.∗

We are proud to present our largest release so far. It is the first release of the 0.9.∗ version series, which goal is it:

- To prepare Elektra for version 1.0.0, which includes incompatible changes like new error codes (which are already part of this release, hence 0.9.0). When 0.9.∗ is mature enough, we will call it 1.0.0.
- To make Elektra future-proof so that during 1.∗. most of Elektra's functionality can be kept stable
- To cleanup Elektra, including removal of compatibility layers and research prototypes.

## 424.3 Business Plans

To get away from a purely research-oriented approach to a mature foundation, we also need paid employees to fix problems.
We plan to introduce following ways of income:

1. donations
2. paid support/feature requests
3. consultancy

If you are interested in any of these, please contact us via business@libelektra.org
The 0.8.∗ version series will be maintained on paid requests. If you have maintenance requests and want 0.8.27 to be released, please contact us via business@libelektra.org
Please note, that Elektra will definitely stay 100% free software (BSD licensed). We do not plan to make any part proprietary. We only introduce a way for paid customers to get desired features or help more quickly.

## 424.4 Highlights

- Cache
- Command-line Options
- Error Codes

### 424.4.1 Cache

Cache is a new global caching plugin. It uses mmapstorage as its storage backend and lazily stores the whole configuration from previous configuration accesses.
With large or many configuration files, the cache brings amazing performance improvements: Let us say, you have 649 INI configuration files mounted with the multifile resolver, completely specified (which means that the specification must be copied to all the configuration settings). Before the cache, retrieving the whole configuration would take 6 or even 13 seconds. With the cache, the whole operation now takes less than 0.5 seconds after the first access.
This is an important step towards the goal of Elektra to integrate all configuration files present on a system.
Limitations:

- Mountpoints that are not connected with files, currently cannot be cached.
- The cache currently does not work together with other global plugins.

By default, the cache will automatically enable itself once the cache plugin is installed. The cache can be found in ~/.cache/elektra.
We also added tools for enabling, disabling and clearing the cache (kdb cache {enable,disable,default,clear}).
A big thanks to _(Mihael Pranjić)_ for the excellent work.

### 424.4.2 Command-line Options

Gopts is a new plugin that integrates support for command-line options to applications:

- The gopts plugin retrieves the values of argc, argv and envp needed for elektraGetOpts and then makes the call. It is intended to be used as a global plugin, so that command-line options are automatically parsed when kdbGet is called. _(Klemens Böswirth)_
- The plugin works under WIN32 (via GetCommandLineW and GetEnvironmentString), MAC_↩ OSX (_NSGetArgc, _NSGetArgv) and any system that either has a sysctl(3) function that accepts KERN_PROC_ARGS (e.g. FreeBSD) or when procfs is mounted and either /proc/self or /proc/curproc refers to the current process. If you need support for any other systems, feel free to add an implementation.

This means, that using the plugin, you do not need to call elektraGetOpts yourself anymore.
kdbEnsure is a new function in elektra-kdb. It can be used to ensure that a KDB instance meets certain clauses specified in a contract. In principle this a very powerful tool that may be used for a lot of things. All changes made by kdbEnsure are purely within the KDB handle passed to the function.
For example, kdbEnsure can be used, to ensure the availability of command-line options for your application.
Limitations:

- kdbEnsure only works, if the list plugin is mounted in all appropriate global positions.
- kdbEnsure does not take care of dependencies between plugins.
- Mounting of non-global plugins is not supported.

A big thanks to _(Klemens Böswirth)_ for the excellent work.

### 424.4.3 Error Codes

With this release, we changed our messy error code system into a more structured and clean way. Similar to SQLStates we changed to structure of our error codes and migrated them. Have a look into the new codes. This allows us to easily extend the specification without breaking existing codes and to avoid risking duplicated errors as we had before. _(Michael Zronek)_
We were able to reduce the former 214 to now only 9 error codes.
For background information read:

- about error codes
- about error message format

A big thanks to _(Michael Zronek)_ for the excellent work.

## 424.5 Plugins

The following section lists news about the plugins we updated in this release. In total, we added 9 plugins and removed 2 plugins.

### 424.5.1 Type (New Version)

The type plugin was completely rewritten in C. The old version is now called cpptype. _(Klemens Böswirth)_
The new type plugin also provides the functionality of the enum and the boolean plugin. These plugins are now considered obsolete and you should use type instead.
A few notes on compatibility:

- the new type does not support the full feature set of enum and boolean, but it supports the features we consider useful.
- the new type doesn't support FSType and empty. These have been deprecated for a long time and there are good alternatives available.
- the new type supports enum, wchar and wstring as types, whereas the old cpptype would throw an error for these. In most cases this won't be a problem, but you should be aware of this breaking change.
- the new type does not support check/type/min and check/type/max, please use the range plugin instead.

To switch from enum to the new type, you have to add either check/type=enum or type=enum. Without a check/type or type metakey, the type plugin will ignore the key. We now also support converting enum values to and from integer values (see README).
To switch from boolean to the new type, you don't have to do anything, if you used the default config. If you used a custom configuration please take a look at the README.

### 424.5.2 Base64

- We fixed some warnings about implicit type conversions reported by UBSan in the base64 plugin. _↩ (René Schwaiger)_

### 424.5.3 Crypto and Fcrypt

- Empty GPG key IDs in the plugin configuration are being ignored by the crypto plugin and the fcrypt plugin. Adding empty GPG key IDs would lead to an error when gpg is being invoked. _(Peter Nirschl)_
- Apply Base64 encoding to the master password, which is stored within the plugin configuration. This fixes a problem that occurs if ini is used as default storage (see 2591). _(Peter Nirschl)_
- Fix compilation without deprecated OpenSSL APIs. Initialization and deinitialization is not needed anymore. _(Rosen Penev)_

#### 424.5.4 CSVStorage

- Support DOS newlines for the `csvstorage` plugin. _(Vlad - Ioan Balan)_

#### 424.5.5 Filecheck

- We fixed some warnings about implicit type conversions reported by `UBSan`. _(René Schwaiger)_

#### 424.5.6 INI

- Fixed `INI` plugin when only the root key needs to be written. _(Mihael Pranjić)_
- Plugin writes to INI files without spaces around '=' anymore. Reading is still possible with and without spaces. _(Oleksandr Shabelnyk)_

#### 424.5.7 Macaddr

- Added a plugin to handle MAC addresses. `kdbGet` converts a MAC address into a decimal 64-bit integer (with the most significant 16 bits always set to 0), if the format is supported. `kdbSet` restores the converted values back to there original form. _(Thomas Bretterbauer)_

#### 424.5.8 mINI

- We fixed compiler warnings reported by GCC 9 in the `unit test code` of the plugin. _(René Schwaiger)_

#### 424.5.9 Mmapstorage

- `mmapstorage` is now able to persist the Global KeySet, which is used by the `cache` plugin. _(Mihael Pranjić)_
- Fixed support for `kdb import` and `kdb export`. _(Mihael Pranjić)_

#### 424.5.10 Multifile

- Fixed segmentation fault in `kdbError()` function. _(Mihael Pranjić)_
- Added Global Keyset handle to storage plugin. _(Mihael Pranjić)_
- Fixed use of wrong resolver handle in the `kdbError()` function. _(Mihael Pranjić)_

#### 424.5.11 Quickdump

- `quickdump` is a new storage plugin. It implements a more concise form of the `dump` format, which is also quicker too read. Contrary to dump, quickdump only stores keynames relative to the parent key. This allows easy relocation of configurations. _(Klemens Böswirth)_
- quickdump now also uses an variable length integer encoding to further reduce file size. _(Klemens Böswirth)_

#### 424.5.12 Reference

- Fixed missing Metadata in README and METADATA.ini. _(Michael Zronek)_
- Update README.md web tool to show, how to test REST API on localhost. _(Dmytro Moiseiuk)_

#### 424.5.13 RGBColor

- `New plugin` to validate hex formatted colors (e.g. #fff or #abcd) and normalize them to rgba (4294967295 (= 0xffffffff) and 2864434397 (= 0xaabbccdd) respectively). It also has support for named colors according to the `extended color keywords` from CSS3. _(Philipp Gackstatter)_

### 424.5.14 Semlock

Removed due to:

- constant pain
- never worked properly
- poor design
- no time in future to maintain _(Kurt Micheli)_

### 424.5.15 Spec

- The spec plugin was partly rewritten to better support specifications for arrays. This includes some breaking changes concerning the less used (and also less functional) parts of the plugin. To find out more about these changes take a look at the README. It now better reflects the actually implemented behavior. _(Klemens Böswirth)_

### 424.5.16 Specload

- The `specload` plugin is a special storage plugin. Instead of using a storage file it calls an external application to request its specification. For the transfer it relies on the `quickdump` plugin. _(Klemens Böswirth)_
- Currently changing the specification is only allowed in a very limited way. However, in future the plugin should allow overriding a specification in all cases where this can be done safely. NOTE: While the plugin technically allows some modifications, because of a problem with the resolver this cannot be used right now (see `limitations`).
- We also export `elektraSpecloadSendSpec` to abstract over the `quickdump` dependency. _(Klemens Böswirth)_

### 424.5.17 Syslog

- We fixed an incorrect format specifier in a call to the `syslog` function. _(René Schwaiger)_

### 424.5.18 Unit

- `New plugin` to validate units of memory and normalize them into bytes. E.g. 20 KB (normalized to 20000 Byte). _(Marcel Hauri)_

### 424.5.19 YAJL

The `YAJL` plugin which parses JSON files:

- now allows setting a value to the mountpoint. This is represented as a top level value in JSON if no other key is present. _(Philipp Gackstatter)_
- no longer lists empty parent keys with `kdb ls`. _(Philipp Gackstatter)_
- signifies arrays with the metakey array according to the array decision. _(Philipp Gackstatter)_
- no longer produces additional `___dirdata` entries for empty array keys. See also issue `#2477`. _(Philipp Gackstatter)_

### 424.5.20 YAMBi

- YAMBi is now able detect multiple syntax errors in a file. _(René Schwaiger)_
- The error message now includes more information about the location of syntax errors. For example, for the incorrect YAML input `config.yaml`:

```
key 1: - element 1
 - element 2
key 2: scalar
       - element 3
```

, the plugin prints an error message that includes the following text:

```
config.yaml:2:2: syntax error, unexpected start of sequence, expecting end of map or key
                 - element 2
                 ^
config.yaml:4:8: syntax error, unexpected start of sequence, expecting end of map or key
                       - element 3
                       ^
```

. _(René Schwaiger)_

- YAMBi now supports Elektra's boolean data type. _(René Schwaiger)_
- The plugin now handles YAML key-value pairs without a value at the end of a file correctly. _(René Schwaiger)_
- The plugin now converts YAML key-value pairs with empty value to null/empty keys. _(René Schwaiger)_
- YAMBi now converts empty files to a key set containing an empty version of the parent key. _(René Schwaiger)_

### 424.5.21 YAML CPP

- The plugin now handles keys that are part of a map, but use a basename ending with array syntax correctly. For example, in a key set that contains keys with the following names:

```
user/array/#0
user/array/#1
user/map/#0
user/map/key
user/map/#1
```

, `user/array/#0` and `user/array/#1` represent array elements, while `user/map/#0`, and `user/map/#1` do not, since the key set also contains the key `user/map/key`. The following `Markdown Shell Recorder` snippet shows the new behavior of the plugin:

```
kdb mount config.yaml user yamlcpp
kdb set user/array/#0 one
kdb set user/array/#1 two
kdb set user/map/#0   three
kdb set user/map/key  four
kdb set user/map/#1   five
kdb file user | xargs cat
#> array:
#>   - one
#>   - two
#> map:
#>   "#0": three
#>   "#1": five
#>   key: four
```

. _(René Schwaiger)_

- YAML CPP now handles the conversion from and to Elektra's boolean type properly. _(René Schwaiger)_
- The plugin converts "sparse" key sets properly. For example, for the key set that contains **only** the key:
  - `user/parent/#1/#2/map/#0` with the value `arr`

  and uses `user/parent` as parent key, YAML CPP stores the following YAML data:

```
- ~
- - ~
  - ~
  - map:
      - arr
```

. _(René Schwaiger)_

- YAML CPP now supports mixed data (nested lists & sequences) better. For example, the plugin now correctly converts the YAML data

```
root:
  - element: one
  - element: two
```

to the key set that contains the following keys:

```
user/tests/yaml/root
user/tests/yaml/root/#0/element
user/tests/yaml/root/#1/element
```

### 424.5.22 YAML Smith

- YAML Smith now converts keys that shares a common prefix correctly. For example, the last command in the script:

```
kdb mount config.yaml user/tests/yaml yaml
kdb set user/tests/yaml/common/one/#0 value
kdb set user/tests/yaml/common/two/#0 first
kdb set user/tests/yaml/common/two/#1 second
kdb export user/tests/yaml yamlsmith
```

now prints the following YAML data:

```
common:
  one:
    - "value"
  two:
    - "first"
    - "second"
```

. _(René Schwaiger)_

- The plugin now converts Elektra's boolean values (`0`, `1`) back to YAML's boolean values (`true`, `false`). _(René Schwaiger)_

### 424.5.23 Yan LR

- The build system now disables the plugin, if you installed a version of ANTLR 4 that does not support ANTLR's C++ runtime (like ANTLR `4.5.x` or earlier). _(René Schwaiger)_
- We fixed an ambiguity in the YAML grammar. _(René Schwaiger)_
- The build system now regenerates the modified parsing code, every time we update the grammar file. _(René Schwaiger)_
- The plugin now reports the location of syntax errors correctly. _(René Schwaiger)_
- The lexer for the plugin now emits start tokens for maps at the correct location inside the token stream. This update fixes a problem, where the plugin sometimes reported incorrect error messages for the *first* syntax error in a YAML file. _(René Schwaiger)_
- The plugin now stores the end position of map start tokens correctly. Before this update the plugin would sometimes not show the markers (^) that point to the error positions inside the input. _(René Schwaiger)_
- Yan LR now supports Elektra's boolean data type. _(René Schwaiger)_
- The plugin now handles YAML key-value pairs that contain no value at the end of a file correctly. _(René Schwaiger)_
- The plugin now converts YAML key-value pairs with empty value to null/empty keys. _(René Schwaiger)_
- The plugin converts "empty" YAML files to a key set that contains an empty version of the parent key. _(René Schwaiger)_

### 424.5.24 YAwn

- YAwn is now able to print error messages for multiple syntax errors. _(René Schwaiger)_
- We also improved the error messages of YAwn, which now also contain the input that caused a syntax error. For example, for the input

```
key: value
  - element
```
the plugin prints an error message that contains the following text:
```
config.yaml:2:3: Syntax error on input "start of sequence"
                   - element
                   ^
```
. _(René Schwaiger)_

- The plugin now supports Elektra's boolean data type. _(René Schwaiger)_
- YAwn handles YAML key-value pairs that contain no value at the end of a file correctly. _(René Schwaiger)_
- The plugin now converts YAML key-value pairs with empty value to null/empty keys. _(René Schwaiger)_
- YAwn now stores empty files as a key set containing an empty parent key. _(René Schwaiger)_

### 424.5.25 YAy PEG

- YAy PEG now also supports PEGTL 2.8. _(René Schwaiger)_
- The plugin now includes the input that could not be parsed in error messages. _(René Schwaiger)_
- We improved the error messages for certain errors slightly. For example, the error message for the input

```
"double quoted
```
now includes the following text
```
1:14: Missing closing double quote or incorrect value inside flow scalar
      "double quoted
                    ^
```
. _(René Schwaiger)_

- YAy PEG now supports compact mappings:

```
- key1: value1
  key2: value2
```
and compact sequences:
```
- - element1
  - element2
```
correctly. _(René Schwaiger)_

- The plugin now supports Elektra's boolean data type. _(René Schwaiger)_
- YAy PEG now converts YAML key-value pairs with empty value to null/empty keys. _(René Schwaiger)_
- The plugin now translates an empty file to a key set that contains a single empty parent key. _(René Schwaiger)_

## 424.6 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 424.6.1 Compatibility

We introduced several incompatible changes:

- different error codes are returned
- INI and YAJL plugins might write different files with the same KeySets

We changed following symbols:

- elektraIsReferenceRedundant
- elektraResolveReference
- elektraPluginFindGlobal
- kdbEnsure

### 424.6.2 Core

- kdbGet now calls global postgetstorage plugins with the parent key passed to kdbGet, instead of a random mountpoint. _(Klemens Böswirth)_
- Fixed a double cleanup error (segmentation fault) when mounting global plugins. _(Mihael Pranjić)_
- Logging in Elektra was changed with this release. If Elektra is compiled with ENABLE_LOGGER enabled, we now log warnings and errors to stderr and everything except debug messages to syslog. If ENABLE_DEBUG is also enabled, debug messages are logged to syslog as well. Previously you had to make some manual changes to the code, to see most of the logging messages. _(Klemens Böswirth)_
- The logger does not truncate the file name incorrectly anymore, if __FILE__ contains a relative (instead of an absolute) filepath. _(René Schwaiger)_
- Disabled any plugin execution when we have a cache hit or no update from backends. The old behavior can be enabled for testing using ENABLE_DEBUG and adding the "debugGlobalPositions" metakey to the parentKey of the kdbGet invocation. _(Mihael Pranjić)_
- Removed ingroup from error messages to reduce verbosity. _(Michael Zronek)_
- Fixed minor problem when kdb_long_double_t is not available (e.g. mips32). _(Matthias Schoepfer)_
- Only add benchmarks if BUILD_TEST is set in CMake. _(Matthias Schoepfer)_
- We fixed some warnings about implicit conversion to unsigned int reported by UBSan. _(René Schwaiger)_

### 424.6.3 Ease

- The functions for reference resolving used in the reference plugin have been extracted into libease. This lets other parts of Elektra easily use references and ensures a consistent syntax for them. _(Klemens Böswirth)_

## 424.7 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

- JNA is now not experimental anymore. _(Markus Raab)_
- gsettings is not default anymore when installed. _(Markus Raab)_
- Add fix for creating the Key and KeySet objects in the HelloElektra.java file. _(Dmytro Moiseiuk)_
- We fixed a warning about a deprecated default constructor in the C++ binding reported by GCC 9.0. _(René Schwaiger)_

## 424.8 Tools

- kdb get -v now displays if the resulting value is a default-value defined by the metadata of the key. _(↩ Thomas Bretterbauer)_
- kdb cp now succeeds if the target-keys already have the same values as the source-keys. _(Thomas Bretterbauer)_
- web-ui does not show empty namespace anymore _(Josef Wechselauer)_
- kdb import does not fail anymore if executed more than once with the same target in the spec-namespace. _(Thomas Bretterbauer)_
- kdb mount avoid adding sync if sync is already provided. _(Markus Raab)_

- `kdb list-tools` now supports `KDB_EXEC_PATH` environment variables that contain spaces. _(René Schwaiger)_
- `gen-gpg-testkey` is added to the default tools list (see #2668)._(Peter Nirschl)_
- `kdb getenv` now executed correctly from within tests _(Markus Raab)_
- `kdb-bash-completion` now works on Mac (see #2836). _(Eduardo Santana)_
- `kdb rm` supports `--without-elektra` and returns 11 on key not found. _(Markus Raab)_

### 424.8.1 Code generation

`kdb gen` is now no longer an external tool implemented via python, but rather a first class command of the `kdb` tool. For now it only supports code generation for use with the highlevel API. Try it by running `kdb gen elektra <parentKey> <outputName>`, where `<parentKey>` is the parent key of the specification to use and `<outputName>` is some prefix for the output files. If you don't have your specification mounted, use `kdb gen -F <plugin>:<file> elektra <parentKey> <outputName>` to load it from `<file>` using plugin `<plugin>`.
. _(Klemens Böswirth)_

## 424.9 Scripts

- The `reformat-shfmt` script now also formats `tests/shell/include_common.sh.in`. Additionally it ensures that the file is 1000 lines long, so that line numbers of files using it are easier to read. _(Klemens Böswirth)_
- The clang-format wrapper script now also checks the supported maximum version of Clang-Format. _(René Schwaiger)_
- The script `reformat-shfmt` now also reformats shell support files (`*.in`) in the `scripts` folder. _(René Schwaiger)_
- The `reformat-*` scripts now allow you to specify a list of files that should be formatted. Only files actual suitable for the reformat script, will reformat. So e.g. calling `reformat-cmake src/include/kdbprivate.h` doesn't change any files. _(Klemens Böswirth)_
- The script `scripts/dev/reformat-all` is a new convenience script that calls all other `reformat-*` scripts. _(Klemens Böswirth)_
- The script `scripts/pre-commit-check-formatting` can be used as a pre-commit hook, to ensure files are formatted before committing. _(Klemens Böswirth)_
- The link checker now prints broken links to the standard error output. _(René Schwaiger)_
- We added a script, called `benchmark-yaml` that compares the run-time of the YAML plugins:
  - `YAML CPP`,
  - `Yan LR`,
  - YAMBi,
  - YAwn, and
  - YAy PEG

  for a certain input file with `hyperfine`. _(René Schwaiger)_
- Added `kdb reset` and `kdb reset-elektra`, fixed `kdb stash`. _(Markus Raab)_

## 424.10 Benchmarks

- The benchmarking tool `benchmark_plugingetset` now also supports only executing the `get` method for the specified plugin. For example, to convert the data stored in the file `benchmarks/data/yaypeg_test.in` with the YAy PEG plugin to a key set you can now use the following command:

```
benchmark_plugingetset benchmarks/data user yaypeg get
```

. _(René Schwaiger)_

## 424.11 Documentation

### 424.11.1 Style

- The documentation now uses `fenced code blocks` to improved the syntax highlighting of code snippets. _(René Schwaiger)_
- We added recommendations about the style of Markdown headers to our coding guidelines. _(René Schwaiger)_
- We now use `title case` for most headings in the documentation. _(René Schwaiger)_
- We added instructions on how to reformat code with
  - `Clang-Format`,
  - `cmake-format`,
  - `Prettier`, and
  - `shfmt`

  to the coding guidelines. _(René Schwaiger)_

### 424.11.2 Tutorials

- We added a basic tutorial that tells you how to write a (well behaved) storage plugin. _(René Schwaiger)_
- Improved the `checkconf` section in the plugin tutorial. _(Peter Nirschl)_
- We added a tutorial on how to benchmark the execution time of plugins using `benchmark_plugingetset` and `hyperfine`. _(René Schwaiger)_
- The new profiling tutorial describes how to determine the execution time of code using
  - `Callgrind`, and
  - `XRay`

  . _(René Schwaiger)_
- For beginners we added a `tutorial` that guides them through the process of contributing to libelektra. _(Thomas Bretterbauer)_
- Added a section on `elektraPluginGetGlobalKeySet` in the plugin tutorial. _(Vid Leskovar)_
- Added a step-by-step tutorial for beginners to run all tests with Docker. _(Oleksandr Shabelnyk)_
- Extend/improve Java bindings related documentation in `tutorial` and readme. _(Oleksandr Shabelnyk)_
- Added a step-by-step tutorial for running reformatting scripts with Docker. _(Oleksandr Shabelnyk)_
- Covered Resolving Missing ∗.so Library Error in tutorial. _(Oleksandr Shabelnyk)_
- Added a basic tutorial on How-To: Write a Java Plugin _(Dmytro Moiseiuk)_ and _(Miruna Orsa)_

### 424.11.3 Spelling Fixes

- Write Elektra with capital letter in cascading tutorial. _(Vlad - Ioan Balan)_
- Add typo fix to the hello-elektra tutorial. _(Dmytro Moiseiuk)_
- Add typo fix to the Java kdb tutorial. _(Dominik Hofmann)_
- Fixed capitalization of the initial letter in Readme. _(Miruna Orsa)_
- Improved readability in README. _(Philipp Gackstatter)_
- We fixed some spelling mistakes in the documentation. _(René Schwaiger)_
- Fix typo in root README.md and 'built-in' => 'built-in' in several places _(Raphael Gruber)_
- Fixed typos in `cassandra.ini` _(arampaa)_

### 424.11.4 Other

- The `Markdown Link Converter` now uses the style

`filename:line:0`
instead of
`filename|line col 0|`
to show the location data for broken links. This is also the same style that Clang and GCC use when they display location information for compiler errors. This update has the advantage, that certain tools such as `TextMate` are able to convert the location data, providing additional features, such as clickable links to the error source. _(René Schwaiger)_

- The `Markdown Link Converter` uses the index `1` for the first line number instead of `0`. This update fixes an off-by-one-error, when the user tries to use the error location data printed by the tool in a text editor. _(René Schwaiger)_
- We added a badge for `LGTM` to the `main ReadMe file`. _(René Schwaiger)_
- Added LCDproc and `Cassandra` specification examples. These examples provide a good guideline for writing specifications for configurations. _(Michael Zronek)_
- Drastically improved the error message format. For more information look here. _(Michael Zronek)_
- Added a guideline for writing consistent and good error messages. For more information look here. _(Michael Zronek)_
- Every `kdb` command now accepts `v` and `d` as option to show more information in case of warnings or errors. _(Michael Zronek)_
- Improved qt-gui error popup to conform with the new error message format. _(Raffael Pancheri)_
- We fixed the format specifiers in the `"Hello, Elektra" example`. _(René Schwaiger)_
- Expanded the Python Tutorial to cover installation under Alpine Linux. _(Philipp Gackstatter)_
- We wrote a tutorial which is intended to help newcomers contributing to libelektra. _(Thomas Bretterbauer)_
- We fixed various broken links in the documentation. _(René Schwaiger)_
- Fix finding of jni.h library. _(Dmytro Moiseiuk)_
- Added license for asciinema. _(Anastasia @nastiaulian)_
- We incorporated `kdb-introduction` into the `man page for kdb` _(Markus Raab)_

## 424.12 Tests

- We now test the `Directory Value Plugin` with additional test data. _(René Schwaiger)_
- The variables:
  - `SPEC_FOLDER`
  - `SYSTEM_FOLDER`
  - `USER_FOLDER`

  in the inclusion file for shell test were set incorrectly, if the repository path contained space characters. _(René Schwaiger)_
- The `CFramework` now also compares the names of meta keys. _(René Schwaiger)_
- The release notes check does not report an illegal number anymore, if the release notes were not updated at all. _(René Schwaiger)_
- We added a test for the keyhelper-class which checks if rebasePath calculates the new path for cascading target-keys correctly. _(Thomas Bretterbauer)_
- Enable MSR for the crypto and fcrypt tutorial ( `#1981`)._(Peter Nirschl)_
- We fixed the `Markdown Shell Recorder` test for the command `kdb get`. _(René Schwaiger)_
- The tests
  - `testscr_check_real_world`,
  - `testscr_check_resolver`, and
  - `testscr_check_spec`

  now also works correctly, if the `user` and `system` directory file paths contain space characters. _(René Schwaiger)_

### 424.12.1 Source Code Checks

- The formatting instructions printed by `check_formatting` now also work correctly, if
  - the `diff` output does not start with the test number added by CTest, and
  - you use a non-POSIX shell such as `fish`

  . _(René Schwaiger)_
- We reformatted the CMake source code with cmake-format `0.5.4` and also check the style of CMake code with this new version of the tool. _(René Schwaiger)_
- We now check the source code of the repository with `LGTM`. _(René Schwaiger)_
- We fixed various warnings about
  - missing or duplicated include guards,
  - undefined behavior,
  - incorrect format specifiers,
  - unnecessary statements,
  - short names for global variables, and
  - empty `if`-statements

  reported by `LGTM`. _(René Schwaiger)_

## 424.13 Build

### 424.13.1 CMake

- We now disable the option `INSTALL_SYSTEM_FILES` by default. This change makes it possible to install Elektra using `Homebrew` on Linux without any changes to `Elektra's Linuxbrew formula`. _(René Schwaiger)_ Add `-DINSTALL_SYSTEM_FILES=ON` for previous behavior.
- The build system now rebuilds the `JNA binding` with Maven, if you change any of the Java source files of the binding. _(René Schwaiger)_
- `testshell_markdown_tutorial_crypto` is not compiled and executed if `gen-gpg-testkey` is not part of TOOLS. _(Peter Nirschl)_
- Plugin tests are now only added, if `BUILD_TESTING=ON`. _(Klemens Böswirth)_
- The symbol list for the static version is now exported directly from a CMake function. _(Klemens Böswirth)_
- Building Elektra with enabled `io_glib` binding does not require libuv anymore. _(René Schwaiger)_

### 424.13.2 Docker

#### 424.13.2.1 Alpine Linux

- Our Docker image for Alpine Linux now uses the base image for Alpine Linux 3.9. _(René Schwaiger)_
- We added PEGTL to the Alpine Docker image. _(René Schwaiger)_

#### 424.13.2.2 Debian

- We now use the default JDK on Debian sid, since the package `openjdk-8-jdk` is not available in the official unstable repositories anymore. _(René Schwaiger)_
- We added
  - `Bison`, and
  - YAEP

  to the image for Debian sid. _(René Schwaiger)_
- We now offer images for the latest stable version of Debian codenamed "buster". _(René Schwaiger)_

#### 424.13.2.3 Ubuntu

- We added a Dockerfile for Ubuntu Disco Dingo. _(René Schwaiger)_

#### 424.13.2.4 Other Updates

- The Docker images for
  - `Debian stretch`, and
  - Debian sid,

  now include the Python YAML library recommended by `cmake-format`. _(René Schwaiger)_

### 424.13.3 Vagrant

- The Vagrant file for Ubuntu Artful Aardvark now installs the Python YAML library recommended by `cmake-format`. _(René Schwaiger)_

## 424.14 Infrastructure

### 424.14.1 Cirrus

- We added the build job `link symbol Check`, which checks the documentation for broken links. _(René Schwaiger)_

### 424.14.2 Jenkins

- We disabled the tests:
  - `testmod_crypto_botan`,
  - `testmod_crypto_openssl`,
  - `testmod_dbus`,
  - `testmod_dbusrecv`,
  - `testmod_fcrypt`,
  - `testmod_gpgme`, and
  - `testmod_zeromqsend`

  , since they are `known to fail in high load scenarios`. _(René Schwaiger)_
- We added deprecated plugins to the tests. _(Markus Raab)_
- We increased the automatic timeout for jobs that show no activity from 5 to 10 minutes. _(René Schwaiger)_
- We improved the exclusion patterns for the `Coveralls coverage analysis`. _(René Schwaiger)_
- We now again build the API documentation of `master` and we now also build the API documentation of `PRs`. _(Markus Raab)_
- We added buster build jobs. _(Markus Raab)_

### 424.14.3 Restyled

- We added a configuration file for `Restyled`. Currently `Restyled` monitors changes to Shell code in pull requests and fixes code that does not fit the coding guideline, by adding additional formatting commit to PRs. _(René Schwaiger)_

### 424.14.4 Travis

- We removed the build job for the Haskell binding and Haskell plugin. For more information, please take a look `here`. _(Klemens Böswirth)_
- We always use GCC 9 for the build job `green apple GCC`. This update makes sure that the build job succeeds, even if Homebrew adds a new major version of the compiler. _(René Schwaiger)_
- We simplified our Travis configuration file, removing various unnecessary and unused code. In this process we also got rid of the caching directives, we previously used to speed up the Haskell build job `green apple Haskell`. _(René Schwaiger)_

## 424.15 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Our entry for Elektra has been approved in the Free Software Directory: `https://directory.fsf.org/wiki/Elektra`
- Added GitHub build status badges to website _(hesirui)_

- We updated part of a test for the snippet converter. _(René Schwaiger)_
- Fixed anchor links on the website _(hesirui)_
- Added Docsearch in top-bar of website _(hesirui)_

## 424.16 Outlook

We are currently working on following topics:

- Go bindings and improved Web-UI _(Raphael Gruber)_
- semantic 3-way merging _(Dominic Jaeger)_
- improved error handling _(Michael Zronek)_
- Rust bindings _(Philipp Gackstatter)_
- elektrify LCDproc _(Klemens Böswirth)_ and _(Jakob Fischer)_
- configuration upgrades _(Lukas Kilian)_
- default storage _(René Schwaiger)_ and _(Jakob Fischer)_
- shell completion _(Ulrike Schäfer)_
- improved developer experience _(Hani Torabi)_
- Ansible bindings _(Thomas Waser)_
- misconfiguration tracker _(Vanessa Kos)_
- plugin interface improvements _(Vid Leskovar)_

## 424.17 Statistics

About 40 authors changed 1278 files with 49409 insertions(+) and 13883 deletions(-) in 2025 commits.
We closed 114 issues for this release.

## 424.18 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 424.19 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: /home/markus/elektra-0.9.0.tar.gz
- size: 7390149
- md5sum: 5cf9935515aba0567d6014a3693e415a
- sha1: 05ebe99c87b89a7cac58bf45cd3aff200cc1fd8d
- sha256: fcdbd1a148af91e2933d9a797def17d386a17006f629d5146020fe3b1b51ddd8

The release tarball is also available signed by Markus Raab using GnuPG from here or on GitHub
Already built API documentation can be found here or on GitHub.

## 424.20 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 425

# 0.9.1 Release

- guid: 8c64fe8a-87a5-4b72-b772-d98c8a4a5efd
- author: Mihael Pranjić
- pubDate: Tue, 26 Nov 2019 14:55:19 +0100
- shortDesc: KDE Integration, Rust&Go Bindings, Code Generation

We are proud to release Elektra 0.9.1.

## 425.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`

## 425.2 Highlights

- We are working on integrating Elektra into `KDE`. A new Elektra plugin called `kconfig` was added which can read KDE's kconfig ini files. Additionally we are working on a `fork` of KDE's `KConfig` configuration system and patching it to use libelektra. _(Dardan Haxhimustafa)_ and _(Felix Resch)_
- The `elektra` and `elektra-sys` crates have been published to crates.io for easier usage of our Rust binding. _(Philipp Gackstatter)_
- We improved our `Go bindings`. _(Raphael Gruber)_
- Code Generation is ready for productive use. _(Klemens Böswirth)_

### 425.2.1 Code Generation

While the new `kdb gen` was already included in the last release, it is now fully functional and ready for productive use. To get started take a look at the new man-page for `kdb-gen(1)`.
If you specifically want to use it with the High-Level API take a look at `this tutorial`.
We also created a new CMake function that will be available, if you include Elektra via CMake's `find_package`. The function is called `elektra_kdb_gen` and can be used to tell CMake about files that are generated via `kdb gen`. _(Klemens Böswirth)_

### 425.2.2 Further Highlights

- We migrated our `build server` and `website` to up-to-date hardware. _(Markus Raab and Djordje Bulatovic)_
- Elektra now has a technical preview of a new merge library. It is written in C99 and can currently be used with `kdb cmerge`. _(Dominic Jäger)_

- We make it easier for new developers to join Elektra, e.g., see new get started. _(Hani Torabi)_
- We made many cleanups, to get Elektra ready for 1.0!

## 425.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 425.3.1 General

- We removed 12 obsolete or unfinished plugins:
  - `boolean`,
  - `cachefilter`,
  - `cpptype`,
  - `dini`,
  - `enum`,
  - `regexstore`,
  - `required`,
  - `haskell`,
  - `simplespeclang`,
  - `regexdispatcher`,
  - `typechecker`,
  - `struct`. _(Markus Raab, René Schwaiger)_
- We unified the name of the config check function of the plugins to `nameOfPluginCheckConf`. Before this update some plugins used the name `nameOfPluginCheckConfig` instead. _(René Schwaiger)_
- We improved the error messages in `crypto`, `fcrypt`, and `gpgme` plugins. _(Peter Nirschl)_
- We now correctly handle return codes (error codes) of `execv` in the GPG module. _(Peter Nirschl)_
- We improved the documentation of `ksAppendKey` regarding ownership of keys. _(Raphael Gruber)_
- We removed the experimental status of the following plugins:
  - `gopts`
  - `specload`. _(Klemens Böswirth)_
- We added the following experimental plugins:
  - `kconfig`. _(Dardan Haxhimustafa)_

### 425.3.2 Camel

We removed the experimental plugin. For a plugin that is able to parse similar syntax, please take a look at the

- YAJL , and
- YAML CPP

plugins. _(René Schwaiger)_

### 425.3.3 GOpts

- The error message, if non of the gopts variants can be compiled, was improved. _(Klemens Böswirth)_
- A better error, if the plugin fails to load `argv` from the system, was added. _(Klemens Böswirth)_
- A function to detect help mode, without invoking `elektraGetOpts` was added. It simply checks, whether `--help` is one of the string in `argv`. _(Klemens Böswirth)_
- Increase test timeout from 120s to 240s. _(Mihael Pranjić)_

### 425.3.4 KConfig

- We added a plugin which can be used to parse kconfig INI files. _(Dardan Haxhimustafa)_

### 425.3.5 Mmapstorage

- We now store the OPMPHM inside of the mmap format. _(Mihael Pranjić)_
- The storage format was changed and many sanity checks were improved or added. _(Mihael Pranjić)_
- Enforce consistency by writing the magic file footer last. _(Mihael Pranjić)_
- Filter empty meta KeySets to save space. _(Mihael Pranjić)_

### 425.3.6 Noresolver

- The plugin now correctly sets the path in the `parentKey`. It therefore now supports set calls. _(Klemens Böswirth)_

### 425.3.7 Path

- The `Markdown Shell Recorder` test of the plugin now also works, if you execute it as root user. _(René Schwaiger)_

### 425.3.8 Spec

- There is now the config key `missing/log` that allows logging of all missing `required` keys. _(Klemens Böswirth)_
- `spec` now internally handles errors differently. There should be no external impact apart from better performance. _(Klemens Böswirth)_

### 425.3.9 Specload

- We now treat relative paths as relative to `KDB_DB_SPEC` instead of the current working directory. _(Klemens Böswirth)_
- Changes to `default` or `type` metadata are no longer supported, since they are not safe in every case. _(Klemens Böswirth)_
- The plugin no longer has the `experimental` status. _(Klemens Böswirth)_

### 425.3.10 Tcl

- We made sure that building the plugin works, if you use the latest version of CMake (`3.15.3`) and Boost (`1.71`). _(René Schwaiger)_

### 425.3.11 Type

- We added an option to disable the restoring of boolean values. This useful for storage formats like YAML that have native boolean types. _(Klemens Böswirth)_

### 425.3.12 Yajl

- Yajl now correctly supports Elektras boolean types using the `type` plugin. For example, setting `on`, `enable` or `true` all map to JSONs native `true` value. See the type plugin for more details about boolean types. _(Philipp Gackstatter)_

### 425.3.13 YAwn

- We removed the plugin in favor of Yan LR. _(René Schwaiger)_

### 425.3.14 YAy PEG

- We removed the plugin in favor of Yan LR. _(René Schwaiger)_

## 425.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 425.4.1 Compatibility

We introduced several incompatible changes:

- The library `libelektra`, which is a collection of different elektra libraries, is now removed. Users of CMake or pkg-config should not be affected. Otherwise change `-lelektra` to `-lelektra-core -lelektra-kdb` or whatever parts of Elektra your application uses. _(Markus Raab)_
- The conversion functions `elektraKeyTo*` and `elektra*ToString` are now part of the `elektra-ease` library instead of the `elektra-highlevel` library. This should not cause any breaking changes since `elektra-highlevel` already depends on `elektra-ease`. In addition the header `elektra/conversion.h` is kept for compatibility. _(Klemens Böswirth)_
- Fixes in documentation that might disallow some code operating in grey areas before. _(Markus Raab)_
- We removed `keyRel` and `keyRel2` since it can be easily replaced by other existing functions. _(Philipp Gackstatter)_

We changed the following symbols:

- ELEKTRA_PLUGIN_COMMIT
- elektraKeyToString
- elektraKeyToBoolean
- elektraKeyToChar
- elektraKeyToOctet
- elektraKeyToShort
- elektraKeyToUnsignedShort
- elektraKeyToLong
- elektraKeyToUnsignedLong
- elektraKeyToLongLong
- elektraKeyToUnsignedLongLong
- elektraKeyToFloat
- elektraKeyToDouble
- elektraKeyToLongDouble
- elektraBooleanToString
- elektraCharToString
- elektraOctetToString
- elektraShortToString
- elektraUnsignedShortToString
- elektraLongToString

- elektraUnsignedLongToString
- elektraLongLongToString
- elektraUnsignedLongLongToString
- elektraFloatToString
- elektraDoubleToString
- elektraLongDoubleToString
- kdb_octet_t
- kdb_boolean_t
- kdb_short_t
- kdb_long_t
- kdb_long_long_t
- kdb_unsigned_short_t
- kdb_unsigned_long_t
- kdb_unsigned_long_long_t
- kdb_char_t
- kdb_float_t
- kdb_double_t
- kdb_long_double_t

### 425.4.2 Core

- A new plugin function, `kdbCommit`, was implemented. The function is carried out in the `commit` phase of `kdbSet` and separates the commit functionality from the `kdbSet()` function. _(Vid Leskovar)_
- `kdbconfig.h` is no longer included in the installed headers. This is because it could cause conflicts with other `config.h`-type headers from applications. _(Klemens Böswirth)_
- `ksAppendKey`: state that it only fail on memory problems. _(Markus Raab)_
- Fix memory leak in `kdbGet`. _(Markus Raab)_
- Implemented `kdberrors.h` directly without generation of the `specification` file because of drastically reduced error code count _(Michael Zronek)_
- `keyIsDirectBelow` was renamed to `keyIsDirectlyBelow`. _(Philipp Gackstatter)_
- `keyMeta` was added to provide access to a key's underlying KeySet that holds its metadata keys. _(Philipp Gackstatter)_
- Removed the obsolete `ksLookupByString` and `ksLookupByBinary`, as well as deprecated `KDB_O_*` options. _(Philipp Gackstatter)_
- Added `keyLock` and `keyIsLocked`. _(Manuel Mausz)_
- Removed `keyVInit`. _(Manuel Mausz)_

### 425.4.3 Opts

- The option `-h` is no longer used to indicate help mode. Only `--help`, will invoke help mode. _(Klemens Böswirth)_

### 425.4.4 Proposal

- Removed or moved several functions of kdbproposal.h:
  - elektraKsToMemArray was moved to kdbease.h,
  - elektraLookupOptions was moved to kdbprivate.h,
  - keySetStringF was moved to kdbinternal.h,
  - Removed ksPrev and elektraKsPrev,
  - Removed elektraRenameKeys and replaced it with ksRenameKeys. _(Philipp Gackstatter)_

## 425.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

- We removed the Haskell and GI bindings. _(Markus Raab)_
- Avoid unnecessary copying std::string where possible (setString and setMeta only). _(Manuel Mausz)_
- CPP: add Key::is{Name,Value,Meta}Locked. _(Manuel Mausz)_
- GLIB: removed gelektra_keyset_resize. _(Manuel Mausz)_
- GLIB: removed gelektra_keyset_rewind, gelektra_keyset_next, gelektra_keyset_current, gelektra_keyset_getcursor, gelektra_keyset_setcursor. _(Manuel Mausz)_
- GLIB: renamed gelektra_keyset_atcursor to gelektra_keyset_at. _(Manuel Mausz)_
- gsettings: adapt iterator. _(Manuel Mausz)_
- SWIG: Add KeyNotFoundException exception. _(Manuel Mausz)_
- SWIG: Fix KeySet equality operators. _(Manuel Mausz)_
- SWIG/Python: hash(key) will throw unless the key name is locked. _(Manuel Mausz)_
- SWIG/Python: Add operator overloadings for len(Key), repr(Key), str(KeySet), repr(KeySet). _(Manuel Mausz)_
- SWIG/Python: Add alternative Key constructor Key(name, value, [dict]). _(Manuel Mausz)_
- SWIG/Python: Add KeySet.extend([key1, key2,...]) and KeySet.append(key1, key2, ...). _(Manuel Mausz)_
- SWIG/Python: Add KeySet.append(key_name, key_value, key_opts) which directly creates and appends a key. _(Manuel Mausz)_
- SWIG/Python: Add KeySet.unpack_names(), KeySet.unpack_basenames(), KeySet.filter(func), KeySet.filter_below(where), Key.array_elements(). _(Manuel Mausz)_

### 425.5.1 Java

- Completely overhauled the Java binding to be able to use Elektra plugins directly. A new PluginLoader can load Elektra plugins or a native implemented Java plugin. All Plugins now implement the new Plugin interface. For an example see the test case. _(Michael Zronek)_
- Upgraded maven dependencies for Java binding _(Michael Zronek)_
- The Java binding now supports the error codes in a native way. All exceptions contain the necessary information. _(Michael Zronek)_
- Further improved the Java binding such as wording and documentation. _(Michael Zronek)_

### 425.5.2 Rust

- Add the `elektra-sys` crate which contains raw bindings to libelektra for Rust. _(Philipp Gackstatter)_
- Add the `elektra` crate which contains safe wrapper methods for the raw bindings. The crate contains bindings for the low-level API, which means that the data types `Key` and `KeySet` can now safely be used from Rust. The Rust version of the API has been designed to take advantage of Rust's type system and to be in accordance with the memory safety of Rust. For instance, the Key has been divided into `StringKey` and `BinaryKey`, to prevent type mismatches at compile-time. With the binding for `KDB`, one can take advantage of the elektra ecosystem from Rust. See the Readme for more. _(Philipp Gackstatter)_
- Rewrote the `KDBError` to follow the specification fully and in particular allow catching out of memory errors by catching resource errors. _(Philipp Gackstatter)_
- Added a `keyset!` macro to easily create a keyset with many keys in a single invocation. _(Philipp Gackstatter)_

## 425.6 Tools

- `KDB_EXEC_PATH`, which can be used to add further external tools to `kdb`, now supports `:` to separate paths. `kdb list-tools` and `run_env` were improved to take advantage of this. _(Markus Raab)_
- Checks for `kdbCommit` have been added to kdb plugin-check. _(Vid Leskovar)_
- Added PID file config setting for kdb-run-rest-frontend _(Markus Raab)_
- Added `kdb meta-show` command which prints out all metadata along with its values for a given key. _(Michael Zronek)_
- Removed `kdb vset` as it does not properly put metadata to the spec namespace. _(Michael Zronek)_
- Renamed kdb plugin commands following a hierarchical structure. `kdb info` is now `kdb plugin-info`, `kdb check` is now `kdb plugin-check` and `kdb list` is now `kdb plugin-list`. We also removed the obsolete `kdb fstab`. _(Philipp Gackstatter)_
- Renamed kdb meta commands:
  - `kdb getmeta` is now `kdb meta-get`
  - `kdb lsmeta` is now `kdb meta-ls`
  - `kdb showmeta` is now `kdb meta-show`
  - `kdb rmmeta` is now `kdb meta-rm`
  - `kdb setmeta` is now `kdb meta-set` _(Philipp Gackstatter)_
- Fix test tool `gen-gpg-testkey` by giving a narrower GPG key description. Fixes mismatches with existing GPG keys that contain "elektra.org" as e-mail address. _(Peter Nirschl)_
- `kdb list-commands` and `kdb plugins-list` now sort their output in an alphabetical order _(Anton Höß l)_
- `kdb plugin-list` does now mention in the helptext that with option `-v` the output is sorted by the plugin status _(Anton Höß l)_
- `kdb import`, `kdb export` and `kdb editor` now search the plugin database for suitig plugins so it's now possible to run `kdb export /hello json` instead of having to specify the plugin for the desired format directly. _(Anton Höß l)_
- `get` and `mount`: Remove dependency on `kdbprivate.h` _(Philipp Gackstatter)_

## 425.7 Scripts

- We structured the scripts. _(Markus Raab)_
- Removed the scripts
  - `scripts/elektra-merge`,
  - `scripts/elektra-mount`,
  - `scripts/elektra-umount`,
  - `convert-fstab`,
  - `convert-hosts`,
  - `convert-inittab`,
  - `convert-users`,
  - `scripts/benchmark_libsplit.sh`,
  - `scripts/zsh` and
  - `example-xorg`. _(Markus Raab)_
- Renamed `scripts/run_dev_env` to `scripts/dev/run_env`. _(Markus Raab)_
- The script `draw-all-plugins` now also works properly, if the repository path contains space characters. _(René Schwaiger)_
- The script `link-checker` now deduplicates the list of links before checking them. The timeout and amount of retries was also reduced. Lastly the script now supports a whitelist. Any link stored in `tests/linkchecker.whitelist` will not be checked. _(Klemens Böswirth)_
- We removed a script used to compare the runtime performance of YAML plugins. _(René Schwaiger)_
- Cleanup: separation of dev, admin and completion scripts. _(Markus Raab, Rene Schwaiger)_
- Pre-commit hook `pre-commit-check-formatting` now lives in `scripts/dev/pre-commit-check-formatting`. _(Klemens Böswirth)_
- The new script `reformat-javascript` formats the JavaScript code of the repository using the tool `prettier`. _(René Schwaiger)_
- We renamed
  - the script `reformat-source` to `reformat-c`, and
  - the script `reformat-shfmt` to `reformat-shell`. _(René Schwaiger)_
- The script cmake-format now requires `cmake-format` 0.6. _(René Schwaiger)_
- The new script `reformat-java` formats the Java code in the repository using `clang-format`. _(René Schwaiger)_
- The `Markdown Shell Recorder` now also works correctly on FreeBSD. _(René Schwaiger)_

## 425.8 Documentation

- Added a tutorial on how to write language bindings. Visit our new README. _(Michael Zronek, Raphael Gruber, Philipp Gackstatter)_
- Clarified subtyping in the language bindings tutorial. _(Michael Zronek)_
- A second tutorial on writing bindings for the high-level API was created as well. _(Klemens Böswirth, Raphael Gruber)_
- Added info on how to include xerces plugin with homebrew installation. _(Anton Hößl)_

- The compile instructions do not assume that you use `make` or `gcc` to build Elektra anymore. _(René Schwaiger)_
- Add hints about reformatting with docker. _(Dominic Jäger)_
- Add instructions about sourcing on FreeBSD. _(Dominic Jäger)_
- Add information on debuggers to main testing documentation. _(Dominic Jäger)_
- Added design decision for error code implementations. _(Michael Zronek)_
- Fixed some typos and links in the documentation and add new iterate example. _(Philipp Gackstatter)_
- Clarified warnings metadata in the error-handling guideline. _(Michael Zronek)_
- We fixed minor spelling mistakes in the documentation. _(René Schwaiger)_
- Corrected buildserver documentation. _(Djordje Bulatovic)_
- Add merge library into kdbset example. _(Dominic Jäger)_
- We updated links for the INI parsing library Nickel. _(René Schwaiger)_
- Added some information about contributing to Elektra. _(Hani Torabi)_

## 425.9 Tests

- We changed how the `formatting test` detects code differences. This update should get rid of transient errors as `reported here`. _(René Schwaiger)_
- We disabled the test for the conversion engine. For more information, please take a look at `issue #3086`. _(René Schwaiger)_
- We disabled the test `testmod_zeromqsend` from the command `kdb run_all`, since it caused timeouts in high load scenarios. _(Mihael Pranjić)_
- The (Markdown) Shell Recorder now prints the protocol for a failed test, even if the test modified the database permanently. _(René Schwaiger)_
- We rerun ctest twice to ignore temporary build failures. _(Markus Raab)_

## 425.10 Build

### 425.10.1 CMake

- `kdbtypes.h` is now generated directly via a CMake `configure_file` call. _(Klemens Böswirth)_
- The variable `ELEKTRA_STAT_ST_SIZE_F` now contains the correct format specifier for the `st_size` member of the `stat` struct on macOS. _(René Schwaiger)_
- We simplified and unified the CMake code for the Shell Tests and the Shell Recorder. _(René Schwaiger)_
- CMake now prints warnings about missing man pages. _(René Schwaiger)_
- The build system does not update a man page in the folder doc/man any more, if ronn only changed the creation date of the man page. _(René Schwaiger)_

### 425.10.2 Compilation

- We now have a setup for proper symbol versioning. _(Klemens Böswirth)_
- We do not use implicit typing in the code of the `conditionals` and `yamlcpp` plugin any more. After this update, the code compiles without any warnings, even though we now use the compiler switch `-Wconversion`. _(René Schwaiger)_
- JNA and JNI are not built concurrently anymore to avoid `dependency resolution fails`. _(↩ Michael Zronek)_

### 425.10.3 Docker

- Added Dockerfile for Ubuntu Bionic _(Djordje Bulatovic)_
- We removed all Haskell packages from the Dockerfiles in the folder scripts/docker. _(René Schwaiger)_
- We added a basic Dockerfile for Arch Linux. _(René Schwaiger)_
- We updated the Dockerfile for Alpine Linux. _(René Schwaiger)_

### 425.10.4 Vagrant

- We added a Vagrantfile for a virtual machine based on FreeBSD 12. _(René Schwaiger)_

### 425.10.5 Other

- The reformatting script now checks that the correct version of `cmake-format` is used. _(Klemens Böswirth, René Schwaiger)_
- Improved various error messages and synchronized documentations. _(Michael Zronek)_
- Improved `range` plugin error message. _(Michael Zronek)_
- Improved error codes documentation to clarify the hierarchy for developers. _(Michael Zronek)_
- Release notes now use git's union merge driver. _(Dominic Jäger)_
- Updated asciinema recording to correctly use `sudo`. _(Michael Zronek)_
- Add pkg-config file for kdbmerge. _(Raphael Gruber)_

## 425.11 Infrastructure

### 425.11.1 Cirrus

- The `link symbol Check` build job now merges PRs before checking links. _(Klemens Böswirth)_
- We enabled logging in the build job `red apple Clang`. This update makes sure that Elektra's logging code compiles without warnings on macOS. _(René Schwaiger)_
- All macOS build jobs now use Xcode `11.1` instead of Xcode `10.1`. _(René Schwaiger)_
- We removed all non-POSIX shell code from the Cirrus configuration file. _(René Schwaiger)_
- The macOS build jobs now use Ruby `2.6`. _(René Schwaiger)_
- We do not call `ninja` directly anymore. Instead we use `cmake --build`. This has the advantage that we do not have to care about the Generator used by CMake. _(René Schwaiger)_
- We added the build job `smiling face with horns ASAN`, which builds and executes Elektra on FreeBSD with enabled `AddressSanitizer`. _(René Schwaiger)_
- We now store common commands in one place at the top of the configuration file. This makes it easier to selectively disable certain build jobs. _(René Schwaiger)_
- The new job `books Check` checks
  - that the man pages are up-to-date, and
  - that building the PDF version of the Doxygen documentation works. _(René Schwaiger)_
- The new build job `penguin Fedora` builds and tests Elektra on Fedora Linux. _(René Schwaiger)_

### 425.11.2 Jenkins

- We upgraded all servers to Buster. _(Markus Raab)_
- Jenkins does not auto cancel build jobs of the `master` branch anymore. _(René Schwaiger)_
- Updated xUnit plugin in Jenkinsfile. _(Djordje Bulatovic)_

### 425.11.3 Restyled

- `Restyled` now also formats Markdown files with `prettier`. _(René Schwaiger)_

### 425.11.4 Travis

- The build job `green apple GCC` now uses the `Travis Homebrew addon` to install dependencies. _(René Schwaiger)_
- We now build and test Elektra on Ubuntu `18.04` (Bionic Beaver) instead of Ubuntu `16.04` (Xenial Xerus). _(René Schwaiger)_

## 425.12 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Renamed the sub menu *Getting Started* in the menu *Documentation* to *About Elektra* and removed the sub-sub-menus *Compiling*, *Installation* and *Tutorials*. These sub-sub-menus are now sub-menus of *Documentation*. A new sub menu labeled *Get Started* added to the menu *Documentation* with some newcomer-friendly information. Renamed the *Getting Started* sub menu in *Development* to *Contribute to Elektra*. The green button on the main page is routed to the new *Get Started* page. _(Hani Torabi)_
- The website now lives in the folder website to avoid confusion with the REST backend of the Web-UI. _(↩ Markus Raab)_
- Improve main page of website, restructure getting started. _(Markus Raab)_

## 425.13 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_ and _(Felix Resch)_
- Elektrify LCDproc _(Klemens Böswirth)_ and _(Jakob Fischer)_
- Reduce the number of merge conflicts in 3-way merge. _(Dominic Jäger)_
- Go bindings and improved Web-UI _(Raphael Gruber)_
- Improved Error handling _(Michael Zronek)_
- New default storage _(René Schwaiger)_ and _(Jakob Fischer)_
- Shell completion _(Ulrike Schäfer)_
- Improve Elektra developer experience _(Hani Torabi)_
- Ansible bindings _(Thomas Waser)_
- Plugin interface improvements _(Vid Leskovar)_

## 425.14 Statistics

We closed 75 issues for this release.
About 29 authors changed 1651 files with 82267 insertions(+) and 41690 deletions(-) in 1623 commits.
Thanks to all authors for making this release possible!

## 425.15 Finished Thesis

- Klemens Böswirth: We explore the feasibility of using Elektra in a real-world project. We focused especially on using the high-level API with code-generation. In the thesis, we implemented new versions of LCDproc, one with the low-level API and one with the high-level API. Then we did some benchmarks to compare them. Our results indicate, that Elektra is very much usable in real-world projects. However, we also found that there is still potential for further optimizations.
- Mihael Pranjić: We design a binary storage format for Elektra's data structures and implement a cache plugin based on the format. The cache plugin leverages the `mmap()` system call to read and write data. Our cache implementation for Elektra's data structures makes the access to the KDB more than 80 times faster for larger data sets. The penalty for cache misses is below 20 percent for reasonably sized data sets.

## 425.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 425.17 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- author: mpranj
- file: elektra-0.9.1.tar.gz
- size: 7534156
- md5sum: 42ff587adb7c3f15807ac4dae6722261
- sha1: bf250260a4efa20e5444f0a7f0027430bc7aa8a0
- sha256: df1d2ec1b4db9c89c216772f0998581a1cbb665e295ff9a418549360bb42f758

The release tarball is also available signed by Mihael Pranjic using GnuPG from here or on GitHub
Already built API documentation can be found here or on GitHub.

## 425.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 426

# 0.9.2 Release

- guid: ea79f59e-f471-4658-a11b-1371802814c2
- author: Mihael Pranjić
- pubDate: Tue, 26 May 2020 19:33:30 +0200
- shortDesc: KDE and GNOME Integration, `elektrad` in Go

We are proud to release Elektra 0.9.2.
With the 0.9.x series of releases we shift our focus to bugfixes and stability improvements as needed for the KDE and GNOME integration. We do not guarantee any compatibility in this series.

## 426.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`

## 426.2 Highlights

- KDE integration
- GNOME Integration
- `elektrad` rewritten in Go

### 426.2.1 KDE Integration

We created a `fork` of `KDE's` `KConfig` configuration system and patched it to use libelektra. We have done some initial testing and replaced the `KConfig` library for `Kate` and `KDevelop` successfully.
Additionally, we added a new Elektra plugin called `kconfig`, which can read and write kconfig's INI files. The plugin enables smooth migration of existing KDE configurations. _(Dardan Haxhimustafa)_ and _(Felix Resch)_

### 426.2.2 GNOME Integration

We continued work on Elektra's bindings for `GNOME GSettings`. Our implementation should be able to replace the widely used `dconf` backend. Elektra's `gsettings` bindings are not yet ready for production use, but they are already able to replace `dconf` for a complete GNOME session without problems. We are still lacking proper dbus integration for change notifications. _(Mihael Pranjić)_

### 426.2.3 <tt>elektrad</tt> rewritten in Go

`elektrad` provides an HTTP API to access Elektra remotely. `elektrad` is now completely rewritten in Go, which drastically improves the performance by leveraging the new `go-elektra` bindings instead of calling the `kdb` command-line tool on every request. The new `elektrad` creates a session per user to reuse the same `KDB handle` for correct conflict handling and better performance. _(Raphael Gruber)_

## 426.3 Try out Elektra

You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles.
Get started with Elektra by running `docker run -it elektra/elektra`.

## 426.4 Plugins

We removed the `maintained` status of the following `plugins`:

- blockresolver
- csvstorage
- gitresolver
- list
- multifile
- spec

New maintainers are very much welcomed!

### 426.4.1 Augeas

- Improved error message for Augeas to show lensPath. _(Michael Zronek)_

### 426.4.2 CCode

- The `Markdown Shell Recorder` test of the plugin does not require Bash any more. _(René Schwaiger)_

### 426.4.3 Crypto

- The crypto plugin no longer supports Botan and OpenSSL as provider of cryptographic functions. The support has been removed to improve the maintainability of the code. _(Peter Nirschl)_
- The unit test of the crypto plugin attempts to kill the gpg-agent if a regular shutdown via `connect-gpg-agent` failed. _(Peter Nirschl)_

### 426.4.4 Directory Value

- The plugin now only interprets a `KeySet` as array if the parent contains the metakey `array`. _(René Schwaiger)_

### 426.4.5 Fcrypt

- Improve handling of temporary files after encryption and decryption by trying to perform a manual copy if the call of `rename` fails. This problem might occur if another file system is mounted at `/tmp`. _(Peter Nirschl)_

### 426.4.6 KConfig

- Write support for the KConfig INI format was added. _(Dardan Haxhimustafa)_

### 426.4.7 SWIG

- Configure line (-DBINDINGS="..") for SWIG based bindings have been changed from `swig_foo` to `foo`. _(Manuel Mausz)_
- Exclude SWIG bindings if SWIG Version is 4.0.1 and Python is >= 3.8 or Ruby is >= 2.7 due to incompatibility (#3378, #3379). _(Mihael Pranjić)_

### 426.4.8 SWIG/python

- Added bindings for libelektratools. _(Manuel Mausz)_
- Add test for kdbEnsure. _(Mihael Pranjić)_

### 426.4.9 SWIG/python2

- Removed python2 binding, as python2 support ended. _(Manuel Mausz)_

### 426.4.10 Tcl

- The `Markdown Shell Recorder` test of the plugin now correctly requires the `xmltool` plugin. _(René Schwaiger)_

### 426.4.11 YAMBi

- We removed the plugin in favor of Yan LR. _(René Schwaiger)_

### 426.4.12 YAML CPP

- The plugin now always prints a newline at the end of the YAML output. _(René Schwaiger)_
- The plugin does not interpret a key set such as

```
user/example
user/example/#0
user/example/#1
user/example/#2
```

as array unless the parent key `user/example` contains the metakey `array`. _(René Schwaiger)_

- YAML CPP now always sets and requires the metakey `type` with the value `boolean` for boolean data. _(René Schwaiger)_
- We limited the scope of a logging function of the module. This makes it possible to build Elektra again, if
  - you enabled the logger (`ENABLE_LOGGER=ON`),
  - build the “full” (`BUILD_FULL=ON`) version of Elektra, and
  - include both the Directory Value and YAML CPP plugin in your build configuration. _(René Schwaiger)_

### 426.4.13 Yan LR

- The CMake code of the plugin does not print error messages produced by the tool `ldd` any more. _(René Schwaiger)_
- The plugin now also supports ANTLR 4.8. _(René Schwaiger)_
- We limited the scope of the logging code of the module. For more information, please take a look at the last news entry of the YAML CPP plugin. _(René Schwaiger)_

### 426.4.14 GOpts

- The plugin now supports an offset into `argv` given by the `/offset` config key. If `/offset` is set, `gopts` will ignore a number of arguments at the start of `argv`. This can be used in e.g. python scripts to ignore the interpreter arguments. _(Klemens Böswirth)_
- `gopts` now also writes help message into the key `proc/elektra/gopts/help/message` in addition to setting `proc/elektra/gopts/help = 1`. This is especially useful in non-C/C++ environments. _(Klemens Böswirth)_
- `gopts` is also affected by the changes and improvements to the `opts` library outlined below.

### 426.4.15 Cache

- Respect `XDG_CACHE_HOME` when resolving the mmap cache directory. _(Mihael Pranjić)_

## 426.5 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 426.5.1 Compatibility

- We clarified compatibility requirements for Elektra and its plugins and bindings. Furthermore, we renamed `system/elektra/version/constants/KDB_VERSION_MICRO` to `system/elektra/version/constants/KDB_VERSION_PATCH` to be compatible with `Semantic Versioning 2.0.0`. _(Markus Raab)_

### 426.5.2 Opts

- The library function `elektraGetOpts` now supports sub-commands. Sub-commands are best explained by looking at an application that uses them, like `git`. For example `add` is a sub-command in `git add`, and interprets `-p` differently from `git`: `git -p add` is `git --paginate add`, but `git add -p` is `git add --patch`. `elektraGetOpts` now implements this notion of sub-commands. For more information take a look at the tutorial for command-line-options. By extension this functionality is also available via the `gopts` plugin. _(Klemens Böswirth)_
- The generated help message was improved. It now also gives details about parameter arguments, sub-commands and environment variables in addition to the existing support for option arguments. This also means that it is no longer possible to have multiple keys with the `args=remaining` metadata (because their `opt/help` may not be the same). _(Klemens Böswirth)_

## 426.6 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 426.6.1 python2

- Removed python2 plugin, as python2 support ended. _(Manuel Mausz)_

### 426.6.2 Rust

- Published `elektra` and `elektra-sys` to crates.io. _(Philipp Gackstatter)_

## 426.7 Tools

- Update `kdb cache` tool synopsis to reflect man page. _(Mihael Pranjić)_

- Pull elektrad, webui and webd out of shared web folder to allow fine grained selection of tools. _(Raphael Gruber)_
- webd has updated dependencies. _(Mihael Pranjić)_

## 426.8 Scripts

- The fish completion script now recognizes the new names of subcommands (e.g. `meta-set` instead of `setmeta` ) introduced with Elektra `0.9.1`. _(René Schwaiger)_
- The script reformat-cmake now reformats the code with `cmake-format` 0.6.3. _(René Schwaiger)_
- The scripts
  - reformat-c, and
  - reformat-java

  now uses `clang-format` 9 to reformat the code base. _(René Schwaiger)_
- The script reformat-shell now makes sure that you do not use `shfmt` 3, which formats parts of the code base slightly differently. _(René Schwaiger)_

## 426.9 Documentation

- Improved formatting of the `validation tutorial` _(Anton Hößl)_
- We fixed some minor spelling mistakes. _(René Schwaiger)_
- We updated the man pages of the `web` tool. _(René Schwaiger)_
- Updated documentation for Ubuntu-Bionic Packages. _(Djordje Bulatovic)_
- Fixed an old path of the reformatting script in the `docker reformatting tutorial` _(Jakob Fischer)_

## 426.10 Tests

- We now use `Google Test` version `1.10` to test Elektra. _(René Schwaiger)_
- The C++ test code does not produce warnings about a missing macro argument for `...` any more. _(René Schwaiger)_
- Whitelisted many broken links. _(Mihael Pranjić)_
- Enabled regex in link checker. _(Mihael Pranjić)_
- The formatting check now also works correctly, if it is invoked multiple times. _(René Schwaiger)_
- `KDB_EXEC_PATH` is not being set globally to contain the build directory any longer. _(Peter Nirschl)_
- Rewrite gpg-agent shutdown logic to use `fork` and `execv` instead of `system`. _(Peter Nirschl)_
- Removed a broken link from the link checker. _(Djordje Bulatovic)_

## 426.11 Build

### 426.11.1 Compilation

- We do not use implicit typing in the code of the
  - `augeas`,
  - `base64`, and
  - `blockresolver`

  plugin any more. After this update, the code compiles without any warnings, even though we now use the compiler switch `-Wconversion`. _(René Schwaiger)_

### 426.11.2 Support

- Debian 9 “stretch” (oldstable) is now the oldest supported platform. _(René Schwaiger)_

### 426.11.3 CMake

- Generating the build system now requires CMake `3.4` (released in November 2015). _(René Schwaiger)_
- We fixed warnings about CMake policy `CMP0078` and `CMP0086`. _(René Schwaiger)_
- The CMake functions `add_msr_test` and `add_msr_test_plugin` do not export the list of required plugins as environment variable any more. _(René Schwaiger)_
- The CMake code of the code generation does not print warnings about unknown regex operators any more. _(René Schwaiger)_

### 426.11.4 Docker

- We updated some of the software in the Dockerfile for Debian sid. _(René Schwaiger)_
- Building the documentation Dockerfile for Debian Stretch works again. _(René Schwaiger)_
- Use Python 3, SWIG 4.0 and Ruby 2.5 in the Dockerfile for Debian sid. _(Mihael Pranjić)_
- Disable python binding on `debian-unstable-full-clang` due to upstream `issue`. _(Mihael Pranjić)_
- Use current ruby-dev on Debian sid image as Ruby 2.5 has been dropped. _(Mihael Pranjić)_

## 426.12 Infrastructure

### 426.12.1 Cirrus

- We fixed a minor problem with the package install procedure on macOS build jobs. _(René Schwaiger)_
- We updated the startup command for D-Bus on macOS. _(René Schwaiger)_
- We removed python2 (EOL and removed from homebrew). _(Mihael Pranjić)_
- Use latest macOS Catalina Xcode stable. _(Mihael Pranjić)_
- Use newer FreeBSD images and use image family instead of concrete image names. _(Mihael Pranjić)_
- Disable tcl plugin on FreeBSD images because of test failures (see #3353). _(Mihael Pranjić)_
- Disable curlget plugin for macOS jobs (see #3382). _(Mihael Pranjić)_
- Add more dependencies to Fedora image to cover many tests. _(Mihael Pranjić)_
- Installed Ruby 2.6 to test the ruby bindings and plugins. _(Mihael Pranjić)_
- Upgraded Fedora image to current stable (version 32). _(Mihael Pranjić)_

### 426.12.2 Jenkins

- Fixed `coveralls` coverage report. _(Mihael Pranjić)_
- The build jobs `debian-unstable-clang-asan` and `debian-unstable-full-clang` now use Clang 9 to compile Elektra. _(René Schwaiger)_
- Added the `Jenkins.monthly` in the Jenkins' scripts file. _(Djordje Bulatovic)_
- Enabled building packages for Bionic. _(Djordje Bulatovic)_
- Improve gpgme unit test stability. _(Peter Nirschl)_

- Publishing packages for Bionic to community. _(Djordje Bulatovic)_
- Added Fedora 32 image to main build stage, moved Fedora 31 to full build stage. _(Mihael Pranjić)_
- Fixed path for publishing in Jenkinsfile. _(Djordje Bulatovic)_
- Reliably build the rust bindings based on the same version, by adding back the `Cargo.lock` file. _(Philipp Gackstatter)_

### 426.12.3 Restyled

- Restyled now also checks the formatting of C, C++ and Java code in the repository. _(René Schwaiger)_

### 426.12.4 Travis

- Use newer Xcode 11.4 and ruby 2.6.5 on macOS builds and use macOS 10.15. _(Mihael Pranjić)_
- Disable curlget plugin for macOS jobs (see #3382). _(Mihael Pranjić)_

### 426.12.5 Issue Tracker

- We now automatically close issues after one year of inactivity. _(Mihael Pranjić)_

### 426.12.6 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Fix and re-enable website auto-deployment. _(Mihael Pranjić)_
- Update docker images for website frontend and backend to debian buster. Update dependencies to newer versions. _(Mihael Pranjić)_
- Remove obsolete parts from the website. _(Mihael Pranjić)_

## 426.13 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- Elektrify GNOME _(Mihael Pranjić)_
- Elektrify LCDproc _(Klemens Böswirth)_ and _(Jakob Fischer)_
- Packaging for popular Linux distributions _(Djordje Bulatovic)_
- Improve 3-way merge. _(Dominic Jäger)_
- Go bindings and improved Web-UI _(Raphael Gruber)_
- TOML plugin as new default storage _(Jakob Fischer)_
- Shell completion _(Ulrike Schäfer)_
- Improve Elektra developer experience _(Hani Torabi)_
- Ansible bindings _(Thomas Waser)_
- Plugin interface improvements _(Vid Leskovar)_

## 426.14 Statistics

We closed 40 issues for this release.
About 23 authors changed 653 files with 15221 insertions(+) and 18890 deletions(-) in 815 commits.
Thanks to all authors for making this release possible!

## 426.15 Finished Thesis

René Schwaiger finished his thesis about parsing techniques and parsing tools for configuration files.

## 426.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 426.17 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- author: mpranj
- file: elektra-0.9.2.tar.gz
- size: 7416188
- md5sum: 6e92ebcbef31cdeab91d228b61456947
- sha1: 8f874de3e7a47baa55d7c5106efbbae635fff499
- sha256: 6f2fcf8aaed8863e1cc323265ca2617751ca50dac974b43a0811bcfd4a511f2e

The release tarball is also available signed by Mihael Pranjic using GnuPG from here or on GitHub.
The following GPG Key was used to sign this release: 9C18145C22F9E746D743DEC59ECC0F4CF0359C7B
Already built API documentation can be found here or on GitHub.

## 426.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 427

# 0.9.3 Release

- guid: 9b9e8889-0c77-45ce-8441-8bdf26a110ac
- author: Mihael Pranjić
- pubDate: Fri, 30 Oct 2020 23:03:43 +0100
- shortDesc: TOML Storage Plugin

We are proud to release Elektra 0.9.3. This release again brings us a big step towards Elektra 1.0. It introduces the new soon-to-be-default storage plugin: TOML.

## 427.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run `docker run -it elektra/elektra`.

## 427.2 TOML

The highlight of this release is the `TOML plugin`. The TOML plugin has a similar huge feature set as the INI plugin, but is written in a much cleaner and more maintainable way using flex and bison. Furthermore, it follows the popular `TOML spec`.
It has nearly no run-time dependency, it only needs `base64` if binary values are needed.
A huge thanks to Jakob Fischer for this amazing work!

> Warning: In one of the following `0.9.*` releases, INI will be removed and TOML will become the default plugin. If you are using INI, please migrate to TOML now.

## 427.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 427.3.1 xmltool

- Values of KDBStream changed to fit with elektraKeyFlags: recompilation of application is needed. _(Markus Raab)_

### 427.3.2 TOML

- Added the TOML plugin, which can read and write TOML files using flex and bison. _(Jakob Fischer)_
- Removed the null plugin dependency of the plugin. _(Jakob Fischer)_
- The type metakey is now set for numbers on reading. _(Jakob Fischer)_

### 427.3.3 dump

- The dump plugin got a major update. The new version can read old files, but only write new files. The new files **cannot** be read by the old version of the plugin and will result in a "version error" message. _(Klemens Böswirth)_
- The new version stores keynames relative to the mountpoint, so exported dump files can now be imported into a different mountpoint. _(Klemens Böswirth)_
- dump no longer writes unnecessary zero-bytes into files. This means that as long as all key-values are human-readable, so is the dump output. This makes dump usable for tests and demo purposes, as it is a very simple format closely modelled after a KeySet's structure. This also makes it much easier to manually fix broken dump files. You only need a text editor most of the time. _(Klemens Böswirth)_

## 427.4 Compatibility

Elektra 0.9.* does not make any compatibility promises, as we want a clean 1.0.0 release. In this release, we did the following changes:

- keyswitch_t renamed to elektraKeyFlags. _(Markus Raab)_
- option_t renamed to elektraLockFlags and elektraLookupFlags. _(Markus Raab)_
- cursor_t renamed to elektraCursor. _(Markus Raab)_

  Note: We made this release before merging larger changes. Please expect more fundamental changes in the next releases.

### 427.4.1 Errors

- Improved error messages. _(Markus Raab)_

## 427.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 427.5.1 JNA

- Made examples work again. _(Markus Raab)_

## 427.6 Scripts

- Fix googletest framework path in debian configure script. _(Mihael Pranjić)_
- The fish completion script does not fail any more, if you try to complete a simple command that expects a namespace. For example, completing

```
kdb ls ^
```

with the tab key (⇥) (^ represents the current cursor position) should work correctly again. _(René Schwaiger)_

- The formatting scripts using prettier now use npx, which is included in npm v5.2.0 or newer. This allows us to specify an exact version of prettier to use. _(Klemens Böswirth)_

- The `reformat-shell` script now accepts the alias `shfmt2`, in case `shfmt` v3 is needed for another project. _(Klemens Böswirth)_
- The config for restyled.io now specifies the image to use for each formatter. This clearly reflects the version used for formatting. _(Klemens Böswirth)_
- Add a release script to automate the release process. _(Robert Sowula)_

## 427.7 Documentation

- Updated `GOALS.md`. _(Markus Raab)_
- Describe hierarchy and limitations of `hosts plugin`. _(Markus Raab)_
- The Doxygen PDF documentation now also requires the packages
  - `stix` (part of `texlive-fonts-extra`) and
  - `stmaryrd` (part of `texlive-science` or `texlive-math-extra`). _(René Schwaiger)_
- Write down some fundamental `decisions`, mostly about key names and key set structure. _(Markus Raab in discussions with Klemens Böswirth)_

## 427.8 Tests

- Fixed the `is_not_rw_storage` function. _(Lukas Kilian)_
- We now ensure that the `check_import` and `check_export` tests run for at least one plugin. _(Lukas Kilian)_

## 427.9 Build

### 427.9.1 CMake

- `make uninstall` also uninstalls symlinks. _(Markus Raab)_
- `external-links.txt` and `extra_install_manifest.txt` are cleaned up at CMake runs. _(↩ Markus Raab)_
- Increased CTest timeout for `testscr_check_kdb_internal_suite` due to timouts reached on slow test machines. _(Mihael Pranjić)_

### 427.9.2 Docker

- Added Alpine Linux docker image with latest Elektra installed. This image is published on docker hub as `elektra/elektra`. We will update the image for each Elektra release such that novices can easily test Elektra without compiling or installing. _(Mihael Pranjić)_
- Remove unused `libgtest-dev` from docker images. _(Mihael Pranjić)_
- Add Ubuntu Focal (20.04) docker image. _(Robert Sowula)_
- Update Alpine Linux docker image to 3.12.1. _(Mihael Pranjić)_

## 427.10 Infrastructure

### 427.10.1 Cirrus

- Update FreeBSD images from version 11.3 to 11.4. _(Mihael Pranjić)_
- Increase CPU count for containers to 4. _(Mihael Pranjić)_
- Use Ruby 2.7 on macOS. _(Mihael Pranjić)_
- Export Python 3.8 path for macOS builds. _(Mihael Pranjić)_

### 427.10.2 Jenkins

- Temporarily resolve cyclic dependency between go-elektra and libelektra builds. _(Mihael Pranjić)_
- Add a new Jenkinsfile for release automation. _(Robert Sowula)_
- Our release pipeline now also builds deb packages for Ubuntu Focal (20.04). _(Robert Sowula)_

### 427.10.3 Travis

- Update macOS builds to use GCC 10, Ruby 2.7.1 and Xcode 12.2. _(Mihael Pranjić)_
- Increase wait time on builds to fix build timeout errors ("No output has been received in the last 10m0s"). _(Mihael Pranjić)_
- Export Python 3.9 path for macOS builds. _(Mihael Pranjić)_

## 427.11 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Renamed the source folder of the website and removed the backend. _(Markus Raab)_
- Use strict dependency injection for website modules. _(Marvin Mall)_
- Added `package-lock.json` to ensure repeatable builds. _(Marvin Mall)_

## 427.12 Decisions

We are intensively working on Elektra 1.0. The last details of the semantics of Elektra 1.0 are in discussion. The decisions are documented, but some of them are not finalized.
If you are interested in the discussions, please `subscribe`.

## 427.13 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- Elektrify GNOME _(Mihael Pranjić)_
- Default TOML plugin _(Jakob Fischer)_
- Improve Plugin Framework _(Vid Leskovar)_
- Keyname Overhaul _(Klemens Böswirth)_
- Continious Releases _(Robert Sowula)_

- FUSE Integration _(Alexander Firbas)_
- 1.0 API _(Philipp Gackstatter)_
- Improve 3-way merge _(Dominic Jäger)_
- Shell completion _(Ulrike Schäfer)_
- Improve Elektra developer experience _(Hani Torabi)_
- Ansible bindings _(Thomas Waser)_

## 427.14 Statistics

We closed 15 issues for this release.
About 17 authors changed 533 files with 26133 insertions(+) and 15834 deletions(-) in 545 commits.
Thanks to all authors for making this release possible!

## 427.15 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 427.16 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- author: mpranj
- name: elektra-0.9.3.tar.gz
- size: 7540609
- md5sum: 649fe13100266ab0e9a4cd19b99e2049
- sha1: 02900ef5c8b24cf067930068c62f2ff09b44354d
- sha256: 5022a6ebf004d892ded03fcf6eb3d223942a7fadd2d68f14d847d1f7f243e1d7

The release tarball is also available signed by Mihael Pranjic using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 9C18145C22F9E746D743DEC59ECC0F4CF0359C7B
Already built API documentation can be found here or on GitHub.

## 427.17 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 428

# 0.9.4 Release

- guid: a3f66c2f-ae6a-412b-bdbd-f8950adb2334
- author: Mihael Pranjić
- pubDate: Mon, 01 Feb 2021 22:54:30 +0100
- shortDesc: Key Name Improvements, Debian and Fedora Packaging

We are proud to release Elektra 0.9.4.

## 428.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`

## 428.2 Highlights

- Important Breaking Changes to key names et al. *see below*
- Debian and Fedora Packaging with CPack *see below*

### 428.2.1 Important Breaking Changes

- The structure of key names has been changed. `see below` _(Klemens Böswirth)_
  **This change breaks mountpoint configurations.** Please follow the upgrade procedure `shown below`.
- The backend fallback procedure introduced in Elektra 0.8.15 has been removed and the structure of the `warnings` metadata array has been changed. `see below` _(Klemens Böswirth)_
- We removed the `ini` plugin (superseded by the TOML plugin), the `null` plugin (superseded by the base64 plugin) and the `tcl` plugin _(Markus Raab, Philipp Gackstatter)_

#### 428.2.1.1 Important Changes to Key Names

There have been significant changes to Elektra's key names:

- The most important change is that you now need a `:` after the namespace. So instead of `system/elektra/version` you have to use `system:/elektra/version`.
- The second big change is to array elements. From now on `keyNew ("/array/#10", KEY_END)` will create a `Key` with name `/array/#_10`, to make arrays more user-friendly by preserving numerical ordering.

- The whole implementation for `keySetName`, `keyAddName`, etc. has been completely rewritten. If you rely on specific behavior of Elektra's key names and have already taken the two changes above into account, please refer to the newly created key name documentation and Python reference implementation.
- Metakeys now use the namespace `meta:/`. The accessor functions `keyGetMeta` and `keySetMeta` automatically add this namespace to preserve compatibility. However, if you use the recently introduced `keyMeta` or otherwise directly access the key name of a metakey, you will have to update your code.
- `default:/` is a new namespace used for keys that exist purely to represent a default value (e.g. generated by the `spec` plugin).

  Looking up cascading keys with `ksLookup` now looks at namespaces in the following order:

  - `proc:/`
  - `dir:/`
  - `user:/`
  - `system:/`
  - `default:/`
  - `/` (cascading key itself)

  The final lookup of the cascading key itself, will be removed in a future release. Please update your code to generate `default:/` keys, if you rely on this feature.

  Note: The `spec` plugin already generates `default:/` keys.
- The function `keyInactive` has been removed. The concept of inactive keys no longer exists, use `comment/#` instead.
- `ElektraNamespace` is the new C++ `enum class` for the Elektra's namespaces. You should prefer it to using `KEY_NS_SYSTEM` et al. directly, if you use C++.
- `keyGetFullName` et al. have been removed. The concept of a "full name (with owner)" no longer exists.

A huge thanks to _(Klemens Böswirth)_ for doing these important changes and clean-ups.

#### 428.2.1.2 Mountpoint upgrade

The change to key names breaks existing mountpoint configurations.
It is not hard to fix the mountpoint configs even after the updating to the new version.
There are two places that will still contain the old syntax after the update:

1. Every key below (and including) `system:/elektra/mountpoints/<MOUNTPOINT>` uses an old key names as <MOUNTPOINT>, if the mountpoint was created with `kdb mount`.
2. The value of all keys matching `system:/elektra/mountpoints/*/mountpoint` must be valid key names.

Fixing the first instance is optional. There the key name is just used to create a unique name for the mountpoint.
The second instance, however, must be fixed or Elektra will be unusable.

> **Disclaimer:** We cannot guarantee that the commands below work for all cases. We also make no guarantees that the command will not break things.
>
> Please report any problems.
>
> *You have been warned. Manually backup important data first.*

For the migration you can use the following commands:

```
#! /usr/bin/env sh
kdb export system:/elektra/mountpoints ni > mountpoints.ini
sed -E 's~((^\[?|/mountpoint = )(user|system))((\\\\)?/)~\1:\4~g' mountpoints.ini > mountpoints_corrected.ini
kdb mv -r system:/elektra/mountpoints system:/elektra/mountpoints-backup
kdb import system:/elektra/mountpoints ni < mountpoints_corrected.ini
```

> *Note:* The original `system:/elektra/mountpoints` data will be moved to `system:/elektra/mountpoints-backup`

### 428.2.2 Debian and Fedora Packaging with CPack

We are now using CPack to generate modular Debian, Ubuntu (DEB) and Fedora (RPM) packages. This simplifies the packaging process and solves problems where a PR, which introduces changes to installed files, fails. We can now also set distribution specific dependencies with CPack, which is needed for some packages. _(Robert Sowula)_
We now provide DEB and RPM packages for releases and for every commit on master in our own repositories using CPack for:

- DEB packages for Debian Buster
- DEB packages for Ubuntu Bionic
- DEB packages for Ubuntu Focal
- RPM packages for Fedora 33

A big thanks to _(Robert Sowula)_ for introducing CPack and creating the repositories.

#### 428.2.2.1 Short Installation Guide

##### 428.2.2.1.1 DEB packages

1. First, you need to obtain the repository key:

```
sudo apt-key adv --keyserver keys.gnupg.net --recv-keys F26BBE02F3C315A19BF1F791A9A25CC1CC83E839
```

1. Add `deb https://debs.libelektra.org/<DISTRIBUTION> <DISTRIBUTION> main` into `/etc/apt/sources.list` where `<DISTRIBUTION>` is the codename of your distributions e.g. `focal`, `bionic` or `buster`.

```
apt-get install libelektra5-all
```

**428.2.2.1.2 RPM packages** Download our `.repo configuration file` and add it to yum/dnf.
To get all packaged plugins, bindings and tools install:

```
dnf install libelektra5-all
```

For more available packages, further instructions on how to add our repositories or instructions on how to use our master built packages, please refer to our install documentation.

#### 428.2.2.2 Version Bump

The 0.9.∗ series of Elektra is for development of Elektra 1.0. Elektra 1.0 will be incompatible to 0.8 and as such, we need a SO version bump. We used this release to bump the SO version from 4 to 5 due to the breaking changes that are not visible in the API.

> *Note:* that within 0.9.∗ we likely introduce further breaking changes but we will not bump the SO version again.

The package names which consist of the SO Version also changed from libelektra4∗ to libelektra5∗. If you used our previous repository with master built packages, please make sure to migrate to our new package repositories described above or in our install documentation.
The version of the Java bindings was also bumped from 4 to 5, although the API is also work in progress.
A big thanks to _(Robert Sowula)_ for doing the necessary renamings.

## 428.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 428.3.1 jni

- Fix rare memleak when the `jni` plugin is closed. _(Mihael Pranjić)_

### 428.3.2 mINI

- We changed the `provides` clause in the plugin contract. Now mINI offers support for the `properties format` (`storage/properties`) instead of the INI file format (`storage/ini`). This makes sense, since the plugin never supported the `section syntax` of INI files. _(René Schwaiger)_

### 428.3.3 Quickdump

- Support for the old quickdump v1 and v2 formats has been removed. _(Klemens Böswirth)_

### 428.3.4 Simple INI

- The plugin contract now correctly states that the plugin offers support for the `properties format`. Before it would state that the plugin offered support for the INI file format. This is not true, since the plugin does not support the `section syntax` of the INI file format.

### 428.3.5 YAML CPP

- We fixed an `use after free bug in the plugin`. _(René Schwaiger)_

### 428.3.6 Yan LR

- The plugin now works (with and) requires `ANTLR 4.9`. _(René Schwaiger)_

## 428.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 428.4.1 Compatibility

- We removed the fallback procedure introduced in Elektra 0.8.15 (using `KDB_DB_FILE` (`default.ecf`) for `system:/elektra`, if the bootstrap backend `KDB_DB_INIT` (`elektra.ecf`) isn't found). If you still rely on this feature, either use `kdb upgrade-bootstrap` **before** upgrading, or manually extract `system:/elektra` into `elektra.ecf`.

- There was an update to how warnings are generated. For users this means that the `warnings` metadata now forms a proper array. Specifically, the first 100 warnings are stored below to the meta keys `warnings/#0`, `warnings/#1`, ..., `warnings/#9`, `warnings/#_10`, ..., `warnings/#_99`. After that, warnings will wrap around, so the 101st warning will be stored as `warnings/#0`, 102nd as `warnings/#1` etc.

### 428.4.2 Core

- `kdbSet` now properly handles, if the given `parentKey` is `NULL` or has read-only name, value or metadata. _(Klemens Böswirth)_

### 428.4.3 Proposal

- Removed `elektraKeyGetMetaKeySet` and moved `keySetStringF` to the hosts plugin. _(Philipp Gackstatter)_

- Removed `ksPopAtCursor`. _(Philipp Gackstatter)_

## 428.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 428.5.1 Lua

- Remove ipairs support and add our own iterator to add support for Lua 5.4, since `__ipairs` was deprecated. _(Manuel Mausz)_

### 428.5.2 JNA

- Fixed allocation not correctly conveyed on key set initialization _(Michael Tucek)_

### 428.5.3 C++

- `ElektraNamespace` is the new C++ `enum class` for the Elektra's namespaces. You should prefer it to using `KEY_NS_SYSTEM` et al. directly, if you use C++. The array `ELEKTRA_NAMESPACES` can be used to iterate over all namespaces. _(Klemens Böswirth)_

### 428.5.4 Ruby

- Enable `__declspec` attributes for Ruby 3.0. _(Mihael Pranjić)_

## 428.6 Tools

- The kdb cmd-line tool outputs better error messages on wrong names like `user` or `user:` as `user:/` is now required. _(Markus Raab)_
- The QtGUI was updated to be compatible with the new key name structure. _(Klemens Böswirth)_

## 428.7 Scripts

- We fixed the (possibly) infinitely running function `generate-random-string` in check-env-dep. _(René Schwaiger)_

## 428.8 Documentation

- Finalize 1.0 decisions. _(Markus Raab)_
- Update API design document _(Markus Raab and Stefan Hanreich)_
- Update release instructions _(Robert Sowula)_
- Changed API documentation terms [current, latest] to [latest, master]. The API documentation of the latest release is now available at `https://doc.libelektra.org/api/latest/html/` and of the current Git master at `https://doc.libelektra.org/api/master/html/`. _(Robert Sowula)_

## 428.9 Tests

- Tests that use additional executables can now be installed and run via `kdb <testname>`. Existing tests have been update to support this. _(Klemens Böswirth)_
- Update source formatting check to clang-format 11. _(Mihael Pranjić)_

## 428.10 Build

### 428.10.1 CMake

- Use Lua 5.4 when available. _(Mihael Pranjić)_
- Force `RTLD_NODELETE` on dlopen() when the `ENABLE_ASAN` CMake option is used. This enables ASAN to find symbols which otherwise might be unloaded. _(Mihael Pranjić)_

### 428.10.2 Docker

- We added a Docker image for building the documentation on Debian Sid. _(René Schwaiger)_
- We removed the Docker image for building the documentation on Debian Stretch. _(René Schwaiger)_
- Add Fedora 33 Dockerfile for Cirrus and Jenkins CI. _(Mihael Pranjić)_
- Debian Sid: update to clang 11. _(Mihael Pranjić)_

## 428.11 Infrastructure

### 428.11.1 Cirrus

- Upgrade Cirrus Fedora docker image to Fedora 33. _(Mihael Pranjić)_
- Upgrade to Ruby 3.0 for macOS builds. _(Mihael Pranjić)_

### 428.11.2 GitHub Actions

- We added a build job that translates Elektra with GCC on macOS. _(Mihael Pranjić, René Schwaiger)_

### 428.11.3 Jenkins

- We refactored shared code between pipelines into a Jenkins Shared Library. _(Robert Sowula)_
- We now use Debian Sid to build the documentation instead of Debian Stretch. The Doxygen version in Debian stretch contains a bug that causes the generation of the PDF documentation to fail. _(René Schwaiger)_
- Use Fedora 33 and 32, drop Fedora 31 use in Jenkins. _(Mihael Pranjić)_
- The Main and Release Pipeline now creates packages for Debian Buster, Ubuntu Bionic, Ubuntu Focal and Fedora-33. These packages are also installed and automatically tested before they are published. To install these packages, please refer to our Install documentation. _(Robert Sowula)_
- We updated our Release Pipeline to push changes directly to our Git repositories. _(Robert Sowula)_

### 428.11.4 Travis

- Move macOS GCC 10 build job to GitHub Actions. _(Mihael Pranjić)_

## 428.12 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date.

## 428.13 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth)_
- Improve Java Development Experience _(Michael Tucek)_
- Elektrify GNOME _(Mihael Pranjić)_
- Continious Releases _(Robert Sowula)_

- KDB access using FUSE _(Alexander Firbas)_
- Default TOML plugin _(Jakob Fischer)_
- Improve Plugin Framework _(Vid Leskovar)_
- Improve 3-way merge _(Dominic Jäger)_
- Shell completion _(Ulrike Schäfer)_
- Ansible bindings _(Thomas Waser)_

## 428.14 Statistics

We closed 21 issues for this release.
About 13 authors changed 1280 files with 26471 insertions(+) and 29959 deletions(-) in 428 commits.
Thanks to all authors for making this release possible!

## 428.15 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 428.16 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.9.4.tar.gz
- size: 7562103
- md5sum: 670a3e9ab225ef53cb2db9058225d7d9
- sha1: 2a5729462fc61694f2b81ca7dbd3620f09cbbf72
- sha256: e1f11f063ab262ce056238ca17aa60442a450a0bb6c5f57a9959df0365576bc6

The release tarball is also available signed using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 428.17 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 429

# 0.9.5 Release

- guid: 8a56a045-3d2e-427d-84bb-8256635159d2
- author: Mihael Pranjić
- pubDate: Mon, 12 Apr 2021 08:43:05 +0200
- shortDesc: Java Binding Improvements, Breaking Change to `kdbOpen`

We are proud to release Elektra 0.9.5.

## 429.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit https://libelektra.org.
You can also read the news on our website
You can try out the latest Elektra release using our docker image elektra/elektra. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run `docker run -it elektra/elektra`.

## 429.2 Highlights

- Breaking change to `kdbOpen`. *see below*
- Ongoing improvements of Java bindings and publishing of bindings to maven central for easy dependency integrations in Java projects

### 429.2.1 <tt>kdbOpen</tt> Contracts

The signature of `kdbOpen` has been changed from

```
KDB * kdbOpen (Key * errorKey);
```

to

```
KDB * kdbOpen(const KeySet * contract, Key *parentKey);
```

You can use `kdbOpen (NULL, errorKey)` to get the same behavior as before.
The new parameter `contract` is similar to what could be done via `kdbEnsure` (which has been removed). Currently, the contract allows you to mount global plugins and add data into the global KeySet (passed to all plugins) during `kdbOpen`. This alone is already quite powerful, but we might more functionality in future releases.
For now, there are three use cases for the `contract` parameter. All of them are covered by helper functions:

```
int elektraGOptsContract (KeySet * contract, int argc, const char * const * argv, const char * const * envp,
       const Key * parentKey, KeySet * goptsConfig);
int elektraIoContract (KeySet * contract, ElektraIoInterface * ioBinding);
int elektraNotificationContract (KeySet * contract);
```

With `elektraGOptsContract` you can mount and set up the `gopts` plugin used for command-line argument parsing. The other two functions are the new way to configure Elektra's notification feature.
For more information take a look at doc/dev/kdb-contracts.md

## 429.3 Plugins

The following section lists news about the plugins we updated in this release.

### 429.3.1 Cache

- The cache plugin now only caches the parts of the global keyset that are below system:/elektra/cache or below system:/elektra/cached. The part below system:/elektra/cache is meant for internal data of the cache, so you should put data below system:/elektra/cached, if you want it to be cached. _(Klemens Böswirth)_

### 429.3.2 internalnotification

- Fix use of kdb_long_double_t on armel platforms ( #3450). _(Mihael Pranjić)_

### 429.3.3 Dbus & Dbusrecv

- Internal changes to ensure compatibility with the new elektraNotificationContract. _(Klemens Böswirth)_

### 429.3.4 YAML Smith & Yan LR

- Removed plugins yamlsmith and yanlr. _(René Schwaiger)_

### 429.3.5 Zeromqsend & Zeromqrecv

- Internal changes to ensure compatibility with the new elektraNotificationContract. _(Klemens Böswirth)_

## 429.4 Libraries

The text below summarizes updates to the C (and C++)-based libraries of Elektra.

### 429.4.1 Compatibility

- keyCopy and keyDup now take an additional flag. See below.
- kdbEnsure was removed and integrated into kdbOpen, which now takes an additional KeySet * contract parameter. See above

### 429.4.2 Core

- The keyCopy and keyDup functions have been changed. They now take a flags argument which specifies which parts of the Key should be copied. The API also changed slightly. Most importantly NULL values are handled differently. For example, keyDup (NULL, KEY_CP_ALL) returns a key similar to what keyNew ("/", KEY_END) produces, whereas previously keyDup (NULL) returned NULL. _(Klemens Böswirth)_
- We added keyReplacePrefix, a function that allows you to easily move a key from one parent to another. _(Klemens Böswirth)_
- kdbEnsure was removed and replaced by similar functionality added to kdbOpen. *see above* _(Klemens Böswirth)_
- KEY_END is now defined as (void *) 0 instead of 0. This allows us to mark keyNew with the GCC attribute __attribute__ ((sentinel)), which causes a compiler warning, if keyNew calls don't use KEY_END as their last argument. _(Klemens Böswirth)_

### 429.4.3 Io

- elektraSetIoBinding has been removed. Use elektraIoContract instead. _(Klemens Böswirth)_

### 429.4.4 Notification

- elektraNotificationOpen has been removed. Use elektraNotificationContract instead. elektraNotificationClose has also been removed. There is no replacement, cleanup now happens automatially during kdbClose. _(Klemens Böswirth)_
- The contract for transport plugins has been changed. The exported functions "openNotification", "closeNotification" and"setIoBinding"are no longer used. Instead, plugins should retrieve the I/O binding from the keysystem:/elektra/io/bindingin the global keyset. The notification callback and context that were passed to"openNotification", can now be read from the global keyset as well. The keys aresystem:/elektra/notification/callbackandsystem:/elektra/notification/context` respectively. _↩ (Klemens Böswirth)_

## 429.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 429.5.1 JNA

- Since internal iterator support for KeySet is due to being dropped, the following methods have been removed:
  - Elektra::ksNext
  - Elektra::ksCurrent
  - Elektra::ksGetCursor
  - Elektra::ksSetCursor
  - KeySet::next
  - KeySet::current
  - KeySet::rewind
  - KeySet::getCursor
  - KeySet::setCursor

  Until internal KeySet iterator support has been dropped form native library, Elektra::ksRewind is being retained while also being annotated as 'deprecated for removal'. The reason is, that we still need to rewind a KeySet before passing it to a native plugin via NativePlugin::set, NativePlugin::get or NativePlugin::error. _(Michael Tucek)_

  Furthermore Elektra::ksPop and KeySet::pop have been removed and KeySet::remove has been introduced as replacment. Until internal KeySet iterator support has been dropped form native library, Elektra::ksRewind is being retained while also being annotated as 'depricated for removal'. The reason is, that we still need to rewind a KeySet before passing it to a native plugin via NativePlugin::set, NativePlugin::get or NativePlugin::error. _(Michael Tucek)_

  Further more Elektra::ksPop and KeySet::pop have been removed and KeySet::remove has been introduced as replacment. _(Michael Tucek)_
- Renamed KeyException specializations: KeyInvalidNameException, KeyTypeConversion↩ Exception, KeyTypeMismatchException
- Migration from Maven to Gradle _(Michael Tucek)_
- Updated documentation for usage of published artifacts _(Michael Tucek)_
- Integration of Maven Central publishing on Elektra release _(Robert Sowula)_

#### 429.5.1.1 Outlook

Ongoing work on bringing the JNA binding up to scratch and improving developer experience. Both for JNA binding API consumers, as well as future JNA binding contrubutors. _(Michael Tucek)_

### 429.5.2 Python & Lua

Add support for keyset.remove(key). _(Manuel Mausz)_

## 429.6 Tools

- `webd`: update `ini`, `y18n` and `elliptic` dependencies. _(Mihael Pranjić)_
- Make search for providers not skip rest of plugins on exceptions. _(Markus Raab)_

## 429.7 Examples

- Fix enums in examples/spec. _(Markus Raab)_

## 429.8 Documentation

- Document names of different components. _(Markus Raab)_
- Update buildserver documentation _(Robert Sowula)_
- Reworked `METADATA.ini` _(Markus Raab)_
- Minor rewording in INSTALL.md _(-kraschitzer)_
- Write notes that `\\` are due to shell recorder, and are not to be copied _(Markus Raab)_
- Add link to `Go` bindings _(Markus Raab)_
- Fix order of tutorials _(Markus Raab)_
- Added API-Reviews for multiple functions in the public API _(Stefan Hanreich)_
- Minor rewording in java-kdb.md _(@aaronabebe)_
- Added a short Visual Studio 2019 tutorial (/doc/tutorials/contributing-windows.md) _(Dominic Jäger)_
- Added hint regarding WSL filesystem configuration (/doc/tutorials/contributing-windows.md) _(@tucek)_
- Fixed broken link in yanlr-plugin readme _(@lawli3t)_
- Minor readability improvement in highlevel.md _(Tobias Schubert @qwepoizt)_
- Fix examples of spec plugin. _(Robert Sowula)_

## 429.9 Tests

- Added small test for jna Return plugin (`Return.java`), `KeyNameIterator.java` _(@aaronabebe)_

## 429.10 Packaging

- Change shlibs version compatibility policy of Debian packages to ">=". _(Robert Sowula)_
- Automate publishing of the release Elektra Docker images. _(Robert Sowula)_

## 429.11 Build

### 429.11.1 CMake

- Fix issue where the library runpaths of the jni plugin could not be resolved. _(Robert Sowula)_

### 429.11.2 Docker

- Update Alpine Linux images to version 3.13.1 and update Elektra release image. _(Mihael Pranjić)_

## 429.12 Infrastructure

### 429.12.1 Cirrus

- Update FreeBSD images from version 12.1 to 12.2 _(Robert Sowula)_
- Update brew before installing packages and print brew config. _(Mihael Pranjić)_
- Restart `dbus` service before running tests and find `DBUS_LAUNCHD_SESSION_BUS_SOCKET` manually (as workaround). _(Mihael Pranjić)_
- Use macOS Big Sur images. _(Mihael Pranjić)_

### 429.12.2 GitHub Actions

- Fix issues with `dbus` and java paths, exclude `jni`. _(Mihael Pranjić)_

### 429.12.3 Jenkins

- Update daily job to always keep the latest Docker images containing installed Elektra packages that were build on master or during release. _(Robert Sowula)_
- Add a cleanup of the aptly database to the daily job. _(Robert Sowula)_

## 429.13 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- It is now possible to have two links on the same line of a Markdown file rendered on the website. _(Klemens Böswirth)_
- The file doc/KEYNAMES.md is now rendered on the website. _(Klemens Böswirth)_
- Update `ini` dependency. _(Dependa Bot)_
- Update many dependencies (Node 14.x LTS, angular, bootstrap, ..) and fix broken RSS feed permalinks. _(Mihael Pranjić)_

## 429.14 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth)_
- Improve Java Development Experience _(Michael Tucek)_
- Elektrify GNOME _(Mihael Pranjić)_

- Continious Releases _(Robert Sowula)_
- KDB access using FUSE _(Alexander Firbas)_
- Default TOML plugin _(Jakob Fischer)_
- Improve Plugin Framework _(Vid Leskovar)_
- Improve 3-way merge _(Dominic Jäger)_
- Shell completion _(Ulrike Schäfer)_
- Ansible bindings _(Thomas Waser)_

## 429.15 Statistics

We closed 20 issues for this release.
About 19 authors changed 515 files with 19081 insertions(+) and 10602 deletions(-) in 375 commits.
Thanks to all authors for making this release possible!

## 429.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 429.17 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.9.5.tar.gz
- size: 7636892
- md5sum: 2245727ed0042645d98de34a1872fbb4
- sha1: c0181bbee212a46b5a9eda3180ff7673f657d6ed
- sha256: 0b6ee9d6bf13c3749f4d014df444606f84a2f5a797a541002f8d4e745007c3a5

The release tarball is also available signed using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 429.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 430

# 0.9.6 Release

- guid: ad3d9308-4019-46dc-9de0-b3b82de5302a
- author: Mihael Pranjić
- pubDate: Mon, 07 Jun 2021 09:48:41 +0200
- shortDesc: Java Fixes, Documentation Updates, GCC 11 and Clang 12 Compatibility

We are proud to release Elektra 0.9.6, the 7th release in preparation for Elektra 1.0.

## 430.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 430.2 Highlights

- Java fixes: JNI plugin fixed and JNA bindings improved
- Fedora 34 and Debian Bullseye packages added
- Documentation Improvements & Cleanups
- GCC 11 and Clang 12 compatibility

### 430.2.1 JNI plugin fixed

The JNI plugin was encountering a double free on open. This has been fixed in conjunction with an update to JNA binding release mechanism. The previously disabled JNI test have been fixed and enabled.
For how to write plugins, please refer to java-plugins.md as well as the JNI plugin and JNA binding documentation.
**Note** that it is currently not possible to mount plugins written in Java (see `#3881`).

## 430.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 430.3.1 JNI

- Fixed double free issues and re-enabled tests
- Updated documentation
- Increased minimum required JDK version to 11

_(Michael Tucek)_
Special thanks to _(Klemens Böswirth)_, _(Mihael Pranjić)_ and _(Robert Sowula)_ for helping with the problem analysis!

### 430.3.2 Dbus, Dbusrecv and Zeromqsend

- Internal changes to ensure compatibility with the new `elektraNotificationContract`. _(Klemens Böswirth)_

### 430.3.3 Xerces

- Store length of an array in metakey array according to array decision. _(Robert Sowula)_

### 430.3.4 YAML Smith and Yan LR

- Removed plugins. _(René Schwaiger)_

### 430.3.5 ni

- Silence Clang 12 warnings about suspicious string literal concatenation. _(Mihael Pranjić)_

### 430.3.6 Lua

- Removed outdated information from docs _(-kraschitzer)_

## 430.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 430.4.1 Core

- Remove keyCompareBy(Name)?Owner _(-kraschitzer)_

## 430.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 430.5.1 SWIG

- Remove `-Wno-shift-overflow warnings` option from SWIG bindings compile flags. _(Mihael Pranjić)_
- Suppress SWIG/Ruby bindings warning about `operator != ignored`. _(Mihael Pranjić)_

### 430.5.2 JNA

- Increased minimum required JDK version to 11
- Gradle wrapper and docker images upgraded to 7.0.2
- Minumum Gradle version decreased to 6.0
- Upgraded JNA dependency from 4.5.2 to 5.8.0
- Updated Java binding API documentation
- Migrated native resource clean-up from `finalize()` to `Cleaner`
  - Please revisit the documentation for `Key::release` and `KeySet::release` for recommended resource release handling
  - Currently automated native key and key set resource clean-up is deactivated due to some possible race conditions which might result in double free errors. Manual key release is also disabled to not break current API release semantic. This leads to memory leaks, which is a known issue and will be resolved in an upcoming release.
- Introduced multiple exceptions when native API calls fail - see updated Java doc for details
- Introduced early parameter validation for values which would otherwise lead to unspecific errors in native API calls
- Update `Key` API introducing the following changes:
  - Extracted exceptions from `Key` class
  - Fixed `Key::getCurrentMeta`
  - Moved `Elektra.KeyNewArgumentFlags` to `Key.NewArgumentTag`
  - Changed return types from `int` to `boolean` or enabled a fluent interface where appropriate
  - Renamed `Key::*Integer` to `Key::*Int`
  - Renamed `KeyInvalidNameException` to `KeyNameException`
  - Renamed `KeyTypeMismatchException` to `KeyBinaryTypeNotSupportedException`
  - Introduced `Key::get*AndRelease` convenience methods
  - Introduced `Key::createNameless`
  - Introduced `KeyReleasedException` being thrown when a released `Key` is being accessed
  - Introduced `KeyMetaException`
  - Removed unused `KeyTypeConversionException`
  - Removed `Key::needsSync`
  - Removed `Key::isNull`
    - `KeyReleasedException` is now being thrown when a released (= previously `isNull`) `Key` is being accessed
    - `Key`s with no backing native key pointer cannot be created anymore
- Updated `KeySet` API introducing the following changes:
  - Changed return type enabling a fluent interface where appropriate
  - Renamed `KeySet::head` to `KeySet::first`
  - Renamed `KeySet::tail` to `KeySet::last`
  - Introduced `KeySetReleasedException` being thrown when a released `KeySet` is being accessed
  - Introduced `KeySetAppendException`
  - Removed `KeySet::create(int, Object[])`
  - Removed `KeySet::needsSync`

  - Methods which have been returning a nullable `Key`, now return an `Optional<Key>´ −KeySet↩ ::lookup∗now returns`Optional<Key>−Key::getMeta<tt>now returnsOptional<Key>`
    * Example: ```java // checking whether the key has been found BEFORE API change Key found = ks.lookup("/some/key"); if (found != null) { // process found key } java // checking whether the key has been found AFTER API change ks.lookup("/some/key").ifPresent(k -> // process found key ); ``

- `Updated`KDB`API introducing the following changes:`
  - `IntroducedKDBClosedExceptionbeing thrown when a closedKDBsession is being accessed`
  - `IntroducedKDB::get(Key)`
  - `IntroducedKDB::open()`
  - `IntroducedKDB::open(KeySet)`
  - `IntroducedKDB::goptsContract(String[], String[], Key, KeySet)`
  - `Changed return type enabling a fluent interface where appropriate`
- `Updated tests accordingly`

`_(Michael Tucek)_`

## 430.6 Tools

- `Remove kdb set functionality that creates a null key. _(Robert Sowula)↩ _`
- `Rename elektraStrnDup to elektraMemDup _(-kraschitzer)_`
- `Update specmount error message _(-kraschitzer)_`
- `Update elektraMemDup to void * and update the documentation. _(Mihael Pranjić)_`
- `There have been a few bugfixes for elektrad. _(Klemens Böswirth)_`
- `Update lodash and hosted-git-info dependencies of webd due to security update. _(Mihael Pranjić)_`

## 430.7 Scripts

- `Require clang-format 12 for reformatting C and Java. _(Mihael Pranjić)↩ _`
- `Use basename of release file in generate-hashsums. _(Mihael Pranjić)_`
- `Use shfmt v3.2.4 to reformat shell scripts. _(Mihael Pranjić)_`
- `Use cmake-format v0.6.13 (cmakelang) to reformat CMake. _(Mihael Pranjić)↩ _`
- `Aptly repositories are now automatically created if they do not exist during a package release. _(Robert Sowula)_`

## 430.8 Documentation

- Added reviews for all functions contained in the Elektra Core API. _↩ (@lawli3t)_
- Added packaging section to news template. _(Mihael Pranjić)_
- Minor readability improvement in highlevel.md. _(Tobias Schubert @qwepoizt)↩ _
- Fix examples of spec plugin. _(Robert Sowula)_
- Added reviews for all functions contained in the Elektra Core API. _↩ (@lawli3t)_
- Document package names of plugins, bindings and tools. _(Robert Sowula)↩ _
- Small update in API documentation related to different namespaces in returned keys. _(Markus Raab)_
- Improved documentation of noresolver. _(Markus Raab)_
- Improved plugin tutorial. _(Markus Raab)_
- Adding info about syncing forks to doc/Git.md. _(Klemens Böswirth)_
- Work on COMPILE.md and INSTALL.md. _(-kraschitzer)_
- Update and correct licensing information. _(-kraschitzer)_
- Rename RELEASE.md. _(-kraschitzer)_
- Improved documentation for module kdb in Elektra Core. _(@lawli3t)_
- Improved documentation for module key in Elektra Core. _(@lawli3t)_
- Improved documentation for module keyname in Elektra Core. _(@lawli3t)↩ _
- Improved documentation for module keyvalue in Elektra Core. _(@lawli3t)↩ _
- Improved documentation for module keymeta in Elektra Core. _(@lawli3t)↩ _
- Improved documentation for module keytest in Elektra Core. _(@lawli3t)↩ _
- Improved documentation for module keyset in Elektra Core. _(@lawli3t)↩ _
- Fixed example in the "mount-configuration-files" tutorial #3722. _(↩ Philipp Oppel)_
- Update and correct third party licensing information. _(-kraschitzer)↩ _
- Use Ronn-NG instead of unmaintained ronn to generate man pages. _↩ (Mihael Pranjić)_
- Re-generate man pages to add missing information and remove unnecessary symbols and escaping. _(Mihael Pranjić)_
- Update doc/Doxyfile to Doxygen 1.9.1 and fix a syntax error with the FILTER_PATTERNS directive. _(Mihael Pranjić)_

## 430.9 Tests

- Fix failing testshell_markdown_tutorial_crypto on Mac OS and other OS with GnuPG version >= 2.3.1. _(Peter Nirschl @petermax2)_
- Use clang-format 12 for Restyled and update Restyled version. _(Mihael Pranjić)_
- Update all Restyled formatters to current versions. _(Mihael Pranjić)_

## 430.10 Packaging

- We now package the Ruby bindings, ruby plugin and the gitresolver plugin. _(Robert Sowula)_
- We added Fedora 34 packages. _(Mihael Pranjić)_
- We added Debian Bullseye packages. _(Robert Sowula)_

## 430.11 Build

### 430.11.1 CMake

- Disable binding tests when BUILD_TESTING is disabled. _(Robert Sowula)_
- Remove unused FindCppCMS.cmake CMake module and unused Boost variables. _(Mihael Pranjić)_

### 430.11.2 Docker

- Add Fedora 34 images. _(Mihael Pranjić)_
- We added release images that come with pre-installed dependencies and sudo permissions for each distribution we build packages for. _(Robert Sowula)_
- Use Clang 12 and Gradle 7.0 in Debian Sid image. _(Mihael Pranjić)_
- Remove Boost and some unused dependencies from all Docker images. _(Mihael Pranjić)_
- Use Gradle 7.0 and Ronn-NG 0.10.1.pre1 in Docker images. _(Mihael Pranjić)_
- Remove unused Debian Buster doc image. _(Mihael Pranjić)_

## 430.12 Infrastructure

### 430.12.1 Cirrus

- Use Clang 12 and Python 3.9 for macOS builds. _(Mihael Pranjić)_
- Pin GnuPG version to 2.2.x. _(Mihael Pranjić)_
- Update Fedora image to version 34. _(Mihael Pranjić)_
- Clean up unused dependencies in Arch Linux image and add Ronn-NG to generate man pages. _(Mihael Pranjić)_

### 430.12.2 GitHub Actions

- Pin GnuPG version to 2.2.x. _(Mihael Pranjić)_
- Enable jni plugin and fix JAVA_HOME detection. _(Mihael Pranjić)_

### 430.12.3 Jenkins

- We now build and test on Fedora 34 and 33. Fedora 32 was removed from the CI. _(Mihael Pranjić)_
- Build release documentation on Debian Sid, due to newer TeX and Doxygen packages. _(Mihael Pranjić)_

### 430.12.4 Travis

- Update Ubuntu to Focal, use GCC 10 and clean up travis scripts. _(Mihael Pranjić)_

## 430.13 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Update highlight.js due to a ReDOS vulnerability and upgrade other dependencies as well. _(Mihael Pranjić)_
- Catch errors when code highlighting fails. _(Mihael Pranjić)_
- Get rid of unused code: authentication, backend, users, snippets and conversion service. _(Mihael Pranjić)_
- Fix docsearch sourcemap error. _(Mihael Pranjić)_
- Update lodash dependency due to security update. _(Mihael Pranjić)_

## 430.14 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth)_
- Improve Java Development Experience _(Michael Tucek)_
- Elektrify GNOME _(Mihael Pranjić)_
- Continious Releases _(Robert Sowula)_
- KDB access using FUSE _(Alexander Firbas)_
- Default TOML plugin _(Jakob Fischer)_
- Improve Plugin Framework _(Klemens Böswirth)_
- Improve 3-way merge _(Dominic Jäger)_
- Shell completion _(Ulrike Schäfer)_
- Ansible module _(Thomas Waser)_

## 430.15 Statistics

We closed 40 issues for this release.
About 17 authors changed 627 files with 15988 insertions(+) and 16768 deletions(-) in 465 commits.
Thanks to all authors for making this release possible!

## 430.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 430.17 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.9.6.tar.gz
- size: 7650067
- md5sum: ed33e7b61f2b1ed3742f3bc6dd046d53
- sha1: fd6082ee38e31e54b66a96a50fc4d20c9c107c89
- sha256: c8e75f4d21bf3bd6b1028e776af9ff644a17a7dfbb1f2052f50392767deea197

The release tarball is also available signed using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66↩ AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 430.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org
Best regards, Elektra Initiative

# Chapter 431

# 0.9.7 Release

- guid: 39F907DA-8B5B-4984-9D19-33BAB7B71B3D
- author: Mihael Pranjić
- pubDate: Fri, 09 Jul 2021 10:19:29 +0200
- shortDesc: FUSE Tool, TOML Improvements, ElektraSettings GSettings Bindings

We are proud to release Elektra 0.9.7.

## 431.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 431.2 Highlights

### 431.2.1 FUSE Tool

We added an **experimental** preview of the Filesystem in User Space FUSE tool. This tool enables the inspection and modification of the KDB, in the form of a classical filesystem. _(Alexander Firbas)_

### 431.2.2 ElektraSettings GSettings Backend

The **experimental** ElektraSettings GSettings backend has been updated and is working well with GNOME 40. We do not recommend to use it on production systems yet, but we have been testing ElektraSettings as a replacement for `dconf` successfully and want to share the progess with you. If you want to try ElektraSettings back up all your data first. _(Mihael Pranjić)_

### 431.2.3 TOML Improvements

Multiple critical bugs have been fixed in the **experimental** `TOML` plugin *see below*. We are currently working towards using the `TOML` plugin as the default `storage plugin` for Elektra. _(Klemens Böswirth and Jakob Fischer)_

## 431.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 431.3.1 @ref src_plugins_email_README_md "email"

- Introduce email address validation plugin based on regex. _(a-kraschitzer)_

### 431.3.2 Resolver

- Fix invalid cache key name. _(Mihael Pranjić)_

### 431.3.3 Length

- Implement a plugin that validates that a string length is less or equal to given number. _(Philipp Oppel)_

### 431.3.4 Blacklist

- Implement a blacklist plugin that rejects values specified in a metadata array. _(Robert Sowula)_

### 431.3.5 TOML

- Fixed a bug ( #3896) that caused the `toml` plugin to swallow the first letter of all keys (after the namespace), if the parent key was a root key (e.g. `user:/`). _(Klemens Böswirth)_
- The `type` metakey is now set for numbers on reading. _(Jakob Fischer)_
- Rewrote some error messages, to make them less technical. _(Jakob Fischer)_
- Fixed parsing of floats/empty keynames/multiline strings. _(Jakob Fischer)_

### 431.3.6 Python

- Fix format string overflow and add error checking when appending to `sys.path`. _(Mihael Pranjić)_

## 431.4 Libraries

The text below summarizes updates to the C (and C++)-based libraries of Elektra.

### 431.4.1 Core

- A few rare bugs (mostly related to empty keyname parts `/%/`) in the keyname validation and canonicalization logic have been fixed. _(Klemens Böswirth)_
- Fix default backend key name for cache compatibility. _(Mihael Pranjić)_

## 431.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 431.5.1 JNA

- Currently the binding's automated clean-up of native keys and key sets is deactivated until JNI segmentation fault issues are resolved in an upcoming release. After these issues have been resolved, manual clean-up of native resources will be discouraged. Therefore the `Key::get*AndRelease` convenience methods introduced with the last release have been removed. _(Michael Tucek)_
- Upgrade Gradle to 7.1.1. _(Mihael Pranjić)_

### 431.5.2 Gsettings

- Fix user and default (system) namespaces and key names. _(Mihael Pranjić)_
- Rewrite `dbus` change notification mechanism. _(Mihael Pranjić)_

## 431.6 Scripts

- Add a script that automates the process of inserting source archive hashsums and Git statistics into the release notes during a release. _(Robert Sowula)_

## 431.7 Documentation

- JNI documentation updates and small fixes. _(Markus Raab)_
- Small updates in notification tutorial. _(Markus Raab)_
- Add tutorial about writing specifications. _(Aaron Abebe `aaron.abebe@gmail.com`)_
- Change GPG keyserver for receiving the apt key from keys.gnupg.net to keyserver.ubuntu.com _(Robert Sowula)_
- The man pages now use the date of the last change recorded in git. _(Klemens Böswirth)_

## 431.8 Tests

- Upgrade GoogleTest frameworks to version 1.11.0. _(Mihael Pranjić)_
- Add additional test cases for module `key`. _(@lawli3t)_
- Add additional test cases for module `keyname`. _(@lawli3t)_
- Add additional test cases for module `keyvalue`. _(@lawli3t)_
- Add tests for module `keyset`. _(@lawli3t)_

## 431.9 Packaging

- Add packages for following bindings: `glib`, `io_ev`, `io_glib` and `io_uv`. _(Robert Sowula)_

## 431.10 Build

### 431.10.1 Docker

- Upgrade Alpine Linux images to 3.14.0. _(Mihael Pranjić)_

## 431.11 Infrastructure

### 431.11.1 Jenkins

- Add the deployment of the `website` to the release pipeline, therefore removing the need to wait until the main pipeline succeeds after a release. _(Robert Sowula)_
- Restructure the release job stages to make it more failsafe and enable a re-run without any version conflict until the last stage. _(Robert Sowula)_
- Fix invalid package artifact path in release pipeline. _(Robert Sowula)_
- Clean Jenkins workspaces after builds. _(Mihael Pranjić)_

## 431.12 Outlook

We are currently working on following topics:

- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth)_
- Improve Java Development Experience _(Michael Tucek)_
- Elektrify GNOME _(Mihael Pranjić)_
- Continious Releases _(Robert Sowula)_
- KDB access using FUSE _(Alexander Firbas)_
- Default TOML plugin _(Jakob Fischer)_ and _(Klemens Böswirth)_
- Improve Plugin Framework _(Klemens Böswirth)_
- Improve 3-way merge _(Dominic Jäger)_
- Shell completion _(Ulrike Schäfer)_
- Ansible module _(Thomas Waser)_

## 431.13 Statistics

We closed 40 issues for this release.
About 18 authors changed 307 files with 6547 insertions(+) and 1914 deletions(-) in 304 commits.
Thanks to all authors for making this release possible!

## 431.14 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 431.15 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.9.7.tar.gz
- size: 7712448
- md5sum: 4355e7df0dcf4178974097604f996747
- sha1: c418d344d72879dd2b3fd6fa8e9831c921cfaba5
- sha256: 12b7b046004db29317b7b937dc794abf719c400ba3115af8d41849127b562681

The release tarball is also available signed using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 431.16 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 432

# 0.9.8 Release

- guid: 0CB8C139-730C-4CCD-9FB4-0C7C4AA4DBF2
- author: Mihael Pranjić
- pubDate: Mon, 04 Oct 2021 00:02:45 +0200
- shortDesc: Redshift Elektrified, HL API & Java Binding Improvements

We are proud to release Elektra 0.9.8.

## 432.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 432.2 Highlights

- `kdb` now prohibits write operations on cascading keys that miss a corresponding existing key. See the details in the `Tools` section below and the new subsection on cascading writes in the tutorial on cascading keys for further information. _(Alexander Firbas)_
- Redshift is now elektrified. _(Tobias Schubert @qwepoizt)_

### 432.2.1 Redshift and Elektra

We have created a version of `Redshift` that uses Elektra for configuration management!
We removed and refactored Redshift's code for loading configuration files, parsing CLI options and validating configuration to use Elektra. Redshift with Elektra has about 700 fewer lines of code (-16%) and is a great example of what Elektra is all about: More applications with less code for configuration management!
To test it, take a look at our PR `Refactor to use Elektra` and follow the instructions provided in `CONTRIBUTING.md`.

#### 432.2.1.1 Benefits of Redshift using Elektra

Refactoring Redshift to use Elektra brings the following benefits:

- Fewer lines of code: reduction of ~700 LOC or 16% (measured across all `*.c` and `*.h` files exclusive of files automatically generated by Elektra).
- Adding new configuration settings is easier and takes less time: Validation and parsing of setting values (from configuration file and CLI options) is handled by Elektra - no custom code required!

- Clean separation of application code and the specification of supported configuration settings (including defaults and validation rules).
- Automatic generation of CLI help text.

### 432.2.2 HL API improvements

Redshift with Elektra uses the high-level API.
We have made a large number of improvements to the high-level API in the course of refactoring Redshift.
The highlights are:

- Improved detection of differences in specification between an application's compilation and runtime.
- Improved validation of CLI options.
- Early detection of errors in specification files.
- Updated and improved documentation and tutorials.
- Various other bugfixes.

More improvements and details are explained in later sections of these release notes.
Thanks to *Klemens Böswirth*, *Markus Raab* and *Tobias Schubert*!

### 432.2.3 Windows releases

We are now shipping experimental releases for Windows 32- and 64-bit! They can be downloaded here.
A big success is that Redshift already works with Elektra under Windows.

## 432.3 Plugins

The following section lists news about the plugins we updated in this release.

### 432.3.1 gopts

- The gopts plugin now includes deeply nested options and arguments in the generated help message. _↩ (Tobias Schubert @qwepoizt)_
- Errors from gopts are now correctly reported. _(Klemens Böswirth)_
- Fix wrong variable names in gopts_win32.h. _(Tobias Schubert @qwepoizt)_

### 432.3.2 range

- The range plugin now uses metakey type as fallback, if check/type is not specified. _(Tobias Schubert @qwepoizt)_
- The range now treats all validation problems as warnings during kdbGet(). _(Tobias Schubert @qwepoizt)↩ _

### 432.3.3 spec

- The spec plugin now runs before other postgetstorage plugins, so that validation can happen during kdbGet as well. This is especially relevant in combination with gopts. _(Klemens Böswirth)_
- Make spec plugin (with no support for # and _ in key names) work in mingw-w64 builds. _(Tobias Schubert @qwepoizt)_

### 432.3.4 sync

- Add support for mingw-w64 builds using fflush. _(Tobias Schubert @qwepoizt)_

### 432.3.5 wresolver

- Add missing `ELEKTRA_PLUGIN_COMMIT` export. _(Tobias Schubert @qwepoizt)_

### 432.3.6 TOML

- Improvements to the parser, comment handling and especially quoting of strings. _(Klemens Böswirth)_
- The `toml` plugin now supports all four kinds of strings via the `tomltype` metadata. The plugin also remembers which kind was used and handles escape sequences properly, instead of always converting to basic strings. For details take a look at the updated README _(Klemens Böswirth)_
- The `comment/#/space` metakey is now used correctly to store the actual whitespace characters from the file, instead of a number. _(Klemens Böswirth)_

## 432.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 432.4.1 Compatibility

- Introduced public C API function `ksSearch`
- Previously public function `ksSearchInternal` is now static. Use `ksSearch` instead.

_(Michael Tucek)_

### 432.4.2 Core

- Remove obsolete `ksNeedSync` function. _(Mihael Pranjić)_
- Replace various occurences of `sprintf` by `snprintf` and fix out of bounds array access in markdown-linkconverter. _(Mihael Pranjić)_

### 432.4.3 High-level API

- Modified High-level API to treat all warnings as errors. _(Tobias Schubert @qwepoizt)_
- Implemented support for warnings in High-level API error handling. _(Tobias Schubert @qwepoizt)_
- Fix a small bug for warnings in High-level API. _(Tobias Schubert @qwepoizt)_
- Fix resource management in High-level API error handling. _(Tobias Schubert @qwepoizt)_
- Implement a check to detect whether an application's specification was properly `mount`ed and `spec-mount`ed. _(Tobias Schubert @qwepoizt)_
- Implement a check to detect whether an application's specification was changed after installation. _(Tobias Schubert @qwepoizt)_
- Add sanity-checks to resource management. _(Tobias Schubert @qwepoizt)_
- Refactor and modularize code. _(Tobias Schubert @qwepoizt)_
- Update and improve inline documentation. _(Tobias Schubert @qwepoizt)_
- Remove "minimal validation" in favor of the new checks (see above). _(Tobias Schubert @qwepoizt)_

### 432.4.4 Ease

- Implement calculation of a specification token (=sha-256 hash). _(Tobias Schubert @qwepoizt)_
- Add `asmonier's sha-2` for sha-256 hash calculation. _(Tobias Schubert @qwepoizt)_

## 432.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 432.5.1 Java binding

- Upgraded Java binding gradle wrapper to 7.2.
- Java source files are formatted using the Google Java format
- Renamed zero argument static factory method `Key::createNameless` to `Key::create`. To migrate to this change, just update calling code to use the new method name.
- Updated method documentation previously publishing the error key based error handling approach to the Java binding consumer. Such arguments are now explicitly only used for returning warning information in case no error occurred. In case of an exceptional state, appropriate exceptions are thrown. Such exceptions provide access to the underlying key containing warning and error information as metadata. Please review API usage to consider the more elaborated explanation of how Elektra uses this argument's value. Affected signatures:
  - Updated javadoc for `KDB::open(Key)`
  - Updated javadoc for `KDB::open(KeySet, Key)`
  - Updated javadoc for `KDB::close(Key)`
  - Updated javadoc for `KDB::get(Key)`
  - Updated javadoc for `KDB::get(KeySet, Key)`
  - Updated javadoc for `KDB::set(KeySet, Key)`, better explaining the relevance of the second argument `parentKey`
- Introduced `KeySet::remove(Key)` and `KeySet::remove(String)`
- Removed `KeySet::lookup(Key, int)` and `KeySet::lookup(String, int)` as well as accompanying flag definitions `KeySet::KDB_O_NONE`, `KeySet::KDB_O_DEL` and `KeySet::KDB_O_POP`. Please use `KeySet::lookup(Key)` and `KeySet::lookup(String)` instead. Instead of `KeySet::KDB_O_DEL`, please consider using `Key::release`. The proper replacement for `KeySet::KDB_O_POP` is `KeySet::remove(Key)` or `KeySet::remove(String)`.
- Native library proxy interface `Elektra` is now package private (previously was public).
- Added example Java plugin `whitelist` (see here)
- Changed `Key nextMeta()` to `Optional<Key> nextMeta ()` no longer throwing NoSuchElementException for non-exceptional behavior
- Native library proxy interface `Elektra` is now package private (previously was public)
- Added example Java plugin `whitelist`
- Added support of binary valued keys:
  - Introduced `Key::getBinary()` and `Key::setBinary(byte[])`
  - Renamed `KeyBinaryTypeNotSupportedException` to `KeyStringValueException`
  - Introduced `KeyBinaryValueException`
  - Improved `Key` test coverage
- Fixed example project in `examples/external/java/read-keys-example`
  - now works with a standard installation of Elektra
  - updated code to work with current Java binding
- `KeySetReleasedException` and `KeyReleasedException` have been replaced by the native `IllegalStateException`

- Introduced abstraction `ReadableKey` to better reflect the limitations of metadata keys via a type hierarchy. Metadata keys are now returned as `ReadableKey`s:
  - `Key` extends `ReadableKey`
  - `Key` class is now final
  - Changed `Key Key::nextMeta()` to `Optional<ReadableKey> Key::nextMeta()`, no longer throwing NoSuchElementException for non-exceptional behavior
  - Changed `Key Key::currentMeta()` to `ReadableKey Key::currentMeta()`
  - Changed `Optional<Key> Key::getMeta(String)` to `Optional<ReadableKey> Key::getMeta(String)`
  - Metadata keys can no longer be manually released
  - Removed `Key::incRef`, `Key::decRef` and `Key::getRef`
  - `ReadableKey/Key` now implements `Comparable<ReadableKey>`
    - `int Key::cmp(Key)` has been renamed to `int Key::compareTo(Key)`
    - `ReadableKey` now implements `equals` and `hashCode` in line with the contract for `int Key::compareTo(Key)`
  - `ReadableKey/Key` no longer implements `Iterable<String>` for iterating over the parts of a key's name - use `Iterator<String> ReadableKey::keyNameIterator()` instead
  - `Key` now implements `Iterable<Key>` to iterate over a key's metadata `ReadableKey`s
  - Fixed API method typo: Renamed `ReadableKey::isDirectBelow/Key::isDirectBelow` to `isDirectlyBelow`
- `KeyNameIterator` and `KeySetIterator` are now package private
- `KeySetAppendException` has been renamed to `KeySetException` and now conveys general `KeySet` related exceptional states
- `KeySet` now implements `SortedSet<Key>` (see `Java API`). Previously `KeySet` was only implementing `Iterator<Key>`. Now a native key set can be used via its `KeySet` representation wherever one of the following Java Collection Framework interfaces is supported:
  - `Iterable`
  - `Collection`
  - `Set`
  - `SortedSet`

_(Michael Tucek)_

### 432.5.2 GLib

- Compile `glib` binding with `-Wno-pedantic` for compatibility. _(Mihael Pranjić)_

## 432.6 Tools

- Really add all tools when using `-DTOOLS=ALL`. _(Markus Raab)_
- ZeroMQ Hub: fix compilation and man page. _(Markus Raab)_
- Configure packaging for FUSE tool. _(Alexander Firbas)_
- FUSE: fix bug preventing binary writes. _(Alexander Firbas)_
- Ambiguous write operations are now disabled in kdb. _(Alexander Firbas)_
- `webd`: update npm dependencies. _(Mihael Pranjić)_

### 432.6.1 KDB

- `kdb set`, `kdb meta-set`: Only allow writes to the cascading namespace if the lookup succeeds. Otherwise, the operation is ambiguous and therefore aborted. No more guessing of namespaces in case a cascading key is given (`user:, system:` for `kdb set`, `spec:` for `kdb meta-set`), _(Alexander Firbas)_
- `kdb set`, `kdb meta-set`: Validation of keys can no longer be bypassed by using non-cascading keys (except with the new –force (-f) option). _(Alexander Firbas)_
- Disable `-N/--namespace` option in all kdb subcommands _(Alexander Firbas)_
- Implement new name part getter commands `kdb namespace`, `kdb basename` and `kdb dirname`. _(Alexander Firbas)_
- `kdb file`: Remove namespace guessing (in case a cascading key is given, it needs to resolve to an existing key). _(Alexander Firbas)_
- `kdb editor/import`: Disable the use of cascading names (and the 'validate' strategy operating on cascading keys) entirely. _(Alexander Firbas)_
- Update numerous tests to comply with changes above. _(Alexander Firbas)_
- Add a new subsection on cascading writes to the tutorial on cascading keys. _(Alexander Firbas)_
- `kdb gen`: Generate specification token during code-generation and add it to generated contract. _(Tobias Schubert @qwepoizt)_
- `kdb gen`: Improve naming of variables to make code easier to understand. _(Tobias Schubert @qwepoizt)_
- `kdb spec-mount`: Improve usability by failing with helpful error messages, if the specification contains errors. _(Tobias Schubert @qwepoizt)_

## 432.7 Scripts

- Add script for mingw-w64 i686 build. _(Tobias Schubert @qwepoizt)_

## 432.8 Documentation

- Add link and small improvements to tutorial about writing specifications and logger. _(Markus Raab)_
- doc: add pre/postconditions and invariants to module key _(@lawli3t)_
- doc: add pre/postconditions and invariants to module keymeta _(@lawli3t)_
- Fix broken links _(@lawli3t)_
- Remove previous authors. _(Markus Raab)_
- add pre/postconditions and invariants to module keytest _(@lawli3t)_
- Updated the news template. _(Mihael Pranjić)_
- Update and improve tutorial and in-code comments for high-level API _(Tobias Schubert @qwepoizt)_
- Improve documentation of opts library _(Tobias Schubert @qwepoizt)_
- Update tutorial "High-level API (with code-generation)" to reflect change of `loadConfiguration()`'s signature in release 0.9.5 _(Tobias Schubert @qwepoizt)_
- add pre/postconditions and invariants to module keyvalue _(@lawli3t)_
- Update and improve inline documentation of `kdb gen`. _(Tobias Schubert @qwepoizt)_
- Fix broken links. _(Robert Sowula)_

## 432.9 Tests

- Fix failing `testshell_markdown_tutorial_crypto` on Mac OS and other OS with GnuPG version >= 2.3.1. _(Peter Nirschl @petermax2)_
- Use clang-format 12 for Restyled and update Restyled version. _(Mihael Pranjić)_
- Update all Restyled formatters to current versions. _(Mihael Pranjić)_
- Add additional test cases for module `keytest` _(@lawli3t)_
- Update tests for high-level API to work with new specification token mechanism. _(Tobias Schubert @qwepoizt)_
- Add tests for libease's sha-256. _(Tobias Schubert @qwepoizt)_
- Add tests for sha-256 hash calculation of a KeySet. _(Tobias Schubert @qwepoizt)_
- Add additional test cases for module `keymeta` _(@lawli3t)_

## 432.10 Packaging

- Add packages for openSuse Leap 15.3. _(Robert Sowula)_

## 432.11 Build

### 432.11.1 CMake

- Add files generated by CMake to .gitignore. _(Tobias Schubert @qwepoizt)_
- Add support for i686 to mingw-w64 toolchains. _(Tobias Schubert @qwepoizt)_
- Add plugins type, cache, spec, gopts, sync to mingw-w64 builds. _(Tobias Schubert @qwepoizt)_

### 432.11.2 Docker

- Add docker images for ABI tests. _(Robert Sowula)_
- Enable BuildKit features to leverage tmpfs to speed up `docker build` commands. _(Mihael Pranjić)_
- Bump Gradle to version 7.2. _(Mihael Pranjić)_
- Bump Debian Buster images to Bullseye and Stretch images to Buster. We still leave one Debian Stretch job due to upstream Debian LTS support until June 2022. _(Mihael Pranjić)_
- Add Dockerfiles for openSUSE Leap 15.3 and CentOS Stream 8. _(Robert Sowula)_
- Add docker image for OpenWrt package building. _(Robert Sowula)_
- Add files generated by docker when tutorial `run-all-tests-with-docker` is followed to .gitignore. _(Tobias Schubert @qwepoizt)_

### 432.11.3 Restyled

- Upgrade to latest stable restylers. _(Mihael Pranjić)_
- Added Google Java formatter _(Michael Tucek)_

## 432.12 Infrastructure

### 432.12.1 Jenkins

- Add ABI test stage for release pipeline. _(Robert Sowula)_
- Move check stages that don't build the code to a dedicated stage, to avoid confusion when parallel builds are aborted. _(Robert Sowula)_
- Add test stages for openSUSE and CentOS. _(Robert Sowula)_
- Use `tmpfs` in Docker to speed up the test suite. _(Mihael Pranjić)_
- Add OpenWrt package building stage to release pipeline. _(Robert Sowula)_
- Add debian-bullseye-mingw-w64-i686 build to Jenkinsfile. _(Tobias Schubert @qwepoizt)_

### 432.12.2 Cirrus

- Bump FreeBSD images to 12.2 and 13.0 using the LLVM 12 toolchain, drop FreeBSD 11. _(Mihael Pranjić)_
- Fix cirrus-file parsing errors. _(Mihael Pranjić)_
- Redistribute CPU and memory resources and enable greedy instances. _(Mihael Pranjić)_

### 432.12.3 GitHub Actions

- Migrate most macOS build jobs to GitHub actions to speed up builds. _(Mihael Pranjić)_
- Upgrade macOS GCC build job to GCC 11. _(Mihael Pranjić)_

## 432.13 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Update npm dependencies. _(Mihael Pranjić)_

## 432.14 Outlook

We are currently working on following topics:

- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth)_
- KDB access using FUSE _(Alexander Firbas)_
- Improve Java Development Experience _(Michael Tucek)_
- Improve Plugin Framework _(Klemens Böswirth)_
- Default TOML plugin _(Klemens Böswirth)_, _(Markus Raab)_ and _(Jakob Fischer)_
- Elektrify KDE _(Dardan Haxhimustafa)_, _(Felix Resch)_ and _(Mihael Pranjić)_
- Elektrify GNOME _(Mihael Pranjić)_
- Continious Releases _(Robert Sowula)_
- Improve 3-way merge _(Dominic Jäger)_
- Shell completion _(Ulrike Schäfer)_
- Ansible module _(Thomas Waser)_

## 432.15 Statistics

We closed 59 issues for this release.
About 17 authors changed 396 files with 13155 insertions(+) and 8331 deletions(-) in 597 commits.
Thanks to all authors for making this release possible!

## 432.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 432.17 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.9.8.tar.gz
- size: 7753097
- md5sum: d978c17aae94d79f9d1f26b547bc46fe
- sha1: 9725bfd6fca832ed472290e9de3711e01e9bfe54
- sha256: b1e8908c138b84e788fdff25eab1c2b07e0b422a5fd1667814539ea02f151c58

The release tarball is also available signed using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 432.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 433

# 0.9.9 Release

- guid: 3177C958-9473-41BA-9918-A56A18CF20E8
- author: Mihael Pranjić
- pubDate: Thu, 10 Mar 2022 07:40:35 +0100
- shortDesc: Bug Fixes, Java Plugins, Elektra 1.0 Decisions

We are proud to release Elektra 0.9.9.

## 433.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 433.2 Highlights

- Bug fixing in FLOSS course
- Java plugins
- 1.0 decisions

### 433.2.1 Bug Fixing in FLOSS Course

As you will read in this release notes, a massive amount of bugs were fixed within this release. Many of them were resolved from students participating in a `university course about FLOSS` This demonstrates that homework of students can be very useful and in public service.
In the upcoming term there will be a course about `configuration management` in which Elektra will also be used and improved upon.

### 433.2.2 Java Plugins

The version of the `process` plugin, makes it much easier to implement plugins in Java. You can now call an implementation `org.libelektra.Plugin` via `process` and the `org.libelektra.process.Plugin↩ Process` class. To mount Java plugins the new helper script `kdb mount-java` can be used.
For more information take a look at `the updated tutorial` and the new man page for `kdb mount-java`.

### 433.2.3 1.0 Decisions

With this release we greatly updated our decisions for the 1.0 release. This brings us one big step closer to 1.0.

## 433.3 Plugins

The following section lists news about the `plugins` we updated in this release.

### 433.3.1 filecheck

- Removed unused variable that threw an error in filecheck.c. _(Vaibhav Ganesh @flackojr)_

### 433.3.2 mmapstorage

- Removed unused variable that threw an error in mmapstorage.c. _(Vaibhav Ganesh @flackojr)_

### 433.3.3 csvstorage

- Add `array` metakey to the parentKey of imported Keys _(@muskater)_ _(@4ydan)_ _(@lawli3t)_

### 433.3.4 specload

- Change and move `keyCompareMeta (const Key * k1, const Key * k2)` from `src/libs/elektra/keyt` to `src/plugins/specload/specload.c` and integrate functionality of keyCompare (const Key _ key1, const Key _ key2) into `isChangeAllowed (Key * oldKey, Key * newKey)`, because that is the only place where it was used. _(@flo91)_

### 433.3.5 uname

- Minor improvement of source code readability in uname.c _(@lawli3t)_

### 433.3.6 quickdump

- Fixed an issue with type-limits on ARM32 (see issue #4217). _(Klemens Böswirth @kodebach)_

### 433.3.7 dump

- The exported functions `serialize` and `unserialize` have been renamed to `serialize` and `unserialize`. _(Klemens Böswirth @kodebach)_
- New exported functions `int fserialize(KeySet * ks, FILE * file, Key * error↩ Key)` and `int funserialize(KeySet * ks, FILE * file, Key * errorKey)` have been added. These are wrappers around `serialize` and `unserialize` that allow calling from C with a standard `FILE *`. _(Klemens Böswirth @kodebach)_

### 433.3.8 process

- The plugin was completely rewritten. The new version is incompatible with the old version. The new plugin that uses a simple protocol to allow an external application to act as a plugin. This can for example be used to write plugins in Java without going through JNI. _(Klemens Böswirth @kodebach)_

## 433.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 433.4.1 Compatibility

- Remove the deprecated flags `KEY_NAME` and `KEY_COMMENT` (closes issue #3152) _(Florian Lindner @flo91)_

### 433.4.2 Core

- `KeySet` now also has a reference counter like `Key`. The new functions `ksIncRef` and `ksDecRef` behave like their counterparts `keyIncRef` and `keyDecRef`. `ksDel` also behaves like `keyDel` in regard to reference counting, i.e. it does nothing unless the reference count is 0. The reference counting is very useful for bindings (especially with automatic garbage collection). _(Klemens Böswirth)_
- Clarified that our reference counting mechanism is more related to a shared lock than to the concept of shared ownership. _(Klemens Böswirth)_
- Both the reference count for `Key` and for `KeySet` now use `uint16_t` to reduce memory usage. `Key` previously used `size_t`. _(Klemens Böswirth)_
- Reorder `Key` and `KeySet` struct members to aviod padding and make space for a new `uint16_t` member, reserved for future use. _(Mihael Pranjić @mpranj)_
- Improve `keyReplacePrefix` by using new `keyCopy` function instead of manually copying the name of the `Key` _(@lawli3t)_
- Added else error to core for elektraGetCheckUpdateNeeded _(Aydan Ghazani @4ydan)_
- Include NULL terminators in hashing to avoid collisions _(@lawli3t)_
- Fix check for valid namespace in keyname creation _(@JakobWonisch)_
- Fix `keyCopyMeta` not deleting non existant keys in destination (see #3981) _(@JakobWonisch)_
- The `ELEKTRA_ERROR_*_NAME` and `ELEKTRA_WARNING_*_NAME` constants have been removed from the public API. Use `ELEKTRA_ERROR_*` and `ELEKTRA_WARNING_*` instead. _(Klemens Böswirth @kodebach)_
- Fixed a bug that prevented the creation of cascading keys whose name contains a colon (`:`). _(Klemens Böswirth @kodebach)_

## 433.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 433.5.1 Java binding

- Integrated the `HelloElektra` example as Gradle subproject to allow it to directly depend on the current binding _(Michael Tucek)_
- Extend `HelloElektra` example with cutpoint and value setting example _(@JakobWonisch)_
- Updated Gradle to 7.4. _(Mihael Pranjić @mpranj)_
- Added integration with the new `process` plugin. _(Klemens Böswirth @kodebach)_
- Integrated the `HelloElektra` example as gradle subproject to allow it to directly depend on the current binding _(Michael Tucek)_
- Add `LinkChecker` Java Plugin. _(@aaronabebe)_

### 433.5.2 FUSE Binding

- Added check for existence of accessed path before opening new file descriptor _(@lawli3t)_

### 433.5.3 Python Binding

- Added examples for append, extend and remove keysets in python. _(@4ydan)_

## 433.6 Tools

- Implement `kdb validate <key>`, collect warnings and errors while kdb.get() and kdb.set(), see #3674 _(@flo91)_, _(@JakobWonisch)_
- Remove names from kdb mount _(@JakobWonisch)_
- Add `kdb mount-java` helper script for mounting Java plugins _(Klemens Böswirth @kodebach)_

## 433.7 Scripts

- Updated `reformat-c` script to use clang-format version 13. _(Mihael Pranjić @mpranj)_
- Fix bug where the PATH environment variable would get overwritten in some of the Docker images. Reduce image size _(Ivaylo Ivanov)_
- Allow JSON to be also written as json. _(@muskater)_

## 433.8 Documentation

- Integrate missing pages to website _(Ivaylo Ivanov)_
- Improved compilation documentation _(Ivaylo Ivanov)_
- Fix Links in README.md and add small clarifications. _(Markus Raab)_
- Remove previous authors. _(Markus Raab)_
- Add pre/postconditions and invariants to module keytest _(@lawli3t)_
- Updated the news template. _(Mihael Pranjić @mpranj)_
- Update and improve tutorial and in-code comments for high-level API _(Tobias Schubert @qwepoizt)_
- Improve documentation of opts library _(Tobias Schubert @qwepoizt)_
- Update tutorial "High-level API (with code-generation)" to reflect change of `loadConfiguration()`'s signature in release 0.9.5 _(Tobias Schubert @qwepoizt)_
- Add pre/postconditions and invariants to module keyvalue _(@lawli3t)_
- Update and improve inline documentation of `kdb gen`. _(Tobias Schubert @qwepoizt)_
- Fix broken links. _(Robert Sowula)_
- Emphasize that `type` is required when the HL API is used. _(Tobias Schubert @qwepoizt)_
- Add debugging tutorial. _(Tobias Schubert @qwepoizt)_
- Improve wording and formatting of DESIGN.md _(@lawli3t)_
- Correct various typing-, spelling- and grammar-errors in the .md-files in the directory doc and its subdirectories. _(Florian Lindner @flo91)_
- Make version description in key names man page consistent _(@JakobWonisch)_
- Fix typo in elektra-backends man page _(@JakobWonisch)_
- Fix readability in bootstrapping man page _(JakobWonisch)_
- Explained in the docker test tutorial how to run the container with podman instead of docker. _(@muskater)_
- Add a new example on how to use keyCopy. _(@muskater)_
- Fix small error in the "Get Started" guide: the build and test command used a wrong directory and would not work if they were copy and pasted. _(@muskater)_

- Added verification to the "Arrays" tutorial _(Ivaylo Ivanov)_
- Remove deprecated `type=int` from `.ini` files _(Ivaylo Ivanov)_
- Added verification to the "Validation" tutorial _(Ivaylo Ivanov)_
- Fix some typos in the "Getting Started" page _(Ivaylo Ivanov)_
- Added debian buster tutorial to python bindings tutorial _(@4ydan)_
- Made the debian tutorial a bit more precise and removed sudo command _(@4ydan)_
- Fixed some typos in the "namespaces.md" documentation _(@muskater)_
- Fix an error and some overmatching problems in scripts/spelling.sed and fix errors in documentation (by running the scripts/dev/fix-spelling script) _(Florian Lindner @flo91)_
- Added some improvements to the core API documentation _(@muskater)_
- Update and improve the CLion tutorial (doc/tutorials/contributing-clion.md), add screenshots _(@flo91)_
- Improve documentation for storage plugins _(@lawli3t)_
- Add list of sources mentioning or linking to Elektra _(@JakobWonisch)_
- Linked to the installation instruction of the webui in its README file and added references to Docker in the get-started-guide. _(@muskater)_ _(@lawli3t)_ _(Aydan Ghazani @4ydan)_
- Added screenshots and a quick walk through in the Qt-GUI README. _(@muskater)_ _(@lawli3t)_ _(Aydan Ghazani @4ydan)_
- Improve example for kdb-restore man page and fix typos _(@JakobWonisch)_
- Fix some typos in the "Getting Started" page _(Ivaylo Ivanov)_
- Add screenshots with hints to CLion PR tutorial _(@JakobWonisch)_
- Fix typo in elektra-backends man page _(@JakobWonisch)_
- Expanded the webside guide for easier understanding and linked to cmake.org. _(Philipp Nirnberger @nirnberger)_
- Fix small error in CLion tutorial: CMake options would create a directory named ~ in home directory _(Maximilian Irlinger @atmaxinger)_

## 433.9 Tests

- Cleanup tests/linkchecker.whitelist and fix off-by-1 bug of the counter in the scripts/link-checker script (increase counter before printf) _(Florian Lindner @flo91)_
- Add tests for the intercept/env binding _(Ivaylo Ivanov)_
- add and improve checks in scripts/sed _(Florian Lindner @flo91)_
- Change the cpp Key-class (key.hpp) to check the return values of the called c-functions and throw exceptions if values that indicate an error are returned + add tests that check for this exceptions _(Florian Lindner @flo91)_
- Added more test cases for the keyCopy function _(@muskater)_
- Add exception tests for key C++ bindings _(Ivaylo Ivanov)_
- Added a shell script and a task that checks whether the filenames of newly added files are compliant with the convention. It is executed by the cirrus CI as well as the Jenkins CI _(@muskater)_
- Add a new shellrecoder test to `doc/tutorials/merge.md` _(Florian Lindner @flo91)_
- Added the possibility to enable all the shell recorder test cases for all plugins and fixed some plugin README files in process _(@muskater)_
- Convert example in doc/help/kdb-test.md to shell recorder test _(@JakobWonisch)_

## 433.10 Packaging

- Add initial flatpak package. See `scripts/flatpak/README.md` and `scripts/flatpak/org.libelektra.kdb.yaml`for more info _(Ivaylo Ivanov)_
- Remove hard coded SWIG 3.0 paths. _(Mihael Pranjić @mpranj)_

## 433.11 Build

### 433.11.1 CMake

- Marked certain variables as advanced and separate user-modifiable and unaccessible variables. _(Vaibhav Ganesh @flackojr)_

### 433.11.2 Docker

- Add Fedora 35 images. _(Mihael Pranjić @mpranj)_

## 433.12 Infrastructure

### 433.12.1 Jenkins

- Replace Fedora 33 builds with Fedora 34, and Fedora 34 builds with Fedora 35. _(Mihael Pranjić @mpranj)_

## 433.13 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Update npm dependencies, add forked and update angular-marked module. _(Mihael Pranjić @mpranj)_
- Remove links to Travis CI and replace them with GitHub Actions (with badge). _(Mihael Pranjić @mpranj)_

## 433.14 Other

- Make Elektra `reuse` reuse compliant _(Ivaylo Ivanov)_

## 433.15 Outlook

We are working on following new topics since the last release:

- Ansible-Elektra _(Lukas Hartl)_
- Recording Configuration _(Maximilian Irlinger)_

Furthermore, we are still working on following topics:

- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth)_
- Improve Java Development Experience _(Michael Tucek)_
- KDB access using FUSE _(Alexander Firbas)_
- Shell completion _(Ulrike Schäfer)_
- Elektrify KDE and GNOME _(Mihael Pranjić)_

## 433.16 Statistics

We closed 90 issues for this release.
About 29 authors changed 491 files with 17997 insertions(+) and 6089 deletions(-) in 648 commits.
Thanks to all authors for making this release possible!

## 433.17 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 433.18 Get the Release!

You can download the release from here or GitHub
The hashsums are:

- name: elektra-0.9.9.tar.gz
- size: 8878567
- md5sum: f5109eb0c96fb4164a5437bdebc3bf79
- sha1: a08df79301d56dd8f3711efa1b78b5a4d003d42f
- sha256: 834da360170daa632bbb46dd2e819271327dce1c51be1d7bb2ec22311ded54cb

The release tarball is also available signed using GnuPG from here or on GitHub
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 433.19 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 434

# 0.9.10 Release

- guid: CC66FD33-7491-4BFA-975A-36FAB67D45D6
- author: Mihael Pranjić
- pubDate: Sat, 09 Jul 2022 09:40:18 +0200
- shortDesc: Kotlin Binding, Remove Internal Iterators

We are proud to release Elektra 0.9.10.

## 434.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 434.2 Highlights

- Kotlin Binding
- Remove internal iterators

### 434.2.1 Kotline Binding

We created a new Binding for Kotlin with convenience functions and various utilities. There is also the possibility to convert KeySets to Kotlin data classes or collections and back. Read the Kotlin Readme for more information.
A big thanks to _(@Gratla and @mandoway)_ for this beautiful work.

### 434.2.2 Remove Internal Iterators

In Elektra there are currently two different ways to iterate over KeySets. The so-called "internal" iterator is, however, inferior and creates several problems, e.g. it was a side effect to be considered for every function call that involved a KeySet.
With this release, we started removing the internal iterators by removing `keyRewindMeta`, `keyCurrentMeta`, `ksHead`, and `ksTail` functions. The external iterators are now the way to go, see Iterators.
A huge thanks to _(Florian Lindner @flo91)_ for doing this thankless cleanup task.

## 434.3 Plugins

The following section lists news about the plugins we updated in this release. Overall changes:

- We changed all plugins, except directoryvalue to use external iteration of KeySets _(Florian Lindner @flo91)_

### 434.3.1 Python

- Added .pop(), .cut(), .head() and .tail() examples to keySet example _(Lukas Hartl @lukashartl, Leonard Guelmino @leothetryhard)_
- Added a new DNS plugin fully written in Python _(Lukas Hartl @lukashartl, Leonard Guelmino @leothetryhard)_

### 434.3.2 lineendings - Plugin

- Enable emitting of warnings during kdbGet(), refactor and update methods and return values to match the conventions (e.g. #defined constants for return values) _(Michael Langhammer @milangs, Florian Lindner @flo91)_

### 434.3.3 date

- Exclude the tests for formats that require GNU extensions of strptime on non-GNU systems. _(@kodebach)_

### 434.3.4 Length

- Warnings are now added on kdb get _(@mandoway)_

### 434.3.5 Curlget

- Removed usages (and contents) of VERBOSE macro _(@mandoway)_

### 434.3.6 Sorted

- Added new validation plugin: Sorted. It checks whether an Elektra array is sorted by its value or a given key in a configurable direction _(@mandoway @Gratla)_

### 434.3.7 mini

- Fix usage of bitwise operator with boolean operands. _(Mihael Pranjić @mpranj)_

## 434.4 Libraries

The text below summarizes updates to the C (and C++)-based libraries of Elektra.

### 434.4.1 Compatibility

- Remove keyRewindMeta, keyCurrentMeta, ksHead, and ksTail functions for internal iteration of Keysets and Metadata of Keys _(Florian Lindner @flo91)_

### 434.4.2 Core

- Removed mentions of VERBOSE and replaced debug prints with the logger _(@mandoway)_

## 434.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

- Remove internal iterators for SWIG (Python, Lua, Ruby) and go-bindings _(Florian Lindner @flo91)_

### 434.5.1 Java

- Implement NavigableSet in JNA KeySet _(Burkhard Hampl @bhampl)_
- Added a Java example for meta keys and arrays _(@mandoway)_
- Added examples to HelloElektra.java _(Leonard Guelmino @leothetryhard, Lukas Hartl @lukashartl)_
- Added example which shows how to add a basename for a key. _(Philipp Leeb @Gratla)_
- Introduced Key::setNull, ReadableKey::isNull _(Michael Tucek @tucek)_
- Fixed Key::setBoolean, ReadableKey::isBoolean _(Michael Tucek @tucek)_
- Fixed Java Whitelist plugin tests _(Michael Tucek @tucek)_
- Fixed missing Javadoc in Java Sorted plugin _(Michael Tucek @tucek)_

### 434.5.2 Ruby

- Replace `NULL` in `rb_funcall` with `Qnil` to avoid compiler errors/warnings on some systems. _↩ (@kodebach)_

### 434.5.3 Kotlin

- Added new JNA subproject which builds an Elektra extension library for Kotlin _(@mandoway & @Gratla)_
- Added get(), getOrNull() extension with type inference for primitive types _(@mandoway)_
- Added set() extension with type inference for primitive types _(@mandoway)_
- Added keySet serialization capabilities (to any format and data classes, with array support) _(@mandoway)_
- Added keyOf() extension and keyOf{} builder for key creation _(@Gratla)_
- Added keySetOf() extension and keySetOf{} builder for keySet creation _(@Gratla)_
- Added withKDB() extension which wraps the try block _(@Gratla)_
- Added nameParts extension value which provides a sequence of key name parts _(@mandoway)_
- Added various utility functions like Key.isEmpty, Key.getMetaOrNull, ... _(@Gratla & @mandoway)_
- Added example project for kotlin binding _(@Gratla & @mandoway)_
- Added lookupOrThrow(), lookupOrNull(), and get operator to search for keys in KeySets without Java Optionals _(@mandoway)_
- Fixed setting null, by using new JNA setNull() function _(@mandoway)_

### 434.5.4 Python

- Deleted occurrences of removed property key.fullname _(@mandoway)_

### 434.5.5 CPP

- Removed mentions of VERBOSE _(@mandoway)_

### 434.5.6 Python

- Deleted occurrences of removed property key.fullname _(@mandoway)_

### 434.5.7 CPP

- Removed mentions of VERBOSE _(@mandoway)_

## 434.6 Tools

### 434.6.1 elektrad

- improve logging in `elektrad` _(Lukas Hartl @lukashartl, Leonard Guelmino @leothetryhard)_
- Update `elektrad` to use last version of the `go`-bindings without internal iterators for `Keyset`s and Metadata _(Florian Lindner @flo91)_

### 434.6.2 <tt>webui</tt>

- fix issues from namespace-overhaul _(Lukas Hartl @lukashartl, Leonard Guelmino @leothetryhard)_
- apply non-breaking updates to packages _(Leonard Guelmino @leothetryhard, Lukas Hartl @lukashartl)_

### 434.6.3 <tt>webd</tt>

- fix path building for requests to `elektrad` _(Lukas Hartl @lukashartl, Leonard Guelmino @leothetryhard)_

### 434.6.4 QT GUI

- Removed mentions of VERBOSE _(@mandoway)_
- Fixed dependency for Debian packages _(Markus Raab)_

## 434.7 Scripts

- Fix `kdb reset`. _(Markus Raab)_

## 434.8 Documentation

- Small readability improvement _(@Toniboyyy)_
- Python: add guide for Debian 11 (bullseye) _(Lukas Hartl @lukashartl)_
- Fix some errors in the tutorials `Cascading Lookups` and `Command-line Options` _(Florian Lindner @flo91)_
- Extend and update the tutorial for writing specifications, add section about using specs in production _(Florian Lindner @flo91)_
- Tutorial: add cleanup section to the specification tutorial _(Lukas Hartl @lukashartl)_ and _(@leothetryhard)_
- Add readme-file Iterators about cm2022s project showcasing usage in various programming languages _(Florian Lindner @flo91 and Michael Langhammer @Milangs)_
- Updated elektra-web installation manual (doc/tutorials/install-webui.md) _(Leonard Guelmino @leothetryhard, Lukas Hartl @lukashartl)_
- Improve jna documentation _(Burkhard Hampl @bhampl)_

- Add Stream API example in Java binding documentation _(Richard Stöckl @Eiskasten)_
- Minor readability improvement in `CODING.md` _(@loessberth)_
- Fix dead link and compile instructions _(Burkhard Hampl @bhampl)_
- Update links from certificate section _(Richard Stöckl @Eiskasten)_
- Fix wrong KDBException reference in java tutorial and improve it _(Burkhard Hampl @bhampl and Richard Stöckl @Eiskasten)_
- Update FAQ. _(Markus Raab)_

### 434.8.1 Tutorials

- The tutorial for Contributing from Windows has been updated. _(@kodebach)_
- The tutorial for CLion now contains a section for setting up the WSL compiler _(@mandoway)_
- Rephrased sentence in code-generator.md to enhance readability _(@Gratla)_

## 434.9 Tests

- Add tests for the Error/Warnings-Factory in libtools _(Florian Lindner @flo91)_
- Add tests for keySet in the python binary _(Lukas Hartl @lukashartl, Leonard Guelmino @leothetryhard)_
- Added test for JNA KDB which checks if both get-method implementations return the same result. _(Philipp Leeb @Gratla)_

## 434.10 Build

### 434.10.1 CMake

- CMake now automatically detects all JNA plugins that are added to Gradle. _(@kodebach)_

## 434.11 Infrastructure

### 434.11.1 Jenkins

- make copying of artifacts much faster _(Lukas Hartl)_
- fixed several problems _(Lukas Hartl)_

### 434.11.2 Cirrus && GitHub Actions

- Fix wrong path for clang builds on macOS. _(Mihael Pranjić @mpranj)_

### 434.11.3 Git

- We added a `.gitattributes` file to make it easier to build Elektra with WSL. _(@kodebach)_

### 434.11.4 GitHub

- Added dependabot configuration _(Lukas Hartl @lukashartl)_

## 434.12 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Overhauled the `Get Started` page by adding a brief `kdb` introduction. _(@Milangs)_

## 434.13 Outlook

We are currently working on following topics:

- 1.0 API _(Stefan Hanreich)_ and _(Klemens Böswirth @kodebach)_
- Elektrify KDE and GNOME _(Mihael Pranjić @mpranj)_
- Elektrify XFCE _(Richard Stöckl @Eiskasten)_
- Mounting SQL databases _(Florian Lindner @flo91)_
- Recording Configuration _(Maximilian Irlinger)_
- Ansible-Elektra _(Lukas Hartl)_
- Improving Build Server Infrastructure _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Improve Java Development Experience _(Michael Tucek)_
- KDB access using FUSE _(Alexander Firbas)_
- Shell completion _(Ulrike Schäfer)_
- Rewriting tools in C _(Florian Lindner @flo91)_, _(Maximilian Irlinger)_ and _(Richard Stöckl @Eiskasten)_.

## 434.14 Statistics

We closed `51 issues` for this release.
About 25 authors changed 376 files with 39350 insertions(+) and 13609 deletions(-) in 393 commits.
Thanks to all authors for making this release possible!

## 434.15 Join the Initiative!

We welcome new contributors! Read `here` about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our `issue tracker`.

## 434.16 Get the Release!

You can download the release from `here` or `GitHub`
The `hashsums are:`

- name: elektra-0.9.10.tar.gz
- size: 9150058
- md5sum: c77f5bcf4e8202fd6d8b6ad6e7c841f4
- sha1: d63c24ea6c666b02d0bd9f059f2d73f96009496d
- sha256: ee50fb5e9814b45a8e99f39435b1461d4b7a7daa27eee240bdbfed98f2c4c0f5

The release tarball is also available signed using GnuPG from `here` or on `GitHub`
The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found `here` or on `GitHub`.

## 434.17 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 435

# 0.9.11 Release

- guid: 1390D163-9CEC-493D-839D-3930B9FFB6C4
- author: Mihael Pranjić
- pubDate: Wed, 05 Oct 2022 08:14:35 +0200
- shortDesc: New Backend System, Opensesame Application, FLOSS Course

We are proud to release Elektra 0.9.11.

## 435.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our Docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 435.2 Highlights

- Preparing new-backend merge
- Olimex
- Improving Elektra in FLOSS course

### 435.2.1 Preparing new-backend merge

In a separated branch we rewrote the whole backend system. With this new backend system, backends are now also plugins, allowing backends also to be non-file-based, e.g., using databases. The main purpose of this release is to give a last stable release before master gets disrupted with a huge changeset. The next release is not to be expected in this year.
A huge thanks to _(Klemens Böswirth @kodebach)_, _(Maximilian Irlinger @atmaxinger)_ for the many changes in the branch. Thanks to _(Richard Stöckl @Eiskasten)_ for testing.

### 435.2.2 Olimex

Elektra is used for server, desktop and embedded. With this release, we strengthen our embedded mainstay, specifically in `open source hardware OSHW`. We developed a major application running on OSHW `Olimex` boards. The application is called `opensesame`. It is heavily relying on Elektra and `ansible-libelektra`. In the initial release `opensesame` already allows:

- [x] opening (garage) doors via a novel PIN entry method: you can press and release buttons in any sequence

- [x] switching on entry lights
- [x] ringing doorbells
- [x] detection of fire
- [x] report events to Nextcloud chats (English and German)

To give a smoother experience when running such an application we will develop Ansible scripts to customize the Olimex base images. They will allow changing the language, time zone, static network configuration etc.
Olimex likes this idea and will send us an A20 board. A big thanks to Olimex.

### 435.2.3 Improving Elektra in FLOSS course

Also in the upcoming term Elektra will be object of study how FLOSS initiatives work. Students will make improvements in Elektra as part of their homework, teamwork and, if chosen, also for their project. Alternatively, they can also improve other self-chosen FLOSS initiatives.

## 435.3 Plugins

The following text list news about the plugins we updated in this release.

- fix unused-but-set-variable warnings. _(Markus Raab)_

### 435.3.1 csvstorage

- Remove superfluous if-conditions that lead to a build error on Debian Unstable _(Maximilian Irlinger @atmaxinger)_

### 435.3.2 specload

- fail if either the spec or parentKey parameter of elektraSpecloadSendSpec is NULL _(@hannes99)_

## 435.4 Libraries

The text below summarizes updates to the C (and C++)-based libraries of Elektra.

- fix unused-but-set-variable warnings. _(Markus Raab)_

### 435.4.1 opts

- opts: fix possible 'free(): invalid pointer' error and add test for it _(@hannes99)_

## 435.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 435.5.1 Python

- add merging based on elektraMerge _(Maximilian Irlinger @atmaxinger)_

### 435.5.2 Rust

- start again to publish on crates.io, used by opensesame _(Markus Raab)_

## 435.6 Documentation

- Added `Documentation Guidelines` _(Markus Raab)_
- Decisions for changes to `keyIsBelow` and new `keyGetNextPart` functions _(@kodebach)_
- Apply fix spelling to more files. _(Markus Raab)_

### 435.6.1 Tutorials

- opts: use arg/help instead of arg/name _(@hannes99)_

### 435.6.2 Man Pages

- Update FAQ. _(Markus Raab)_

## 435.7 Tests

- Use GoogleTest framework v1.12.1. _(Mihael Pranjić @mpranj)_

## 435.8 Build

### 435.8.1 CMake

- Fix build with newer libgit2 versions _(Fabian Vogt)_
- We now require at least CMake `3.12` (released in July 2018). _(Maximilian Irlinger @atmaxinger)_

### 435.8.2 Docker

- Bump Alpine Linux to 3.16.0. _(Mihael Pranjić @mpranj)_
- The Docker image for building the documentation is now based on Debian Bullseye. _(Maximilian Irlinger @atmaxinger)_
- Add Fedora 36 images. _(Mihael Pranjić @mpranj)_

## 435.9 Infrastructure

### 435.9.1 Jenkins

- We no longer build and test on Ubuntu Xenial and Debian Stretch due to outdated CMake versions _(Maximilian Irlinger @atmaxinger)_
- Add Fedora 36 builds, remove Fedora 34 builds. _(Mihael Pranjić @mpranj)_

### 435.9.2 Cirrus

- Update FreeBSD images to 13.1 and 12.3 and update packages before builds. _(Mihael Pranjić @mpranj)_
- Bump Fedora builds to Fedora 36. _(Mihael Pranjić @mpranj)_

## 435.10 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date.

## 435.11 Outlook

We are currently working on following topics:

- 1.0 API _(Klemens Böswirth @kodebach)_ and _(Stefan Hanreich)_
- Elektrify KDE and GNOME _(Mihael Pranjić @mpranj)_
- Elektrify XFCE _(Richard Stöckl @Eiskasten)_
- Mounting SQL databases _(Florian Lindner @flo91)_
- Recording Configuration _(Maximilian Irlinger)_
- Ansible-Elektra _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Configure Olimex Base Images _(Maximilian Irlinger)_
- Improving Build Server Infrastructure _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Improve Java Development Experience _(Michael Tucek)_
- Rewriting tools in C _(@hannes99)_

## 435.12 Statistics

We closed 17 issues for this release.
About 11 authors changed 156 files with 4020 insertions(+) and 3298 deletions(-) in 134 commits.
Thanks to all authors for making this release possible!

## 435.13 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 435.14 Get the Release!

You can download the release from

- here or from
- GitHub

The hashsums are:

- name: elektra-0.9.11.tar.gz
- size: 9149900
- md5sum: 227094a7760f8faece1a7b8386d01fce
- sha1: a0d2f0d39c8360f67da96585993223b7f9cdebe6
- sha256: 3509adf83efdf08fa0dc5d51396772addff7d8fc82299c18b146fa4406eecff5

The release tarball is also available signed using GnuPG from

- here or on
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found here or on GitHub.

## 435.15 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 436

# 0.9.12 Release

- guid: F2193578-1773-43A9-85CA-79EA8CE48D7B
- author: Mihael Pranjić
- pubDate: Fri, 03 Mar 2023 08:07:28 +0100
- shortDesc: New Backend Logic, Copy-on-Write, FLOSS Course

We are proud to release Elektra 0.9.12.

## 436.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 436.2 Highlights

- New Backend
- Copy-on-Write
- FLOSS

### 436.2.1 New Backend

The entire logic for backends has been rewritten, to allow for more flexibility und an unlimited number of plugins. Instead of calling plugins directly, `libelektra-kdb` now only calls so-called backend plugins and special hook plugins. There is a contract between `libelektra-kdb` and the backend plugins. All backend plugins must adhere to this contract. To achieve this goal, most backend plugins will call other plugins (like `libelektra-kdb` did previously).
The logic previously implemented in `libelektra-kdb` was moved to the new default backend plugin `backend`. It works like the old system, but now also allows an unlimited number of plugins in positions where that makes sense. For example, you can have unlimited `postgetstorage` plugins, but only a single `getresolver`.
There have also been slight changes to `kdbGet` and `kdbSet`. Please read their API docs to find out, if you rely on any behavior that has been altered. You can also read the new low-level docs to find out all the intricate details.
The structure of `system:/elektra/mountpoints` changed as well. Take a look at the new docs, if you need to know details.

#### 436.2.1.1 Updating config

The mountpoint configuration format contains **breaking changes** and a manual upgrade process is needed. Follow these steps to upgrade the old mountpoint configuration to the new format:

> **Warning**: BACK UP YOUR CONFIG FILES BEFORE UPDATING! We recommend making a backup of the file printed by `kdb file system:/elektra/mountpoints` before updating your installation. In the unlikely case that the migration script fails, you can still use the information from the backup to manually recreate your mountpoints.

To update your existing `system:/elektra/mountpoints` data you can use the migration script.

> **Note**: To run the script you must have Elektra, Python (>= 3.7) and the Python binding installed. The script uses the Python binding to manipulate `Key`s and `KeySet`s, but it does not use the `kdb` CLI tool, or the `KDB` API. It is safe to run this script before, or after you update your Elektra installation.

By default, the script loads the file `/etc/kdb/elektra.ecf`. If you changed where `system:/elektra/mountpoints` is stored, you can provide an alternative path:

```
./migrate-mountpoints.py /path/to/your/mountpoints/config.file
```

> **Note**: Because the script does not use the `KDB` API it only works, if the mountpoints config file uses the default `dump` format.
>
> If your mountpoints config file is not using the `dump` format, you may still be able to use the migration script. However, in that case, you will have to use the script **before** updating your Elektra installation:
>
> 1. Run `kdb export system:/elektra/mountpoints dump` to get a copy of your mountpoints config in `dump` format
> 2. Write this data to a file and run the migration script on the file.
> 3. To get the data back in your original format you can use
>
> ```
> ./migrate-mountpoints.py /path/to/file/from/step2 | kdb convert dump your-format >
>  /path/to/converted/file
> ```
>
> 1. Run `kdb file system:/elektra/mountpoints` to find out where your mountpoint config is stored. Make sure to back up this file, before upgrading your installation.
> 2. Now upgrade your Elektra installation.
> 3. Copy the file `/path/to/converted/file` from step 3 to the location you got in step 4.

The script will read the old mountpoint configuration from the given file. It will convert the configuration and print the new version to `stdout`.
You can inspect the output to make sure, everything is in order. When you are ready to commit the changes, you can manually edit the config file, or use:

```
./migrate-mountpoints.py --output=/etc/kdb/elektra.ecf /etc/kdb/elektra.ecf
```

#### 436.2.1.2 Individual changes

- Implement `hooks`. _(Maximilian Irlinger @atmaxinger)_
- Removed old global plugins code. _(Maximilian Irlinger @atmaxinger)_
- New backend logic, based on PR #2969 by @vLesk. _(@kodebach)_
- Add script to migrate `system:/elektra/mountpoints` to new format. _(@kodebach)_

### 436.2.2 Copy-on-Write

Thanks to _(Maximilian Irlinger @atmaxinger)_ our `Key` and `KeySet` datastructures are now fully copy-on-write! This means noticeably reduced memory usage for cases where keys and keysets are copied and/or duplicated!
We ran some very promising benchmarks, each were performed with 400,000 keys. All benchmarks were executed using `valgrind --tool=massif --time-unit=B --max-snapshots=200 --threshold=0.1`.

| Benchmark | Old Implementation | Copy-on-Write | Size Reduction | Remarks |
|---|---|---|---|---|
| createkeys.↩ c | 5.3 MiB | 6.5 MiB | -22 % | |
| deepdup.c | 10.5 MiB | 8.2 MiB | 22 % | |
| large.c | 18.9 MiB | 15.3 MiB | 19 % | |
| kdb.c | 23.5 MiB | 17.8 MiB | 24 % | |
| kdbget.c | 11.0 MiB | 8.8 MiB | 20 % | |
| kdbmodify.c | 11.0 MiB | 8.8 MiB | 20 % | Same results as kdbget.c |

First, it should be noted that a single key, without counting payload, is about 50% larger with the copy-on-write implementation. This explains why the createkeys.c benchmark yields a negative reduction result. This benchmark only allocates keys, so not much improvement can be expected there. Still, as other stuff also uses heap memory, the overall memory consumption only increased by 22%, which is far less than 50%.
All other benchmarks saw meaningful reductions of heap memory used. One interesting observation is that kdbget.c and kdbmodify.c used exactly the same memory. This can most likely be explained by internal caching within the memory allocator of glibc.
We also performed runtime tests on the same benchmarks using perf stat --repeat 13 to ensure no major performance regressions occur.

| Benchmark | Old Implementation | Deviation | Copy-on-Write | Deviation | Runtime Increase |
|---|---|---|---|---|---|
| createkeys.↩ c | 0.209572 s | 0.36 % | 0.21987 s | 0.77 % | 4.9 % |
| deepdup.c | 0.23025 s | 0.47 % | 0.231804 s | 0.32 % | 0.6 % |
| large.c | 1.14038 s | 0.21 % | 1.14837 s | 0.21 % | 0.7 % |
| kdb.c | 1.9270 s | 2.63 % | 1.93354 s | 0.17 % | 0.3 % |
| kdbget.c | 0.145663 s | 0.17 % | 0.15763 s | 0.70 % | 8.2 % |
| kdbmodify.c | 0.146506 s | 0.19 % | 0.156347 s | 0.15 % | 6.7 % |

Overall, the runtime performance hit is less than 10%. The more a program does, the less the additional overhead of the copy-on-write algorithms matter. One interesting detail is that keyCopy and keyDup have become quite a bit faster. This can be seen by comparing the differences between createkeys.c and deepdup.c. The differences are 21 ms for the old implementation and 12 ms for the copy-on-write implementation.

### 436.2.3 FLOSS

A Free/Libre and Open Source Software (FLOSS) Initiative couldn't survive with many small contributions fixing annoying problems. This release also contains all contributions done via one term of the FLOSS course. The success was tremendous, as shown in the rest of the release notes.
A big thanks to the students for their contributions!

## 436.3 Plugins

The following text lists news about the plugins we updated in this release.

### 436.3.1 yajl

- Fix an issue where trying to set invalid meta-keys won't show an error. _(Juri Schreib @Bujuhu)_

### 436.3.2 list

- Removed the outdated list plugin. _(Maximilian Irlinger @atmaxinger)_ (Was only needed for global plugins, which are now replaced by hooks.)

### 436.3.3 logchange

- Made logchange a notification-send hook plugin. _(Maximilian Irlinger @atmaxinger)_

#### 436.3.4 toml

- Fix bug, where meta-keys that cannot be inserted don't report an error. _(@Bujuhu)_

#### 436.3.5 uname

- Add error handling if uname call fails. _(Richard Stöckl @Eiskasten)_

#### 436.3.6 quickdump

- elektraQuickdumpSet: don't fclose if stdout. _(@hannes99)_

#### 436.3.7 blockresolver

- Add encoding test for blockresolver read. _(@dtdirect)_
- Refactor and restructure blockresolver. _(@dtdirect)_

#### 436.3.8 desktop

- Add a unit test. _(Richard Stöckl @Eiskasten)_

#### 436.3.9 mini

- Fix a bug where writing meta-keys will fail silently. _(Juri Schreib @Bujuhu)_

#### 436.3.10 mmapstorage

- Remove code duplication in the data block calculation. _(Richard Stöckl @Eiskasten)_

#### 436.3.11 network

- Add a retry mechanism. _(Richard Stöckl @Eiskasten)_

#### 436.3.12 xfconf

- Add xfconf storage plugin with the ability to read and write to xfconf channels. _(Richard Stöckl @Eiskasten)_
- Make xfconf valgrind suppressions more flexible to lib updates. _(Mihael Pranjić @mpranj)_

#### 436.3.13 date

- Fix an issue with validationg RFC 822 date-times. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Improve Code Coverage. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_

#### 436.3.14 csvstorage

- Fix a bug where writing unkown meta-keys will fail silently. _(Juri Schreib @Bujuhu)_

### 436.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

#### 436.4.1 Compatibility

- Global plugins do not work anymore, use `hooks` instead.

### 436.4.2 Core

- The Key and KeySet datastructures are now fully copy-on-write. _(Maximilian Irlinger @atmaxinger)_
- `keyCopy` now only allocates additional memory if `KEY_CP_META` or `KEY_CP_ALL` is used. _(Maximilian Irlinger @atmaxinger)_
- Check for circular links (overrides). _(@0x6178656c)_

### 436.4.3 io

- Check file flags for `elektraIoFdSetFlags`: file flags must be exactly one of: read only, write only or read write. _(Richard Stöckl @Eiskasten)_

### 436.4.4 Merge

- Add methods `elektraMergeGetConflictingKeys` and `elektraMergeIsKeyConflicting` to check which keys were causing a merge conflict. _(Maximilian Irlinger @atmaxinger)_

## 436.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 436.5.1 intercept/env

- Remove fallback code. _(Markus Raab)_
- Command-line functionality is broken due to new-backend differences.

### 436.5.2 intercept/fs

- Use `KDB_MAX_PATH_LENGTH` for better portability. _(Markus Raab)_

### 436.5.3 jna

- Documentation: Improved build instructions. _(@Bujuhu)_
- Add validation on get for whitelist plugin. _(@Bujuhu)_
- Upgrade Gradle to 8.0.1. _(Mihael Pranjić @mpranj)_

### 436.5.4 rust

- Fix "feature `resolver` is required" error. _(Markus Raab)_

### 436.5.5 elixir

- Initial release of the Elixir binding. _(@0x6178656c)_
- Mark tests as `memleak`. _(@0x6178656c)_

## 436.6 Tools

### 436.6.1 kdb

- Removed `global-mount` and `global-umount` commands. _(Maximilian Irlinger @atmaxinger)_
- Fixed SIGSEGV when using `kdb find` without argument. _(Christian Jonak-Moechel @joni1993)_

### 436.6.2 elektrad

- Removed leftover package-lock.json file. _(stefnotch)_

## 436.7 Scripts

- Added `automatic spelling corrections` for changeset. _(Maximilian Irlinger @atmaxinger)_
- Introduce shebang-conventions. _(@0x6178656c)_
- Apply auto-fixes from shellcheck. _(@0x6178656c)_
- Use prettier 2.8.4. _(Mihael Pranjić @mpranj)_
- Add scripts to enable and disable pre-commit hooks. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Only let http links pass the check if whitelisted. _(Richard Stöckl @Eiskasten)_
- Link Checker: Add documentation for the usage and how it behaves. _(Richard Stöckl @Eiskasten)_
- Sed: Add spelling correction for "key-value storage". _(@Bujuhu)_
- Fix/extends some shell recorder tests. _(@Joni1993)_
- Use clang-format v15. _(Mihael Pranjić @mpranj)_
- Fix warning parsing in shell recorder. _(@Joni1993)_
- Replaced `egrep` by `grep -E`. _(@0x6178656c and @janldeboer)_
- Add audit-dependencies script to check for vulnerabilities for npm depndencies. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_

## 436.8 Documentation

- Restructured contrib/api. _(Markus Raab)_.
- Improve page on compilation. _(@0x6178656c)_
- Improve page for bindings. _(@0x6178656c)_
- Improve page for getting started. _(@stefnotch)_
- Remove version number from docker README and replace it with latest. _(@Joni1993)_
- Fix grammar for `elektra-granularity.md`. _(@dtdirect)_
- Rephrase sections in `doc/dev/error-\*`. _(@dtdirect)_
- Improve Git.md. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Unify spelling of man pages. _(@stefnotch)_ _(@janldeboer)_
- Extend consistency check `check_doc.sh` to work for contrib, dev and tutorials. _(@Joni1993)_
- Fix internal links. _(@0x6178656c)_
- Update AUR Link from `elektra` to `libelektra` package. _(@Bujuhu)_
- Update example Ansible playbook in VISION.md. _(@Bujuhu)_
- Harmonize spelling of Git. _(@Joni1993)_
- Update packaging instructions for Fedora. _(@0x6178656c)_
- Improve use of gender. _(@0x6178656c)_

- Fix some minor mistakes in CONTRIBUTING.md. _(@Joni1993)_
- Fix various spelling errors. _(@Joni1993)_
- Denoted package names & global variable names in INSTALL.md as `Code`. _(@janldeboer)_
- Improve readability of doc/tutorials/highlevel.md. _(@deoknats861)_
- Improve reference to Podman documentation. _(@0x6178656c)_
- Unify spelling. _(@Joni1993)_
- Fix typo in dev/hooks.md. _(@dtdirect)_
- Remove unused images from doc/images. _(@dtdirect)_
- Fixed Coverage Badge Link. _(@janldeboer)_
- Improve CONTRIBUTING doc. _(Juri Schreib @Bujuhu)_ and _(Nikola Prvulovic @Dynamichost96)_
- Update Doxyfile with Doxygen 1.9.4. _(@0x6178656c)_
- Add project logo to Doxygen in Doxyfile. _(@dtdirect)_
- Add mermaid.js to the project using doxygen-mermaid. _(@dtdirect)_
- Create diagrams in mermaid.js to use in doxygen. _(@dtdirect)_
- Create README for Doxygen and Mermaid JS. _(@dtdirect)_
- Tutorial: Add automatic validation to Docker tutorial _(Schreib @Bujuhu)_ _(Nikola Prvulovic @↩ Dynamichost96)_
- Add mention of audit-dependencies script in doc/todo/RELEASE.md. _(@Bujuhu)_
- Move note in GETSTARTED.md. _(@Joni1993)_
- Use `code` blocks to prevent Markdown from falsy rendering LaTeX. _(@stefnotch)_, _(@janldeboer)_
- Fix broken links in use cases for KDB after files were renamed. _(Florian Lindner @flo91)_
- Replace http links with https. _(Richard Stöckl @Eiskasten)_
- Enhance notifications.md in doc/tutorial. _(@dtdirect)_
- Add tutorial how to suppress memleaks in plugins from dependencies. _(Richard Stöckl @Eiskasten)_
- Write about history to make plans of Elektra's adoption more clear. _(Markus Raab)_

### 436.8.1 Use Cases

- Improve use cases Template. _(@kodebach and Markus Raab)_
- Use cases for KDB. _(@kodebach)_
- Use cases for `libelektra-core`. _(@kodebach)_

### 436.8.2 Decisions

- Decide and implement decision process. _(Markus Raab)_
- Decided library split. _(@kodebach)_
- Decided `decision process`. _(Markus Raab)_
- Draft for man pages. _(Markus Raab)_
- Add decision for change tracking. _(Maximilian Irlinger @atmaxinger)_
- Create decision for allowed and prohibited operation seqences. _(Maximilian Irlinger @atmaxinger)_
- Add decisions about location of headers and use of `#include` in the repo. _(@kodebach)_
- Add decision about metadata semantics. _(@kodebach)_
- Many small fixes to adapt to documentation guidelines and new decision process. _(Markus Raab)_
- Add decision for read-only keynames. _(Maximilian Irlinger @atmaxinger)_
- Revive keyname decision. _(@kodebach)_
- Add decisions for constructor functions and builder functions. _(@kodebach)_
- Add decision for copy-on-write and provide implementation suggestions. _(Maximilian Irlinger @atmaxinger)_
- Update internal cache. _(Markus Raab)_ _(@kodebach)_
- Create transformations. _(Maximilian Irlinger @atmaxinger)_
- Replace TOC-style README.md with folders and generate HTML for website. _(@kodebach)_
- Restructured decisions directories based on new agreed-upon steps. _(Maximilian Irlinger @atmaxinger)_
- Decision for types of `KeySet`s. _(@kodebach)_
- Added `Documentation Guidelines`. _(Markus Raab)_
- Add links and formatting to documents affected by PR#4492 (Document Guidelines) and rephrase some parts. _(Florian Lindner @flo91)_
- Decisions for changes to `keyIsBelow` and new `keyGetNextPart` functions. _(@kodebach)_
- Apply fix spelling to more files. _(Markus Raab)_

### 436.8.3 Tutorials

- Add tutorial for manual installation from the AUR on Arch Linux. _(@Bujuhu)_
- Add Markdown shell recorder validation to install.webui.md. _(@deoknats861)_
- Fix the outdated array tutorial. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Reinstate mounting tutorial. _(@Bujuhu)_
- Make namespaces tutorial verifiable. _(@0x6178656c)_
- Move Podman-related information to a dedicated page. _(@0x6178656c)_

### 436.8.4 Man Pages

- Update man page (patch) as suggested by the CI to fix CI error on master. _(Florian Lindner @flo91)_
- Added links to the website & webui after further reading. _(Philipp Nirnberger @nirnberger)_
- Upgrade `ronn-ng` to 0.10.1.pre3. _(Mihael Pranjić @mpranj)_

## 436.9 Tests

- Fix an Issue where scripts/dev/fix-spelling does not work, if a resolved path contains whitespaces. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Rename scripts/sed to `scripts/spelling.sed`. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Add `memleak` label to `test_getenv`. _(@0x6178656c)_
- Add test using shellcheck. _(@0x6178656c)_
- Remove `--rerun-failed` from `run_*` scripts. _(@kodebach)_
- Fix paths for icheck test. _(Mihael Pranjić @mpranj)_

### 436.9.1 Shell Recorder

- Add check if file exists. _(@0x6178656c)_

## 436.10 Packaging

- Add missing new `backend` plugin to components of libelektra package. _(Mihael Pranjić @mpranj)_

## 436.11 Build

### 436.11.1 CMake

- Fix warning for CMP0115. _(0x6178656c)_
- Change Doxygen configuration for LaTeX. _(0x6178656c)_
- Fix developer warning for package DISCOUNT. _(Dennis Toth @dtdirect)_
- Pass `--stacktrace` to gradle for the JNA builds. _(Maximilian Irlinger @atmaxinger)_
- Adapt npm build flags to remove reproducability issues. _(Juri Schreib @Bujuhu)_ _(Nikola Prvulovic @Dynamichost96)_
- Fix creation of shell recorder tests. _(@0x6178656c)_

### 436.11.2 Docker

- Update packagename `libpcrec++-dev` to `libpcrecpp0v5` in Debian Sid. _(Richard Stöckl @Eiskasten)_
- Add shellcheck to Debian containers. _(@0x6178656c)_
- Use `openjdk-17-jdk` in Debian Sid. _(Maximilian Irlinger @atmaxinger)_
- Add Fedora 37 images. _(Mihael Pranjić @mpranj)_
- Update Debian Sid image to use repository Python modules instead of installing with `pip3` due to upstream debian changes. _(Mihael Pranjić @mpranj)_
- Debian Bullsye: use clang 13. _(Mihael Pranjić @mpranj)_
- Update Alpine Linux to 3.17.2. _(Mihael Pranjić @mpranj)_

## 436.12 Gradle

- Use Gradle 7.5.1. _(Mihael Pranjić @mpranj)_
- Update `java-library.gradle` to use `archiveClassifier` _(Maximilian Irlinger @atmaxinger)_

## 436.13 Infrastructure

### 436.13.1 Jenkins

- Add Fedora 37 builds, drop Fedora 35 builds. _(Mihael Pranjić @mpranj)_
- Run more tests also on Master. _(Markus Raab)_
- Move doc to main build stage. _(Markus Raab)_
- Disable parallel test runs. _(Maximilian Irlinger @atmaxinger)_
- Upgrade Jenkins node container to Debian bullseye. _(@0x6178656c)_
- Undo previous change that added automatic `ctest --rerun-failed` to Jenkins CI. _(@kodebach)_

### 436.13.2 Cirrus

- Use Fedora 37. _(Mihael Pranjić @mpranj)_
- Fix `macos_instance` reference, upgrade to macOS Ventura (by default), use Python 3.11 and Ruby 3.x. _(Mihael Pranjić @mpranj)_
- Automatically rerun `testmod_dbus*` tests on macOS. _(@kodebach)_
- Fix dbus not starting on macOS. _(Maximilian Irlinger @atmaxinger)_

### 436.13.3 GitHub Actions

- Add auto-cancellation-running action. _(Tomislav Makar @tmakar)_
- Automatically rerun `testmod_dbus*` tests on macOS. _(@kodebach)_
- Fix dbus not starting on macOS. _(Maximilian Irlinger @atmaxinger)_
- Change stale issue/PR checking to GitHub action. _(@0x6178656c)_
- Update configuration of stale issue/PR action. _(@0x6178656c)_
- Upgrade actions to recent versions and remove deprecated ruby-setup action. _(Mihael Pranjić @mpranj)_

## 436.14 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- Fix broken `/pythongen` link on homepage. _(@stefnotch)_
- Fix redirect logic to not cause loops. _(@stefnotch)_
- Remove duplicated link to `TESTING.md` file. _(@stefnotch)_, _(@janldeboer)_
- Restructure parts of the links on the website. _(@stefnotch)_, _(@janldeboer)_
- Removed broken links to packages for Linux distributions. _(@Dynamichost96)_
- Update npm packages. _(Mihael Pranjić @mpranj)_
- Change URLs to say man-page with a dash. _(@stefnotch)_ _(@janldeboer)_

## 436.15 Outlook

We are currently working on following topics:

- 1.0 API _(Klemens Böswirth @kodebach)_ and _(Stefan Hanreich)_
- Session recording and better Ansible integration _(Maximilian Irlinger @atmaxinger)_
- Change tracking _(Maximilian Irlinger @atmaxinger)_
- Rewriting tools in C _(@hannes99)_
- Elektrify KDE and GNOME _(Mihael Pranjić @mpranj)_
- Elektrify XFCE _(Richard Stöckl @Eiskasten)_
- Mounting SQL databases _(Florian Lindner @flo91)_
- Recording Configuration _(Maximilian Irlinger)_
- Ansible-Elektra _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Configure Olimex Base Images _(Maximilian Irlinger)_
- Improving Build Server Infrastructure _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Improve Java Development Experience _(Michael Tucek)_

## 436.16 Statistics

We closed about 150 issues for this release.
About 28 authors changed 960 files with 29400 insertions(+) and 20927 deletions(-) in 1421 commits.
Thanks to all authors for making this release possible!

## 436.17 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 436.18 Get the Release!

You can download the release from

- here or
- GitHub

The hashsums are:

- name: elektra-0.9.12.tar.gz
- size: 9297913
- md5sum: a6de9401709283b69ec211681f2a7757
- sha1: cb4e282d1346fda771de7510663652555f8e6c7d
- sha256: 38238ba4a5318f999dc3045da06467abf529344dc46ad3fdf42bdca0155e149c

The release tarball is also available signed using GnuPG from

- here or
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found

- here or
- GitHub.

## 436.19 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments, please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 437

# 0.9.13 Release

- guid: e8fd116d-12ab-4281-aaf3-b6441056dd63
- author: Mihael Pranjić
- pubDate: Tue, 14 Mar 2023 10:05:11 +0100
- shortDesc: Bugfix, Ubuntu Packages

We are proud to release Elektra 0.9.13.

## 437.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 437.2 Highlights

- The main purpose of this bugfix release is to fix a regression ( `#4859`) introduced in Elektra 0.9.12.
- We added fresh Ubuntu Jammy Jellyfish (22.04 LTS) and Kinetic Kudu (22.10) packages.
- Please refer to the `Elektra 0.9.12` release notes for a complete list of changes. Due to breaking changes since 0.9.11 we highly recommend to read the upgrade instructions.

## 437.3 Plugins

The following text lists news about the `plugins` we updated in this release.

### 437.3.1 spec

- Add hook placement to spec plugin in README _(Tomislav Makar @tmakar)_

### 437.3.2 gopts

- Add hook placement to gopts plugin in README _(Tomislav Makar @tmakar)_

### 437.3.3 internalnotifications

- Add Maximilian Irlinger as maintainer _(Maximilian Irlinger @atmaxinger)_

#### 437.3.4 logchange

- Add Maximilian Irlinger as maintainer _(Maximilian Irlinger @atmaxinger)_

#### 437.3.5 dbus

- Add Maximilian Irlinger as maintainer _(Maximilian Irlinger @atmaxinger)_

### 437.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

#### 437.4.1 tools

- Check for hook placement on `plugin-check` _(Tomislav Makar @tmakar)_

#### 437.4.2 merge

- Add Maximilian Irlinger as maintainer _(Maximilian Irilnger @atmaxinger)_

### 437.5 Documentation

- .github rework _(Markus Raab)_
- Added `hook` to `placements` contract in `CONTRACT.ini` _(Tomislav Makar @tmakar)_
- Added `hook` information to hooks.md
- Add correct error code in hosts readme _(Tomislav Makar)_

### 437.6 Build

#### 437.6.1 CMake

- Add `infos/maintainer` in plugins. _(Maximilian Irlinger @atmaxinger)_

#### 437.6.2 Docker

- Add Ubuntu Jammy Jellyfish (22.04 LTS) images. _(Mihael Pranjić @mpranj)_
- Add Ubuntu Kinetic Kudu (22.10) images. _(Mihael Pranjić @mpranj)_

### 437.7 Infrastructure

#### 437.7.1 Jenkins

- Add Ubuntu Jammy Jellyfish (22.04 LTS) builds and drop Bionic builds. _(Mihael Pranjić @mpranj)_
- Add Ubuntu Kinetic Kudu (22.10) builds. _(Mihael Pranjić @mpranj)_

### 437.8 Outlook

We are currently working on following topics:

- 1.0 API _(Klemens Böswirth @kodebach)_ and _(Stefan Hanreich)_
- Session recording and better Ansible integration _(Maximilian Irlinger @atmaxinger)_

- Change tracking _(Maximilian Irlinger @atmaxinger)_
- Rewriting tools in C _(@hannes99)_
- Elektrify KDE and GNOME _(Mihael Pranjić @mpranj)_
- Elektrify XFCE _(Richard Stöckl @Eiskasten)_
- Mounting SQL databases _(Florian Lindner @flo91)_
- Recording Configuration _(Maximilian Irlinger)_
- Ansible-Elektra _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Configure Olimex Base Images _(Maximilian Irlinger)_
- Improving Build Server Infrastructure _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Improve Java Development Experience _(Michael Tucek)_

## 437.9 Statistics

We closed 5 issues for this release.
About 7 authors changed 35 files with 1186 insertions(+) and 534 deletions(-) in 48 commits.
Thanks to all authors for making this release possible!

## 437.10 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 437.11 Get the Release!

You can download the release from

- here or
- GitHub

The hashsums are:

- name: elektra-0.9.13.tar.gz
- size: 9297899
- md5sum: 871eaaad39bc834ceb7fb42cb8de66f0
- sha1: 68984021d08500693d692c2cb61eb3409fe75226
- sha256: 9b7512d493c284afcca9875d093081c85c4cfe4926dea193202fbdc5fe89b468

The release tarball is also available signed using GnuPG from

- here or
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found

- here or
- GitHub.

## 437.12 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments, please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 438

# 0.9.14 Release

- guid: 9e1e0790-e7ae-4947-9906-2c176a4f8dd2
- author: Mihael Pranjić
- pubDate: Thu, 16 Mar 2023 14:43:04 +0100
- shortDesc: Bugfix Release

We are proud to release Elektra 0.9.14.

## 438.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 438.2 Highlights

- The main purpose of this bugfix release is to fix regressions ( `#4859`) introduced in Elektra 0.9.12 and 0.9.13.
- Please refer to the `Elektra 0.9.12` release notes for a complete list of changes. Due to breaking changes since 0.9.11 we highly recommend to read the upgrade instructions.

## 438.3 Plugins

The following text lists news about the `plugins` we updated in this release.

### 438.3.1 timeofday

- Use separate symbols for `set` and `commit` functions to satisfy `kdb plugin-check` _(@kodebach)_
- Use new `elektraPluginGetPhase()` instead of counting executions _(@kodebach)_

### 438.3.2 tracer

- Use separate symbols for `set` and `commit` functions to satisfy `kdb plugin-check` _(@kodebach)_

## 438.4 Tests

- Enable more `kdb plugin-check` tests _(@kodebach)_

## 438.5 Build

### 438.5.1 Docker

- Fix conflicting Java versions in CentOS Stream 8 image _(@kodebach)_

## 438.6 Outlook

We are currently working on following topics:

- 1.0 API _(Klemens Böswirth @kodebach)_ and _(Stefan Hanreich)_
- Session recording and better Ansible integration _(Maximilian Irlinger @atmaxinger)_
- Change tracking _(Maximilian Irlinger @atmaxinger)_
- Rewriting tools in C _(@hannes99)_
- Elektrify KDE and GNOME _(Mihael Pranjić @mpranj)_
- Elektrify XFCE _(Richard Stöckl @Eiskasten)_
- Mounting SQL databases _(Florian Lindner @flo91)_
- Recording Configuration _(Maximilian Irlinger)_
- Ansible-Elektra _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Configure Olimex Base Images _(Maximilian Irlinger)_
- Improving Build Server Infrastructure _(Lukas Hartl)_ and _(Maximilian Irlinger)_
- Improve Java Development Experience _(Michael Tucek)_

## 438.7 Statistics

We closed `2 issues` for this release.
About 4 authors changed 22 files with 299 insertions(+) and 139 deletions(-) in 14 commits.
Thanks to all authors for making this release possible!

## 438.8 Join the Initiative!

We welcome new contributors! Read `here` about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our `issue tracker`.

## 438.9 Get the Release!

You can download the release from

- `here` or
- `GitHub`

The `hashsums are:`

- name: elektra-0.9.14.tar.gz

- size: 9299478
- md5sum: eb0f1d2e5d93bbae122999b5a27be343
- sha1: 8d7a44ae6b4d53c52ab4219f955d6010f87682c8
- sha256: e4632bb6baa78f6a68c312469e41fd1ef07406571749e32f2645b1858d01a58d

The release tarball is also available signed using GnuPG from

- here or
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found

- here or
- GitHub.

## 438.10 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments, please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 439

# 0.10.0 Release

- guid: e0a2b6f4-b18d-47d8-b642-e27907255666
- author: Tomislav Makar
- pubDate: Tue, 23 May 2023 13:33:52 +0200
- shortDesc: New Changetracking API, New spec plugin, Go into repo

We are proud to release Elektra 0.10.0.

## 439.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 439.2 Highlights

- New Changetracking API
- New spec plugin
- Go into repo

### 439.2.1 New Changetracking API

We've created a new KeySet diffing and changetracking API for both internal and external use. Several plugins already had their own implementations to detect changes made to the key database. Every implementation did something different and results varied. With this new release every plugin now uses the same shared implementation, with identical results!
To try the amazing new API yourself, take a look at the tutorial.
Elektra's internal code now also uses this new API to detect which backends to execute. This can lead to better performance in I/O bound cases, where previously certain keys would have been detected as changed when they weren't.
We ran some memory benchmarks and found a slightly increased memory usage on a stock instance of Elektra. If you're using many plugins that do changetracking, the overhead will decrease.

| Number of Keys | Old Implementation (bytes) | New Implementation (bytes) | Memory Increase (%) |
|---:|---:|---:|---:|
| 50 | 225792 bytes | 229580 bytes | 1,68 % |
| 500 | 383180 bytes | 411238 bytes | 7,32 % |

| Number of Keys | Old Implementation (bytes) | New Implementation (bytes) | Memory Increase (%) |
|---|---|---|---|
| 5000 | 1992294 bytes | 2306867 bytes | 15,79 % |
| 50000 | 18245222 bytes | 21181235 bytes | 16,09 % |
| 500000 | 178782208 bytes | 207827763 bytes | 16,25 % |

Apart from memory benchmark, we also ran some performance benchmarks. As the benchmark is heavily I/O bound, the biggest bottleneck is the I/O performance of the system. We could not reliably detect a real, reliably reproducible performance impact measured in seconds. Alternatively, we have measured executed instructions. There seems to be about 10 % overhead, but we don't expect it to be noticeable in real-world workloads.

| Number of Keys | Old Implementation (Instructions) | New Implementation (Instructions) | Performance Overhead (%) |
|---|---|---|---|
| 50 | 18910449 | 19583227 | 3,56 % |
| 500 | 63001911 | 68948096 | 9,44 % |
| 5000 | 526801917 | 586344210 | 11,30 % |
| 50000 | 5730261920 | 6340292587 | 10,65 % |
| 500000 | 104614374974 | 110702166761 | 5,82 % |

### 439.2.2 New spec plugin

The spec plugin was rewritten to use the standardized error handling in Elektra. It is now strictly defined that the `spec` plugin throws a warning on `kdbGet` and on any other call an error.
Default values are now created in the `default` namespace. The instantiated array specifications are now also created in the `default` namespace.
Keys with a require metakey and no default metakey do throw an error now.
Known limitations:

- # and _ keys do not work on MINGW
- No defaults for _ globbing character

For more information see Spec Plugin.

### 439.2.3 Go into repo

The go binding is now inside of `libelektra` repository. It is now also included into our build pipeline. For more information on how to use it see go readme.

> NOTE: On information how to publish new versions of `go-elektra` to go packages see `go-publishing`.

## 439.3 Plugins

The following text lists news about the `plugins` we updated in this release.

### 439.3.1 General

- Updated target name of shared object files according to `#3486`

### 439.3.2 spec

- Rewrite spec plugin, fix bugs and use correct error handling _(Tomislav Makar @tmakar)_
- Remove info metakey from spec plugin _(Tomislav Makar @tmakar)_

### 439.3.3 counter

- Move `static` variables into functions to avoid global variables _(@kodebach)_

### 439.3.4 logchange

- Utilize new changetracking API _(Maximilian Irlinger @atmaxinger)_
- Add Maximilian Irlinger as maintainer _(Maximilian Irlinger @atmaxinger)_

### 439.3.5 yajl

- The `itKs` global variable workaround, which was used to replace the now removed internal `KeySet` cursor, was replaced with a custom context struct. _(@kodebach)_

### 439.3.6 toml

- The `flex` lexer and `bison` parser are now fully reentrant and therefore thread-safe. _(@kodebach)_

### 439.3.7 multifile

- Remove multifile plugin as it is unmaintained and conflicts with the new backend architecture _(Maximilian Irlinger @atmaxinger)_

### 439.3.8 c

- Add Florian Lindner as maintainer _(Florian Lindner @flo91)_
- Use separate symbols for `set` and `commit` functions to satisfy `kdb plugin-check` _(@kodebach)_
- Use new `elektraPluginGetPhase()` instead of counting executions _(@kodebach)_

### 439.3.9 mmapstorage

**Note**: The plugin is currently disabled, because it is not yet compatible with the COW data structures.

TODO: remove above note, when COW support is added.

- The magic data structures are now fully compile-time constants. The magic number to detect endianness is generated in CMake instead of at runtime. _(@kodebach)_

### 439.3.10 syslog

- Convert to hook plugin _(Maximilian Irlinger @atmaxinger)_
- Utilize new changetracking API _(Maximilian Irlinger @atmaxinger)_

### 439.3.11 lineendings

- Add Florian Lindner as maintainer _(Florian Lindner @flo91)_

### 439.3.12 length

- Add Florian Lindner as maintainer _(Florian Lindner @flo91)_

### 439.3.13 missing

- Add Florian Lindner as maintainer _(Florian Lindner @flo91)_

### 439.3.14 unit

- Add Florian Lindner as maintainer _(Florian Lindner @flo91)_

#### 439.3.15 dbus

- Utilize new changetracking API _(Maximilian Irlinger @atmaxinger)_
- Add Maximilian Irlinger as maintainer _(Maximilian Irlinger @atmaxinger)_

#### 439.3.16 internalnotifications

- Utilize new changetracking API _(Maximilian Irlinger @atmaxinger)_
- Add Maximilian Irlinger as maintainer _(Maximilian Irlinger @atmaxinger)_

### 439.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

#### 439.4.1 Compatibility

#### 439.4.2 Core

- The `syslog` logging code now calls `openlog` before every `syslog` to avoid the use of a global variable. _(@kodebach)_
- Fix memleak in kdb.c, #4925 _(@hannes99)_

#### 439.4.3 loader

- Adapt target rename with `-plugin-` in `dl.c` _(Tomislav Makar @tmakar)_

#### 439.4.4 go

- Golang is now included to build `go binding` _(Tomislav Makar @tmakar)_
- From now on we use tags which are prefixed with `v` so go package versioning works _(Tomislav Makar @tmakar)_

#### 439.4.5 kdb

- Add new changetracking API _(Maximilian Irlinger @atmaxinger)_
- Fix unwanted removal of subkeys when using mv _(Hannes Laimer @hannes99)_
- Fix inconsistent return values in code, tests and man pages _(Hannes Laimer @hannes99)_
- Remove `smount` alias _(Hannes Laimer @hannes99)_

### 439.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

#### 439.5.1 jna

- Updated Java binding related dependencies. _(Michael Tucek @tucek)_
- Updated `KDBException` to only access error key at construction time. _(Michael Tucek @tucek)_
- Removed public naive resource release API. To migrate just remove calls to the affected methods `Key#release()`, `KeySet#release()` and `KDBException#releaseErrorKey()` _(Michael Tucek @tucek)_

- Enabled strict javadoc checking for Gradle build _(Michael Tucek @tucek)_
- add merging based on elektraMerge _(Maximilian Irlinger @atmaxinger)_
- Added support for `ksIncRef` for `KeySet` _(Michael Tucek @tucek)_
- Enabled `ReferenceCleaner` _(Michael Tucek @tucek)_

### 439.5.2 go-elektra

- Move `go-elektra` binding from repository into bindings folder of `libelektra` _(Tomislav Makar @tmakar)_
- Change module for `go-elektra` _(Tomislav Makar @tmakar)_
- Rename `go-elektra` module _(Tomislav Makar @tmakar)_
- Adapt README for `go-elektra` and change module name to match `github.com` _(Tomislav Makar @tmakar)_

## 439.6 Tools

### 439.6.1 elektrad

- Implemented new request to add multiple metakeys for one key _(Tomislav Makar @tmakar)_
- Adding bulk creation request for configuration keys _(Tomislav Makar @tmakar)_

### 439.6.2 webd

- Implemented new request to add multiple metakeys for one key _(Tomislav Makar @tmakar)_
- Adding bulk creation request for configuration keys _(Tomislav Makar @tmakar)_

## 439.7 Scripts

### 439.7.1 Jenkins

- Use only v prefix for tags not in real version _(Tomislav Makar @tmakar)_

### 439.7.2 Release

- Exclude v prefix from `populate-release-notes.sh` script _(Tomislav Makar @tmakar)_
- Handle tag prefix `v` correctly in `release.sh` _(Tomislav Makar @tmakar)_
- Check for `v` prefix in `make-source-package` script and `Jenkinsfile.release` _(Tomislav Makar @tmakar)_
- Use correct tags for `git-release-stats` script _(Tomislav Makar @tmakar)_

## 439.8 Documentation

- Adapt and remove outdated docs https://issues.libelektra.org/4882 _(Tomislav Makar @tmakar)_
- Added missing dependencies in COMPILE.md for APT-based systems _(Michael Tucek @tucek)_
- Add `default` namespace to namespaces documentation _(Tomislav Makar @tmakar)_
- Added Tomislav Makar to `AUTHORS.md` _(Tomislav Makar @tmakar)_

- Added Florian Lindner to `AUTHORS.md` _(Florian Lindner @flo91)_
- .github rework _(Markus Raab)_
- Added `hook` to `placements` contract in `CONTRACT.ini` _(Tomislav Makar @tmakar)_
- Added `hook` information to hooks.md
- Added Hannes Laimer to `AUTHORS.md` _(Hannes Laimer @hannes99)_
- Add README to tools/kdb _(Hannes Laimer @hannes99)_
- Fixed shell-recorder test in `install-webui.md` _(Tomislav Makar @tmakar)_
- Add new shell-recorder test in `install-webui.md` for newly implemented request for adding multiple metakeys for one key _(Tomislav Makar @tmakar)_
- Small fixes in decisions. _(Markus Raab)_
- Remove wikipedia from release todos in RELEASE.md _(Tomislav Makar @tmakar)_
- Update AUTHORS.md info _(@kodebach)_
- Add Stefan Hanreich to AUTHORS.md _(Stefan Hanreich @lawli3t)_

### 439.8.1 Use Cases

- Add specification use case for array-specification _(Tomislav Makar @tmakar)_
- Add specification use case for underline-specification _(Tomislav Makar @tmakar)_
- Add specification use case for simple-specification _(Tomislav Makar @tmakar)_
- Add specification use case for enum-specification _(Tomislav Makar @tmakar)_
- Add complete specification for `dockerd` configuration file (`daemon.json`) _(Tomislav Makar @tmakar)_

### 439.8.2 Decisions

- Decide and implement decision process _(Markus Raab)_
- Decided future library split _(@kodebach)_
- Decided `decision process` _(Markus Raab)_
- Draft for man pages _(Markus Raab)_
- Add decision for change tracking _(Maximilian Irlinger @atmaxinger)_
- Create decision for allowed and prohibited operation seqences _(Maximilian Irlinger @atmaxinger)_
- Add decisions about location of headers and use of `#include` in the repo _(@kodebach)_
- Add decision about metadata semantics _(@kodebach)_
- Many small fixes to adapt to documentation guidelines and new decision process. _(Markus Raab)_
- Add decision for read-only keynames _(Maximilian Irlinger @atmaxinger)_
- Revive keyname decision _(@kodebach)_
- Add decision for copy-on-write and provide implementation suggestions. _(Maximilian Irlinger @atmaxinger)_
- Added explanation on why we wanted to migrate from Maven to Gradle for Java-related build facilities. _(Michael Tucek @tucek)_

### 439.8.3 Tutorials

- Add basic tutorial about changetracking _(Maximilian Irlinger @atmaxinger)_

### 439.8.4 Man Pages

## 439.9 Tests

- Add shell test to verify correct behavior for https://github.com/ElektraInitiative/libelektra/issues/
_(Tomislav Makar @tmakar)_
- Add link to `linkchecker.whitelist` _(Tomislav Makar @tmakar)_

## 439.10 Build

### 439.10.1 Docker

- CentOS 8 Stream: manually install `config-manager` DNF plugin. _(Maximilian Irlinger @atmaxinger)_

## 439.11 Infrastructure

### 439.11.1 Cirrus

- Rename deprecated `d-bus` to `dbus` in `macOS.yml` and `.cirrus.yml` Issue-#4900 _(Tomislav Makar @tmakar)_
- Push FreeBSD 12.3 to 12.4 since 12.3 is end of life. _(Richard Stöckl @eiskasten)_

### 439.11.2 GitHub Actions

## 439.12 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

## 439.13 Miscellaneous

- Many global variables that where used as constants have been made fully `const` _(@kodebach)_

## 439.14 Outlook

We are currently working on following topics:

- Rewriting tools in C _(@hannes99)_
- Elektrify KDE and GNOME _(Mihael Pranjić @mpranj)_
- Elektrify XFCE _(Richard Stöckl @Eiskasten)_
- Mounting SQL databases _(Florian Lindner @flo91)_
- Recording Configuration _(Maximilian Irlinger)_
- Ansible-Elektra _(Maximilian Irlinger)_
- Configure Olimex Base Images _(Maximilian Irlinger)_
- Improving Build Server Infrastructure _(Lukas Hartl)_
- Improve Java Development Experience _(Michael Tucek)_

## 439.15 Statistics

About 9 authors changed 309 files with 10024 insertions(+) and 4025 deletions(-) in 402 commits.
Thanks to all authors for making this release possible!

## 439.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 439.17 Get the Release!

You can download the release from

- here or
- GitHub

The hashsums are:

- name: elektra-0.10.0.tar.gz
- size: 9230228
- md5sum: 39ac04c8a0b07061bea781fd73cb13aa
- sha1: 846dcfa22410403b575a4d4d87970aa4841aafdd
- sha256: cb56e40b37c42e1e235cf9e76c9250024f912cbb61cca1a2888f9305f5228dcd

The release tarball is also available signed using GnuPG from

- here or
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found

- here or
- GitHub.

## 439.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments, please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 440

# 0.11.0 Release

- guid: 518d1bdd-a8ea-48da-ab7c-6621a8670c66
- author: Maximilian Irlinger
- pubDate: Tue, 05 Sep 2023 17:10:40 +0200
- shortDesc: Session Recording, ODBC Backend

We are proud to release Elektra 0.11.0.

## 440.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:

```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 440.2 Highlights

- Session Recording
- ODBC Backend

### 440.2.1 Session Recording

Elektra now comes with a powerful new feature that allows users to record and export changes made to the KDB: session recording. This feature enables you to easily track changes made to the configuration database over time, which helps troubleshoot issues, diagnose errors, and improve system performance. It even lets you undo the changes you've performed!
You can also export the changes as Ansible playbooks using the new Ansible storage plugin! This makes it easy to automate and reproduce system configurations. We think that this feature offers significant time savings and improved accuracy when managing complex systems.
Whether you're a system administrator, developer, or DevOps engineer, we believe that the session recording feature in Elektra will become an essential tool for managing and maintaining system configurations. Try it today and experience the benefits of streamlined configuration management.
**Note:** when you activate session recording, concurrency of Elektra will be somewhat limited. As long as it is active, a global lock will be created to ensure no two processes will write data simultaneously. This behavior is similar as to when multiple processes will write to the same configuration file. Applications should already handle this case gracefully, and just retry writing their configuration.

### 440.2.2 ODBC Backend

Based on the new and more versatile concept for backends, where backends are implemented as plugins, a new backend-plugin that uses ODBC data sources for storing keys has been developed. It was tested on Gentoo Linux with `unixODBC` using `SQLite` and `PostgreSQL` data sources. The ODBC backend-plugin can only be built if the ODBC library is available on the build system. This can be accomplished by installing e.g. unixODBC. Microsoft ODBC (on MS Windows) and `iODBC` should also be supported, but were not tested yet. If you use the plugin with another ODBC implementation as unixODBC, you are very welcome to update the documentation with your experiences!
The tutorial is a good place for getting started with the new ODBC backend for Elektra.

## 440.3 Plugins

The following text lists news about the `plugins` we updated in this release.

### 440.3.1 General

- Updated target name of shared object files according to `#3486`

### 440.3.2 spec

- Remove metakeys from array elements correctly `#4961` _(Tomislav Makar @tmakar)_

### 440.3.3 recorder

- Add recorder plugin. _(Maximilian Irlinger @atmaxinger)_

### 440.3.4 jdbc

- Fix formatting/spelling problem in README.md _(Hannes Laimer @hannes99)_

### 440.3.5 backend_odbc

- Add ODBC backend _(Florian Lindner @flo91)_

### 440.3.6 ansible

- Add `ansible` plugin for exporting keysets as `ansible-libelektra` playbooks. _(Maximilian Irlinger @atmaxinger)_

### 440.3.7 toml

- Fix error reporting when unsupported metakey has been encountered. _(Maximilian Irlinger @atmaxinger)_

### 440.3.8 length

- Remove experimental status. _(Maximilian Irlinger @atmaxinger)_

### 440.3.9 Xfconf

- Implemented the first revision of the Xfconf binding. _(Richard Stöckl @Eiskasten)_
- Make Xfconf storage plugin compatible with new backend and noresolver. _(Richard Stöckl @Eiskasten)_
- This allows to use elektra as a drop-in replacement for applications which use Xfconf. _(Richard Stöckl @Eiskasten)_
- Xfconf applications can now read and write configuration settings to elektra. _(Richard Stöckl @Eiskasten)_
- Use cmake variables in the replace and restore scripts. _(Richard Stöckl @Eiskasten)_

### 440.3.10 c

- Improve the c plugin: some refactoring, add documentation, extend README.md, add unit tests. _(Florian Lindner @flo91)_

## 440.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 440.4.1 kdb

- Add `elektraCopyError` function to copy error from one key to another _(Maximilian Irlinger @atmaxinger)_
- Add `elektraCopyWarnings` function to copy warnings from one key to another _(Maximilian Irlinger @atmaxinger)_
- Add `elektraCopyErrorAndWarnings` function to copy error and warnings from one key to another _(Maximilian Irlinger @atmaxinger)_

### 440.4.2 record

- Add record library used for session recording. _(Maximilian Irlinger @atmaxinger)_

### 440.4.3 ease

- Add `elektraArrayGetPrefix` function. _(Maximilian Irlinger @atmaxinger)_

## 440.5 Bindings

Bindings allow you to utilize Elektra using `various programming languages`. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 440.5.1 C++

- Provide getter for the underlying C object of KDB _(Maximilian Irlinger @atmaxinger)_
- Add `ElektraDiff` binding for C++ _(Maximilian Irlinger @atmaxinger)_
- The `dup` method of `KeySet` now returns a wrapped object _(Maximilian Irlinger @atmaxinger)_
- Add an overload for `KeySet::cut` that accepts a string for the keyname _(Maximilian Irlinger)_
- The `dup` method of `Key` now returns a wrapped object _(Maximilian Irlinger @atmaxinger)_
- Add overloads for `Key::isBelow`, `Key::isBelowOrSame` and `Key::isDirectBelow` that accept a string as the key name _(Maximilian Irlinger @atmaxinger)_
- Include the header `cstdint` in `key.hpp`. It is needed for an enum of type `std::uint8_t` _(Florian Lindner @flo91)_

### 440.5.2 Java

- Added documentation about the design of the binding. _(Michael Tucek @tucek)_

### 440.5.3 Python

- Add `ElektraDiff` binding _(Maximilian Irlinger @atmaxinger)_
- The `__meta__` attribute on a key now returns a proper keyset _(Maximilian Irlinger @atmaxinger)_
- Add new module `kdb.errors` to simplify extracting errors and warnings from keys _(Maximilian Irlinger @atmaxinger)_
- Add new module `kdb.record` for interfacing with the session recording capabilities of Elektra _(Maximilian Irlinger @atmaxinger)_
- Add `getConflictingKeys` method to `kdb.merge.MergeResult`. _(Maximilian Irlinger @atmaxinger)_

### 440.5.4 Rust

- Upgrade bindgen dependency to version 0.55.1. _(Maximilian Irlinger @atmaxinger)_

### 440.5.5 Xfconf

- Implemented the first revision of the Xfconf binding. _(Richard Stöckl @Eiskasten)_
- This allows to use elektra as a drop-in replacement for applications which use Xfconf. _(Richard Stöckl @Eiskasten)_
- Xfconf applications can now read and write configuration settings to elektra. _(Richard Stöckl @Eiskasten)_
- Use cmake variables in the replace and restore scripts. _(Richard Stöckl @Eiskasten)_

## 440.6 Tools

### 440.6.1 kdb

- Add commands for session recording. _(Maximilian Irlinger @atmaxinger)_
- The `kdb mount` command will now automatically detect whether the given path is an absolute path. _(Maximilian Irlinger @atmaxinger)_
- Fix unwanted removal of subkeys when using mv _(Hannes Laimer @hannes99)_
- Fix inconsistent return values in code, tests and man pages _(Hannes Laimer @hannes99)_
- Remove `smount` alias _(Hannes Laimer @hannes99)_

## 440.7 Documentation

- Fix bug in Doxygen comment for `const char * keyName (const Key * key)` which lead to failed building of the refman.pdf on recent TeX Live releases _(Florian Lindner @flo91)_
- Update release documentation regarding version tags. _(Mihael Pranjić @mpranj)_
- Add glossaries for developers and contributors _(Maximilian Irlinger @atmaxinger)_
- Add use cases for using the bindings. _(Richard Stöckl @Eiskasten)_
- Write the Xfconf tutorial. _(Richard Stöckl @Eiskasten)_
- Add use cases for libelektra-core _(@lawli3t)_

### 440.7.1 Use Cases

- Add end-user and developer integration use case _(Hannes Laimer @hannes99)_
- Add use cases for using the bindings. _(Richard Stöckl @Eiskasten)_
- Add use cases for libelektra-core _(@lawli3t)_

### 440.7.2 Decisions

- Add decision for Elixir bindings. _(@0x6178656c)_

### 440.7.3 Tutorials

- Add tutorial for session recording. _(Maximilian Irlinger @atmaxinger)_
- Add tutorial for ODBC backend. _(Florian Lindner @flo91)_

### 440.7.4 Man Pages

- Add man pages for session recording

## 440.8 Tests

- Add macro `succeed_if_keyset_contains_key_with_string` to assert that a certain key with a certain value must exist. _(Maximilian Irlinger @atmaxinger)_

### 440.8.1 Docker

- Use openwrt/sdk instead of openwrtorg/sdk. _(Richard Stöckl @Eiskasten)_
- Reduce fedora docker image size (PR #4637) _(@4ydan)_

## 440.9 Infrastructure

### 440.9.1 Jenkins

- Disable openwrt build until we have found a new maintainer for it. _(Maximilian Irlinger @atmaxinger)_

### 440.9.2 Cirrus

- The arch package `texlive-most` is no longer available, replaced it with other texlive packages. See `https://archlinux.org/packages/?q=texlive` _(Florian Lindner @flo91)_

## 440.10 Outlook

We are currently working on following topics:

- Fix bugs!
- Improvements for `https://opensesame.libelektra.org/`

## 440.11 Statistics

About 16 authors changed 289 files with 20238 insertions(+) and 2629 deletions(-) in 488 commits.
Thanks to all authors for making this release possible!

## 440.12 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 440.13 Get the Release!

You can download the release from

- here or
- GitHub

The hashsums are:

- name: elektra-0.11.0.tar.gz
- size: 9535785
- md5sum: 47e52f34507fefd3e05a399383be7353
- sha1: bc4b1393540c2f0bfbf43855c59204ba7d00d224
- sha256: 4e1f7c986010555a1d30ef2d23c0636373e993bab88e5ec238cac18a469b5cc2

The release tarball is also available signed using GnuPG from

- here or
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found

- here or
- GitHub.

## 440.14 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments, please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 441

# <<VERSION>> Release

This release did not happen yet.
Please always update this file within **every PR**:

1. write what changed
2. use links pointing to your change (See Documentation Guidelines)
3. add your name at the end of the line **Syntax:** _(your name)_

For example, Max would write:
```
- Added a new [doc plugin](https://www.libelektra.org/plugins/doc) _(Max)_
```
Pick a random line to write your changes to minimize the chances of conflicts in this file.
For non-trivial changes, you can choose to be part of the highlighted changes. Please write a highlight section in this case.
After the horizontal line the release notes for the next version starts.

---

```
<<scripts/generate-news-entry>>
```
We are proud to release Elektra <<VERSION>>.

## 441.1 What is Elektra?

Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database. For more information, visit `https://libelektra.org`.
You can also read the news `on our website`.
You can try out the latest Elektra release using our docker image `elektra/elektra`. This is the quickest way to get started with Elektra without compiling and other obstacles, simply run:
```
docker pull elektra/elektra
docker run -it elektra/elektra
```

## 441.2 Highlights

- <<HIGHLIGHT>>
- <<HIGHLIGHT>>
- <<HIGHLIGHT>>

### 441.2.1 <<HIGHLIGHT>>

### 441.2.2 <<HIGHLIGHT>>

### 441.2.3 <<HIGHLIGHT>>

## 441.3 Plugins

The following text lists news about the `plugins` we updated in this release.

### 441.3.1 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.3.2 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.3.3 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.3.4 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.3.5 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.3.6 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.3.7 <<Plugin>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.4 Libraries

The text below summarizes updates to the `C (and C++)-based libraries` of Elektra.

### 441.4.1 Compatibility

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.4.2 Core

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.4.3 <<Library>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.4.4 <<Library>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.4.5 <<Library>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.4.6 <<Library>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.4.7 <<Library>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.5 Bindings

Bindings allow you to utilize Elektra using various programming languages. This section keeps you up-to-date with the multi-language support provided by Elektra.

### 441.5.1 <<Binding>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.5.2 <<Binding>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.5.3 <<Binding>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.5.4 <<Binding>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.5.5 <<Binding>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.6 Tools

### 441.6.1 <<Tool>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.6.2 <<Tool>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.6.3 <<Tool>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.6.4 <<Tool>>

- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.7 Scripts

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.8 Documentation

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.8.1 Use Cases

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.8.2 Decisions

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.8.3 Tutorials

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.8.4 Man Pages

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.9 Tests

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.9.1 C

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.9.2 Shell Recorder

- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.9.3 C++

- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.10 Packaging

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.11 Build

### 441.11.1 CMake

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.11.2 Docker

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.12 Infrastructure

### 441.12.1 Jenkins

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.12.2 Cirrus

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

### 441.12.3 GitHub Actions

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.13 Website

The website is generated from the repository, so all information about plugins, bindings and tools are always up-to-date. Furthermore, we changed:

- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.14 Outlook

We are currently working on following topics:

- <<TODO>>
- <<TODO>>
- <<TODO>>

## 441.15 Statistics

We closed <> issues for this release.
<<scripts/git-release-stats <<VERSION>>.VER-1 <<VERSION>>>>>
Thanks to all authors for making this release possible!

## 441.16 Join the Initiative!

We welcome new contributors! Read here about how to get started.
As first step, you could give us feedback about these release notes. Contact us via our issue tracker.

## 441.17 Get the Release!

You can download the release from

- here or
- GitHub

The hashsums are:
<<scripts/generate-hashsums elektra-<<VERSION>>.tar.gz>>
The release tarball is also available signed using GnuPG from

- here or
- GitHub

The following GPG Key was used to sign this release: 12CC44541E1B8AD9B66AFAD55262E7353324914A
Already built API documentation can be found

- here or
- GitHub.

## 441.18 Stay tuned!

Subscribe to the RSS feed to always get the release notifications.
If you also want to participate, or for any questions and comments, please contact us via our issue tracker on GitHub.
Permalink to this NEWS entry
For more information, see https://libelektra.org.
Best regards, Elektra Initiative

# Chapter 442

# Citations

Please cite Elektra the following way:

```
@article{raab2016elektra,
        doi = {10.21105/joss.00044},
        url = {http://dx.doi.org/10.21105/joss.00044},
        year  = {2016},
        month = {dec},
        publisher = {The Open Journal},
        volume = {1},
        number = {8},
        author = {Markus Raab},
        title = {Elektra: universal framework to access configuration parameters},
        journal = {The Journal of Open Source Software}
}
```

For more information, see:

http://joss.theoj.org/papers/10.21105/joss.00044

# Chapter 443

# Documentation Index

This folder contains documentation for “Elektra – the configuration framework for everyone”. If you do not know what Elektra is, we recommend that you check out our `homepage` first. This ReadMe deals with the content of the documentation folder and should give you a hint where to look for specific information.

## 443.1 Introductory

- Goals: We specify the goals and target audiences for Elektra in this document.
- Why: This document describes why you should use Elektra.
- Vision: This document describes the vision behind Elektra.
- Big Picture: This document provides an birds eye view of Elektra and the key database (KDB).
- Security: This guideline shows how Elektra handles security concerns.
- Tutorials: The tutorials folder provides various **user related tutorials**. If you are interested in **developer related tutorials** instead, then please take a look at the folder dev.
- News: The news folder contains release notes and other recent information about Elektra.
- Paper: This directory contains a research paper about Elektra, also available in `PDF` format.

## 443.2 Using Elektra

- Installation: These instructions tell you how you can install Elektra in your favorite operating system.
- Compile: If you want to compile Elektra from source, please look at this document.
- Debugging: If you want to debug Elektra, please look at this document.
- Help: This folder contains our man pages in Markdown format. The folder man contains these man pages in roff format, which you can read using the Unix utility `man` if you already installed Elektra.
- Keynames: This document describes how Elektra's keynames work.

### 443.2.1 API

- API: This overview of the application programming interface tells you how you can develop an application that uses Elektra.
- Design: This document describes the design of Elektra’s C API.

## 443.3 Advanced Information

- `Metadata`: This document specifies data about the KDB (meta information), like supported data types and configuration options.
- `Contract`: The plugin contract specifies keys and values that an Elektra plugin provides.

## 443.4 Elektra Internals

- Copy-on-write: This document gives a technical overview over our copy-on-write implementation for `Key` and `KeySet`.

## 443.5 Contributing

- Coding: The coding guidelines describe the basic rules you should keep in mind when you want to contribute code to Elektra.
- Git: This document describes how we use the version control system `Git` to develop Elektra.
- Ideas: If you want to contribute to Elektra and do not know what, you can either take a look here or at our `issue tracker`.
- ToDo: This folder contains various ToDo items for future releases of Elektra.
- Authors: This file lists information about Elektra's authors.

## 443.6 Other

- Images: The images folder contains logos and other promotional material.
- Decisions: If you are interested in why Elektra uses a certain technology or strategy, then please check out the documents in this folder.
- Markdown Link Converter: This tool converts links in Markdown files to make them usable in our `Doxygen documentation`.
- Usecases: This folder contains use cases for our `snippet sharing service` and the upcoming web user interface for the KDB.
- Glossary: The glossary explains common terminology used in the documentation.

# Chapter 444

# Security

Security is a very important point in libraries. In most use cases there is nearly no point of danger in using Elektra. But some a very security related, especially when you use a daemon or some kind of distributed configuration.

## 444.1 Access Permissions

We only use access permissions from the kernel, we do not add an additional layer (or daemon). So configuration file access is as secure as with direct access to configuration files.

## 444.2 Namespaces

Elektra by default guarantees that configuration from specific namespaces come from respective paths in the file system:

- `dir`-namespace: from current working directory
- `user`-namespace: from users home directory
- `system` or `spec`-namespace: no restrictions

## 444.3 Environment Variables

Environment variables are usually avoided, but instead Elektra itself is used to configure Elektra. The core is not allowed to use any environment variables.
For some plugins, however, Environment variables are used for better integration in systems. This might be a security risk.

## 444.4 Compiler Options

Can be changed using standard CMake ways. Some hints:
`https://wiki.debian.org/Hardening`

## 444.5 Memory Leaks

We use Valgrind (`--tool=memcheck`) and ASAN (clang+gcc) to make sure that Elektra does not suffer memory leaks and incorrect memory handling.

# Chapter 445

# Testing

## 445.1 Introduction

Libraries need pervasive testing for continuous improvement. Any problem found and behavior described must be written down as test so that it is assured that no new regressions will be added.

## 445.2 Running Tests

Running all tests in the build directory:
```
make run_all
```
And on the target (installed) system:
```
kdb run_all
```
To run `memcheck` tests run in the build directory:
```
make run_memcheck
```
They are supplementary, ideally you run all three.
Some tests write into system paths and into the home directory. This implies that the UID running the tests must have a home directory. To avoid running tests that write to the disk you can use:
```
make run_nokdbtests
```
You can also directly run ctest to make use of parallel testing:
```
ctest -T Test --output-on-failure -j 6
ctest -T MemCheck -LE memleak --output-on-failure -j 6
```
The alternative to `make run_nokdbtests`:
```
ctest -T Test --output-on-failure -LE kdbtests -j 6
```
To only run tests whose names match a regular expression, you can use:
```
ctest -V -R <regex>
```

## 445.3 Required Environment

To run the tests successfully, the environment must fulfill:

- Mounted /dev and /proc (to have stdin and stdout for import & export test cases).
- POSIX tools need to be available (including the `file` tool)
- User must be able to write to system and spec (see below)

If the access is denied, several tests will fail. You have some options to avoid running them as root:

1. To avoid running the problematic test cases (reduces the test coverage!) run `ctest` without tests that have the label `kdbtests`: `ctest --output-on-failure -LE kdbtests` (which is also what `make run_nokdbtests` does)
2. To give your user the permissions to the relevant paths execute the lines below once as root.

   **Warning: Changing permissions on the wrong paths can be harmful! Please make sure that the paths are correct.** In doubt make sure that you have a backup of the affected directories.

   First load the required information and verify the paths:

```
sudo kdb mount-info
echo `kdb sget system:/info/elektra/constants/cmake/CMAKE_INSTALL_PREFIX .`/`kdb sget
      system:/info/elektra/constants/cmake/KDB_DB_SPEC .`
echo `kdb sget system:/info/elektra/constants/cmake/KDB_DB_SYSTEM .`
```

Then change the permissions:

```
sudo chown -R `whoami` `kdb sget system:/info/elektra/constants/cmake/CMAKE_INSTALL_PREFIX .`/`kdb sget
      system:/info/elektra/constants/cmake/KDB_DB_SPEC .`
sudo chown -R `whoami` `kdb sget system:/info/elektra/constants/cmake/KDB_DB_SYSTEM .`
```

After that all test cases should run successfully as described above.

1. Compile Elektra so that system paths are not actual system paths, e.g. to write everything into the home directory (~) use CMake options: `-DKDB_DB_SYSTEM="~/.config/kdb/system" -DKDB_DB_SPEC="~/.config/kdb/spec"` (for an example of a full CMake invocation see `scripts/configure-home`)
2. Use the XDG resolver (see `scripts/configure-xdg`) and set the environment variable `XDG_↩ CONFIG_DIRS`, currently lacks `spec` namespaces, see #734.

## 445.4 Manual Testing

Running executables in the build directory needs some preparation. Here we assume that `build` is the build directory and it is the top-level of Elektra's source code:

```
cd build
. ../scripts/dev/run_env
```

After sourcing `run_env`, you can directly execute `kdb` and other binaries built with Elektra (such as the examples). Pay attention that sourcing depends on the operating system or rather the shell. For example on standard FreeBSD 11.3 you have to execute `sh` in the root of the repository first. Then do *not* use the `source` command but the point `.` as explained above.

## 445.5 Using debuggers

You can use debuggers such as `gdb` to develop Elektra. This can be interesting if you write test cases and they fail at some unknown point. Many tests are put into the `/bin` folder in your build directory. As a consequence, you can `cd` into `/bin` and call, for example

```
gdb hello
```

If you use Docker to run your tests it makes sense to call the debugger in the same container. In this case you might be required to pass additional parameters to Docker. An example for this is passing `--security-opt seccomp=unconfined` to disable address space randomization.

## 445.6 Recommended Environment

The tests are designed to disable themselves if some necessary tools are missing or other environmental constraints are not met. To really run *all* tests (also those that are mostly designed for internal development) you need to fulfil:

- Elektra must be installed (for gen + external test cases).
- A running dbus daemon (Either "system" or "session" daemon).
- `gpg2` or `gpg` binary must be available.

Above environment is needed for both `kdb run_all` (installed test cases) and `make run_all` (test cases executed from the build directory). For `make run_all` following development tools enable even more tests:

- The script `checkbashisms` is needed to check for bashism (`tests/shell/check_bashisms.sh`), it is part of `devscripts`.
- The POSIX compatibility test for shell scripts requires the tool `shfmt`.
- `git`, `clang-format` (version 12), and `cmake-format` to check formatting.
- `pkg-config` must be available (`check_external.sh` and `check_gen.sh`).
- A build environment including gcc (`check_gen.sh`).

- The Markdown Shell Recorder requires POSIX utilities (awk, grep, ...).

You can also use Docker to set up such an environment. There is a tutorial that guides you through the necessary steps.

## 445.7 Adding Tests

For plugins, adding ADD_TEST to add_plugin will execute the tests in testmod_${pluginname}.c. This is done by default for newly generated plugins.
Add CPP_TEST if the test is written in C++. Then testmod_${pluginname}.cpp will be used. These tests use the gtest test framework.
If the tests should not always be executed, the CMake function add_plugintest can be used instead. See scripts/cmake/Modules/LibAddPlugin.cmake for more information.
By using TEST_README in add_plugin (also enabled by default), Markdown Shell Recorder are expected to be in the README.md of the plugin.

## 445.8 Conventions

- All names of the test must start with test (needed by test driver for installed tests).
- No tests should run if ENABLE_TESTING is OFF.
- All tests that write to system: or spec: namespaces (e.g. with kdbSet or by mounting): should be tagged with kdbtests:

```
set_property(TEST testname PROPERTY LABELS kdbtests)
```

- should not run, if ENABLE_KDB_TESTING is OFF.
- should only write below
  - /tests/<testname> (e.g. /tests/ruby) and
  - system:/elektra (e.g. for mounts or globalplugins).
- Before executing tests, no keys must be present below /tests. The test cases need to clean up everything they wrote. (Including temporary files)
- If your test has memory leaks, e.g. because the library used leaks and they cannot be fixed, give them the label memleak with the following command:

```
set_property(TEST testname PROPERTY LABELS memleak)
```

- If your test modifies resources needed by other tests you also need to set RUN_SERIAL:

```
set_property(TEST testname PROPERTY RUN_SERIAL TRUE)
```

## 445.9 Strategy

The testing must happen on every level of the software to achieve a maximum coverage with the available time. In the rest of the document we describe the different levels and where these tests are.

### 445.9.1 CFramework

This is basically a bunch of assertion macros and some output facilities. It is written in pure C and very lightweight. It is located here.

### 445.9.2 ABI Tests

C ABI Tests are written in plain C with the help of `cframework`.
The main purpose of these tests are, that the binaries of old versions can be used against new versions as ABI tests.
So lets say we compile Elektra 0.8.8 (at this version dedicated ABI tests were introduced) in the `-full` variant. But when we run the tests, we use `libelektra-full.so.0.8.9` (either by installing it or by setting `LD_LIBRARY_PATH`). You can check with `ldd` which version is used.
The tests are located here.

### 445.9.3 C Unit Tests

C Unit Tests are written in plain C with the help of `cframework`.
It is used to test internal data structures of libelektra that are not ABI relevant.
ABI tests can be done on these tests, too. But by nature from time to time these tests will fail.
They are located here.

#### 445.9.3.1 Internal Functions

According to `src/libs/elektra/libelektra-symbols.map`, all functions starting with:

- `libelektra`
- `elektra`
- `kdb`
- `key`
- `ks`

get exported. Functions not starting with this prefix are internal only and therefore not visible in the test cases. Test internal functionality by including the corresponding C file.

### 445.9.4 Module Tests

The modules, which are typically used as plugins in Elektra (but can also be available statically or in the `-full` variant), should have their own tests.
Use the CMake macro `add_plugintest` for adding these tests.

### 445.9.5 C++ Unit Tests

C++ Unit tests are done using the Google Test framework. See architectural decision.
Use the CMake macro `add_gtest` for adding these tests.

### 445.9.6 Script Tests

Tests which need scripts are done using shell recorder or directly with POSIX shell commands. See architectural decision.
The script tests have different purposes:

- End to End tests (usage of tools as an end user would do)
- External compilation tests (compile and run programs as a user would do)
- Conventions tests (do internal checks that check for common problems)
- Meta Test Suites (run other test suites)

See here.

### 445.9.7 Shell Recorder

The more elegant way to specify script tests are via the so-called Shell Recorder using Markdown Syntax.
See here.

### 445.9.8 Fuzz Testing

We assume that your current working directory is a newly created build directory. First compile Elektra with afl (~e is source-dir of Elektra):

```
~e/scripts/dev/configure-debian -DCMAKE_C_COMPILER=/usr/src/afl/AFL-2.57b/afl-gcc
       -DCMAKE_CXX_COMPILER=/usr/src/afl/AFL-2.57b/afl-g++ ~e
make -j 5
```

Copy some import files to `testcase_dir`, for example:

```
mkdir -p testcase_dir
cp ~e/src/plugins/toml/toml/* testcase_dir
```

Fewer files is better. Then run, for example:

```
LD_LIBRARY_PATH=`pwd`/lib /usr/src/afl/AFL-2.57b/afl-fuzz -i testcase_dir -o findings_dir bin/kdb import
       user:/tests toml
```

Check if something is happening with:

```
watch kdb export user:/tests
```

### 445.9.9 ASAN

To enable sanitize checks use `ENABLE_ASAN` via CMake.
Then, to use ASAN, run `run_asan` in the build directory, which simply does:

```
ASAN_OPTIONS=symbolize=1 ASAN_SYMBOLIZER_PATH=$(which llvm-symbolizer) make run_all
```

It could also happen that you need to preload ASAN library, e.g.:

```
LD_PRELOAD=/usr/lib/clang/3.8.0/lib/linux/libclang_rt.asan-x86_64.so run_asan
```

or on Debian:

```
LD_PRELOAD=/usr/lib/llvm-3.8/lib/clang/3.8.1/lib/linux/libclang_rt.asan-x86_64.so run_asan
```

#### 445.9.9.1 macOS

If you use macOS you might want to use the `clang` versions provided by Homebrew, since it supports the `LeakSanitizer`. To use Homebrew's version of `clang` you need to first install LLVM:

```
brew install llvm
```

. After that change the `CC` and `CXX` environment variables to point to the clang tools provided by LLVM:

```
export CC=/usr/local/opt/llvm/bin/clang
export CXX=/usr/local/opt/llvm/bin/clang++
```

. Now run CMake and build Elektra just like you normally would. To enable the Leak Sanitizer you need to also set the variable `ASAN_OPTIONS` before you run a test:

```
export ASAN_OPTIONS=detect_leaks=1
```

### 445.9.10 CBMC

For bounded model checking tests, see `scripts/cbmc`.

### 445.9.11 Static Code Checkers

There is a number of static code checkers available for all kind of programming languages. The following section show how the most common ones can be used with `libelektra`.

#### 445.9.11.1 Cppcheck

`Cppcheck` can be used directly on a C or C++ source file by calling it with `cppcheck --enable=all <sourcefile>`. This way it might miss some header files though and thus doesn't detect all possible issues, but still gives useful hints in general.
To analyze the whole project, use it in conjunction with CMake by calling `cmake` with the parameter `-DCMAKE_EXPORT_COMPILE_COMMANDS=ON`. This way CMake creates a file called `compile_commands.json` in the build directory. Afterwards, call `cppcheck` with the CMake settings and store the output as xml:

```
cppcheck --project=compile_commands.json --enable=all -j 8 --xml-version=2 2> cppcheck_result.xml
```

Since the XML file is difficult to read directly, the best way is to convert it to an HTML report. Cppcheck already includes a tool for that, call it with the XML report:

```
cppcheck-htmlreport --file=cppcheck_result.xml --report-dir=cppcheck_report --source-dir=.
```

Now you can view the html report by opening `index.html` in the specified folder to get an overview of the issues found in the whole project.

#### 445.9.11.2 OCLint

OCLint is a static code analyzer for C, C++ and Objective C. To use this tool enable the CMake option `CMAKE_EXPORT_COMPILE_COMMANDS`. The steps below show a step-by-step guide on how to analyze files with OCLint.

1. Create a build directory if you have not done so already and change the working path to this directory:

```
mkdir -p build
cd build
```

1. Run CMake with the option `CMAKE_EXPORT_COMPILE_COMMANDS`:

```
cmake .. -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
```

1. Build Elektra

```
make
```

1. Run the `oclint` command specifying the files you want to analyze

```
cd ..
oclint -p build -no-analytics -enable-global-analysis -enable-clang-static-analyzer src/plugins/toml/*.c
```

#### 445.9.11.3 scan-build

scan-build is a tool that is usually bundled along with LLVM/Clang and is also primarily intended for C and C++ code. On macOS you have to install the package `llvm` with homebrew, then you'll find the tool in the folder `/usr/local/opt/llvm/bin/`.
To use it, change the C compiler and the C++ compiler to the LLVM analyzer. To do this, you can configure the project from scratch and prefix the cmake command with `scan-build`. Alternatively, set the c compiler to `ccc-analyzer` and the C++ compiler to `c++-analyzer` (bundled with LLVM/Clang).
Then you can build the project with `make` like usual, prefixing the command with `scan-build`. The `-o` option specifies where the html results get stored. Ensure you build the project from scratch, otherwise the analyzation might be incomplete.

```
scan-build -o ./scanbuild_result make -j 4
```

Afterwards, the report can be viewed by using the tool `scan-view`, also found in the llvm folder. The report is created in the folder specified above, along with the current date of the analyzation, for instance:

```
scan-view <path specified above>/2017-06-18-171027-27108-1
```

Alternatively, you can also open the `index.html` file in the aforementioned folder, but using the tool the report is enriched with further information.

#### 445.9.11.4 SonarLint

SonarLint is a static code checker primarily intended for Java. It is usually used by installing the corresponding plugin for the used IDE, then there is no further configuration required.

### 445.9.12 Randoop

For using the unit test generator randoop with the jna bindings, see `scripts/randoop/randoop`.

### 445.9.13 Code Coverage

Run:

```
make coverage-start
# now run all tests! E.g.:
make run_all
make coverage-stop
make coverage-genhtml
```

The HTML files can be found in the build directory in the folder `coverage`.

## 445.10 See Also

- COMPILE.
- INSTALL.
- BUILDSERVER.

# Chapter 446

# Introduction

In Elektra different forms of application integrations are possible:

1. A lightweight integration where configuration files are integrated in a global key database. This will not be discussed here, if you are interested please continue reading about mounting
2. Integration techniques without modifying the applications. This will also not be discussed here, if you are interested please read:
   - Intercept Environment
   - Intercept File System
3. Integration where applications directly use Elektra to read and store settings.
   (a) Using the low-level API.
   (b) Using the high-level API.

In this tutorial we will discuss (3.1), i.e., how to extend an application to directly access Elektra's key database. If you are new to Elektra, we recommend you familiarize yourself with the basic concepts using this guide, but when it comes to elektrifying your application (3.2) is mostly likely the better option. So take a look at how to use the high-level API.
When the application is fully integrated in Elektra's ecosystem following benefits arise:

- Benefits that shared libraries have, e.g.
  - All applications profit from fixes, optimization and new features
  - Less memory consumption, because the libraries executable instructions are only loaded once
  - Faster development time, because many non-trivial problems (e.g. OS-dependent resolving of configuration file names with atomic updates) are already solved and tested properly
- The administrator can choose:
  - the configuration file syntax (e.g. XML or JSON)
  - notification and logging on configuration changes
  - defaults on absence of values using specifications
  - and all other features that plugins provide
- The parsing result is guaranteed to be the same because the same parser will be used.
- Other applications can use your configuration as override or as fallback (see below)

## 446.1 Elektrify

We call the process of making applications aware of other's configuration "to elektrify". This tutorial is suited both for new and existing applications.
As first step, locate places where configuration is parsed or generated. Afterwards, use Elektra's data structures instead at these locations. Before we are going to describe how to do this, we will describe some possibilities to keep all advantages your previous configuration system had.
You can keep code you want within Elektra as plugins. This allows your application, and other applications participating in Elektra's ecosystem to access your configuration. Doing this, the syntax of the configuration file stays the same as before. You can keep the same validation as you had before. The application profits from Elektra's infrastructure solving basic issues like getting configuration from other parts of the system, update and conflict detection, and resolving of the file name. In particular we gain a lot because every other program can also access the configuration of your software.
If you do not have the code or want to get rid of it, you can use a variety of already implemented plugins to extend the functionality of the configuration system. There are plenty of plugins that parse and generate configuration files in different formats, do syntactic checks, do notifications (e.g. via dbus), and write out events in their log files.
New applications do not have the burden to stay compatible with the configuration system they had to before. So they will prefer to use more standard plugins and contribute to make them flawless. But they can also use self-written plugins for adding needed behavior or cross-cutting concerns.
To sum up, if a developer wants to **elektrify** software, he or she can do that without any need for changes to the outside world regarding the format and semantics of the configuration. In the interconnected world it is a matter of time until other software also wants to access the configuration, and with elektrified software it is possible for every application to do so.

## 446.2 Get Started

As first step in a C-application you need to create an in-memory `Key`. Such a `Key` is Elektra's atomic unit and consists of:

- a unique name
- a value
- metadata

`Key`s are either associated with entries in configuration files or used as arguments in the API to transport some information.
Thus a key is in-memory and does not need any of the other Elektra objects. We always can create one (the tutorial will use the C-API, but it describes general concepts useful for other languages in the same way):

```
Key *parentKey = keyNew("/sw/org/myapp/#0/current", KEY_END);
```

- The first argument of `keyNew` is the name of the key. It consists of different parts, `/` is the hierarchy-separator:
  - `sw` is for software
  - `org` is a URL/organization name to avoid name clashes with other application names. Use only one part of the URL/organization, so e.g. `kde` is enough.
  - `myapp` is the name of the most specific component that has its own configuration
  - `#0` is the major version number of the configuration (increment if you need to introduce incompatible changes).
  - `current` is the profile to use. Administrators need it if they want to start up applications with different configurations.
- `KEY_END` as C needs a proper termination of variable length arguments.

The key name is standardized to make it easier to locate configuration.

- `Read more about key-functions in API doc.`
- Read more about key names here.

Now we have the `Key` we will use to pass as argument. First we open our key database (KDB):
```
KDB *repo = kdbOpen(parentKey);
```
A `Key` is seldom alone, but they are often found in groups, as typical in configuration files. To represent many keys (a set of keys) Elektra has the data structure `KeySet`. Because the `Key`'s name is unique we can lookup keys in a `KeySet` without ambiguity. Furthermore, we can iterate over all `Key`s in a `KeySet` without a hassle. To create an empty `KeySet` we use:
```
KeySet *conf = ksNew(200, KS_END);
```

- 200 is an approximation for how many `Key`s we think we will have in the `KeySet conf`, intended for optimization purposes.
- After the first argument we can list built-in keys that should be available in any case.
- The last argument needs to be `KS_END`.

Now we have everything ready to fetch the latest configuration:
```
kdbGet(repo, conf, parentKey);
```
Note it is important for applications that the parentKey starts with a slash `/`. This ensures pulling in all keys of the so-called namespace. Such a name cannot physically exist in configuration files, but they are the most important key names to actually work with configuration within applications as we will see when introducing `ksLookup`.

## 446.3 Lookup

To lookup a key, we use:
```
Key *k = ksLookupByName(conf,
        "/sw/org/myapp/#0/current/section/subsection/key",
        0);
```
We see in this example that only Elektra paths are hard coded in the application, no configuration file or similar.
As already mentioned keys starting with slash `/` do not exist in configuration files, but are "representatives", "proxies" or "logical placeholders" for keys from any other namespace.
So that every tool has a consistent view to the key database it is vital that every application does a `ksLookup` for every key it uses. So even if your application iterates over keys, always remember to do a cascading lookup for every single key!
Thus we are interested in the value we use:
```
char *val = keyString(k);
```
We need to convert the configuration value to the data type we need.
To do this manually has severe drawbacks:

- hard coded names might have typos or might be inconsistent
- tedious handling if key or value might be absent
- always calling `ksLookup` which gets tiresome for arrays
- converting to needed data type is error-prone

So (larger) applications should not directly use `KeySet`, but instead use code generation to provide a type-safe frontend.
For more information about that, continue reading `here`.

## 446.4 Specification

Now, we have a fully working configuration system without any hard coded information (such as configuration files). We already gained something. But, we did not discuss how we can actually achieve application integration, the goal of Elektra.
Elektra 0.8.11 introduces the so-called specification for the application's configuration, located below its own namespace `spec`. The specification itself also consists of (meta) key-value pairs.
Keys in `spec` allow us to specify which keys the application reads, which fallback they might have and which is the default value using metadata.

### 446.4.1 Links

The implementation of links are in `ksLookup`. When using cascading keys (those starting with `/`), the following features are now available (in the metadata of respective `spec`-keys):

- `override/#`: use these keys *in favor* of the key itself (note that `#` is the syntax for arrays, e.g. `#0` for the first element, `#_10` for the 11th and so on)
- `namespace/#`: instead of using all namespaces in the predefined order, one can specify namespaces to search in a given order
- `fallback/#`: when no key was found in any of the (specified) namespaces the `fallback`-keys will be searched
- `default`: the value to use if nothing else was found

You can use those features like following:

```
kdb set /overrides/test "example override"
sudo kdb meta-set spec:/test override/#0 /overrides/test
```

This technique provides complete transparency how a program will fetch a configuration value. In practice that means that:

```
kdb get "/sw/org/myapp/#0/current/section/subsection/key"
```

, will give you the *exact same value* as the application gets when it uses the above lookup C code.
What we do not see in the program above are the default values and fallbacks. They are also present in the specification (namespace `spec`).
So lets say, that another application `otherapp` has the value we actually want. We want to improve the integration.
In the case that we do not have a value for `/sw/org/myapp/#0/current/section/subsection/key`, we want to use `/sw/otherorg/otherapp/#0/current/section/subsection/key`.
So we specify:

```
kdb meta-set spec:/sw/org/myapp/#0/current/section/subsection/key \
    "fallback/#0" /sw/otherorg/otherapp/#0/current/section/subsection/key
```

Voila, we have done a system integration between `myapp` and `otherapp`!
Note that the fallback, override and cascading works on *key level*, and not like most other systems have implemented, on configuration *file level*.
To make this work within your application make sure to always call `ksLookup` before using a value from Elektra.

### 446.4.2 Specfiles

We call the files, that contain a complete schema for configuration below a specific path in form of metadata, *Specfiles*.
Particularly a *Specfile* contains metadata that defines

- the mount points of paths,
- the plugins to load and
- the behavior of these plugins.

(note that the `\\` are due to `Markdown Shell Recorder`, do not copy them to your shell)

```
sudo kdb mount tutorial.ecf spec:/sw/org/myapp/#0/current"
cat « HERE | kdb import spec:/sw/org/myapp/#0/current ni  \
[]                                          \
 mountpoint = my-config-file.ini            \
 infos/plugins = ini validation             \
                                            \
[section/subsection/key]                    \
fallback/#0=/sw/otherorg/otherapp/#0/current/section/subsection/key  \
description = A description of the key      \
HERE
kdb meta-ls spec:/sw/org/myapp/#0/current # verify if specification is present now
#> infos/plugins
#> mountpoint
```

Now we apply this *Specfile* to the key database to all keys below `/sw/org/myapp/#0/current`:

```
kdb spec-mount /sw/org/myapp/#0/current
```

Then the configuration of our application will be in my-config-file.ini (because of `mountpoint` in the specification) and it will use the INI format (because of `infos/plugins` in the specification). `section/subsection/key` contains the specification of what we already specified imperatively before.
For a description which metadata is available, have a look in `METADATA.ini`.

## 446.5 Conclusion

Elektra does not hard code any configuration data in your application. Using the `default` specification, we even can startup applications without any configuration file *at all* and still do not have anything hard coded in the applications binary. Furthermore, by using cascading keys for `kdbGet()` and `ksLookup()` Elektra gives you the possibility to specify how to retrieve configuration data. In this specification you can define to consider or prefer configuration data from other applications or shared places. Doing so, we can achieve configuration integration.

## 446.6 See Also

- how to validate configuration with the specification

# Chapter 447

# Arrays

## 447.1 Key-Value Pairs

The main building block of Elektra's database are hierarchical `key-value pairs`. You can create such a pair using `kdb set`:

```
kdb set user:/tests/parent value
#> Create a new key user:/tests/parent with string "value"
```

. The command above created a key `user:/tests/parent` with the value `value`. Since Elektra uses a hierarchical database we can also create keys **below** `user:/tests/parent`:

```
# We can create keys with an empty value by passing an empty string.
kdb set user:/tests/parent/son ""
#> Create a new key user:/tests/parent/son with string ""
kdb set user:/tests/parent/daughter ""
#> Create a new key user:/tests/parent/daughter with string ""
kdb set user:/tests/parent/daughter/grandchild ""
#> Create a new key user:/tests/parent/daughter/grandchild with string ""
```

. We can check the hierarchy of the keys using `kdb ls` and retrieve data using `kdb get`:

```
# Elektra sorts keys alphabetically
kdb ls user:/tests/parent
#> user:/tests/parent
#> user:/tests/parent/daughter
#> user:/tests/parent/daughter/grandchild
#> user:/tests/parent/son
kdb get user:/tests/parent
#> value
kdb get user:/tests/parent/daughter
#>
```

## 447.2 Array Keys

Since Elektra keys sorts keys alphabetically, we can use the key-value pair structure described above to store sequences of values (aka. arrays).

### 447.2.1 Empty Arrays

For an **empty array** (`[]`) we just add the metakey `array`:

```
# Create an empty array with the name `user:/tests/sequence`
kdb meta-set user:/tests/sequence array ""
```

### 447.2.2 Array Elements

To create an **array element** we start the basename of a key with the `#` character and add the index of the array element afterwards. For example, the commands below adds three elements to our array `user:/tests/sequence`:

```
kdb set user:/tests/sequence/#0 'First Element'
#> Create a new key user:/tests/sequence/#0 with string "First Element"
kdb set user:/tests/sequence/#1 'Second Element'
#> Create a new key user:/tests/sequence/#1 with string "Second Element"
# Arrays do not need to be contiguous
kdb set user:/tests/sequence/#3 'Fourth Element'
#> Create a new key user:/tests/sequence/#3 with string "Fourth Element"
```

. As you can see above arrays can contain "empty fields": The key `user:/tests/sequence/#2` is missing.

As Elektra keys are sorted alphabetically, it will automatically prepend with as many _ characters as needed to keep the numerical ordering of the array and the alphabetical order identical:

```
kdb set user:/tests/sequence/#10 'Eleventh Element'
#> Create a new key user:/tests/sequence/#_10 with string "Eleventh Element"
# List all array elements
kdb ls user:/tests/sequence/
#> user:/tests/sequence/#0
#> user:/tests/sequence/#1
#> user:/tests/sequence/#3
#> user:/tests/sequence/#_10
```

### 447.2.3 Metadata

Elektra's arrays **require** that you always add the metakey `array` to the array parent. Otherwise, the values below the parent will not be interpreted as array elements, but rather as normal key-value pairs. To make the `array` metakey more useful storage plugins should save the basename of the last key in the array parent. This of course works only, if the plugins already stores this information in the config file, either

- implicitly (e.g. the data below the array parent is stored as array/list/sequence in the config file), or
- explicitly (e.g. the plugin stores the metakey `array` directly in the config file)

. Either way, in some situations you might have to add this value manually via `kdb meta-set`:

```
# Add array elements
kdb set user:/tests/favorites/superheros/#0 'One-Punch Man'
#> Create a new key user:/tests/favorites/superheros/#0 with string "One-Punch Man"
kdb set user:/tests/favorites/superheros/#1 'Mermaid Man and Barnacle Boy'
#> Create a new key user:/tests/favorites/superheros/#1 with string "Mermaid Man and Barnacle Boy"
kdb set user:/tests/favorites/superheros/#99999 'The guy with the bow and arrow'
#> Create a new key user:/tests/favorites/superheros/#____99999 with string "The guy with the bow and arrow"
# The metakey `array` should save the basename of the last element.
kdb meta-set user:/tests/favorites/superheros array '#99999'
```

. This way you can always retrieve the last element of an array easily:

```
kdb get user:/tests/favorites/superheros/`kdb meta-get user:/tests/favorites/superheros array`
#> The guy with the bow and arrow
```

## 447.3 Closing Remarks

We close this tutorial by removing the data created by the commands above:

```
kdb rm -r user:/tests
```

# Chapter 448

# Benchmarking

## 448.1 Execution Time

One of the usual questions for a standard benchmark is, how much more or less time two or more different programs take to execute on the same hardware. This tutorial will introduce some tools and techniques that will help you to answer this question. For that purpose we compare the time it takes for certain YAML storage plugins to translate YAML data into Elektra's key set structure. Most of the techniques we describe here should be applicable too, if you want to compare the run-time of other parts of Elektra. If you want to know why a certain part of Elektra takes a long time to execute, then you might also be interested in the profiling tutorial.

### 448.1.1 Translating Elektra

If you have never translated the code base of Elektra before, then please take a look here before you continue. Usually you want to compare the execution time of the fastest version of a compiled binary. For that purpose it makes sense to change the CMake build type to `Release`, which means that the generated build system will optimize the code and strip debug symbols. You should also disable the logger and debug code. An example CMake command that uses `Ninja` as build tool could look like this:

```
mkdir build
cd build
cmake -GNinja ..                \
      -DCMAKE_BUILD_TYPE=Release \
      -DENABLE_LOGGER=OFF       \
      -DENABLE_DEBUG=OFF        \
      -DPLUGINS=ALL
ninja
cd .. # Change working directory back to the root of repository
```

### 448.1.2 Using the Plugin Benchmark Helper Tool

Elektra already includes a tool that helps you to benchmark the `get` and `set` methods of a certain plugin called `benchmark_plugingetset`. To show you how to use `benchmark_plugingetset`, we create a file named `test.yamlcpp.in` with the following content:

```
- You,
- Me, &
- The Violence
```

and save it in the folder `benchmarks/data`:

```
mkdir -p benchmarks/data
printf -- '- You,\n'       >  benchmarks/data/test.yamlcpp.in
printf -- '- Me, &\n'      » benchmarks/data/test.yamlcpp.in
printf -- '- The Violence' » benchmarks/data/test.yamlcpp.in
```

. As you can see the filename has to use the pattern:

```
test.$plugin.in
```

, where `$plugin` specifies the name of the plugin the benchmark tool should call. We can now call the `get` method of the plugin YAML CPP using the following shell command

```
build/bin/benchmark_plugingetset benchmarks/data      user    yamlcpp       get
#                                       ↑             ↑       ↑            ↑
#                                parent directory  namespace  plugin   only use `get`
#                                 of config file                      plugin method
```

. If you can want you can also use the `time` utility to measure the execution time of the last command:

```
time build/bin/benchmark_plugingetset benchmarks/data user yamlcpp get
#>       0.00 real         0.00 user         0.00 sys
```

. As you can see in the output above a real configuration file that tests the performance of the YAML CPP plugin should be much larger.

### 448.1.3 Comparing Execution Times

Now that you know how to execute `benchmark_plugingetset`, you can use it to compare the performance of different plugins. Since you usually want

- to run `benchmark_plugingetset` multiple times, and
- compare different plugins

it makes sense to use a benchmarking tool such as `hyperfine` for that task. For our tutorial we assume that you copied the file `keyframes.yaml` to the locations

- `benchmarks/data/test.yamlcpp.in`, and
- `benchmarks/data/test.yanlr.in`

. You can do that using the following commands:

```
mkdir -p benchmarks/data
curl -L https://github.com/ElektraInitiative/rawdata/raw/master/YAML/Input/keyframes.yaml -o
      benchmarks/data/test.yamlcpp.in
cp benchmarks/data/test.yamlcpp.in benchmarks/data/test.yanlr.in
```

. Afterwards you can use:

```
hyperfine --warmup 3 'build/bin/benchmark_plugingetset benchmarks/data user yamlcpp get' \
                     'build/bin/benchmark_plugingetset benchmarks/data user yanlr get'
```

to compare the performance of the plugins. The output of this benchmark would look something like this:

```
Benchmark #1: build/bin/benchmark_plugingetset benchmarks/data user yamlcpp get
  Time (mean ± σ):      18.3 ms ±   0.9 ms    [User: 15.2 ms, System: 1.4 ms]
  Range (min ... max):    17.1 ms ...  21.2 ms    136 runs
Benchmark #2: build/bin/benchmark_plugingetset benchmarks/data user yanlr get
  Time (mean ± σ):      16.2 ms ±   0.9 ms    [User: 12.7 ms, System: 1.7 ms]
  Range (min ... max):    14.8 ms ...  20.0 ms    161 runs
Summary
  'build/bin/benchmark_plugingetset benchmarks/data user yanlr get' ran
    1.13 ± 0.08 times faster than 'build/bin/benchmark_plugingetset benchmarks/data user yamlcpp get'
```

. You can now remove the input files and the folder `benchmarks/data`:

```
rm benchmarks/data/test.yamlcpp.in
rm benchmarks/data/test.yanlr.in
rmdir benchmarks/data
```

# Chapter 449

# Cascading Lookups

This tutorial assumes that you are already familiar with namespaces. This tutorial will only explain cascading lookup. When Elektra looks up a *cascading key* (i.e. key names without a namespace and a leading slash `/`, the namespaces are searched in the following order:

- `spec` (contains metadata, e.g. to modify Elektra's lookup behavior)
- `proc` (process-related information)
- `dir` (directory-related information, e.g. `.git` or `.htaccess`)
- `user` (user configuration)
- `system` (system configuration)

If a key, for example, exists in both **user** and **system** namespace, the key in the **user** namespace takes precedence over the one in the **system** namespace. If there is no such key in the **user** namespace the key in the **system** namespace acts as a fallback.
But let's demonstrate this with an example:

### 449.0.1 Add a Key to the system Namespace

Configuration in the **system** namespace is the same for all users. Therefore, this namespace provides a default or fallback configuration.
With the default Elektra installation only an administrator can update configuration settings within the **system** namespace.

```
# Backup-and-Restore:/tests/tutorial
# Backup old override specification
kdb set user:/tests/overrides $(mktemp)
kdb export system:/tests/overrides dump > $(kdb get user:/tests/overrides)
kdb get /tests/tutorial/cascading/#0/current/test
# RET: 11
# STDERR: Did not find key '/tests/tutorial/cascading/#0/current/test'
# Now add the key ...
sudo kdb set system:/tests/tutorial/cascading/#0/current/test "hello world"
#> Create a new key system:/tests/tutorial/cascading/#0/current/test with string "hello world"
# ... and verify that it exists
kdb get /tests/tutorial/cascading/#0/current/test
#> hello world
```

### 449.0.2 Add a Key to the user Namespace

A user may now want to override the configuration in **system**, so he/she sets a key in the **user** namespace:

```
kdb set user:/tests/tutorial/cascading/#0/current/test "hello galaxy"
#> Create a new key user:/tests/tutorial/cascading/#0/current/test with string "hello galaxy"
# This key masks the key in the system namespace
kdb get /tests/tutorial/cascading/#0/current/test
#> hello galaxy
```

Note that configuration in the **user** namespace only affects *this* user. Other users would still get the key from the **system** namespace.

### 449.0.3 Add a Key to the dir Namespace

The **dir** namespace is associated with a directory. The configuration in the **dir** namespace applies to the **current working directory** and all its subdirectories. This is useful if you have project specific settings (e.g. your Git configuration or a .htaccess file).
As **dir** precedes the **user** namespace, configuration in **dir** can overwrite user configuration:

```
# create and change to a new directory ...
mkdir kdbtutorial
cd kdbtutorial
# ... and create a key in this directories dir-namespace
# By default this data will be saved in the directory `.dir`.
kdb set dir:/tests/tutorial/cascading/#0/current/test "hello universe"
#> Create a new key dir:/tests/tutorial/cascading/#0/current/test with string "hello universe"
# This key masks the key in the system namespace
kdb get /tests/tutorial/cascading/#0/current/test
#> hello universe
# But is only present in the associated directory
cd ..
kdb get /tests/tutorial/cascading/#0/current/test
# hello galaxy
```

### 449.0.4 Add a Key to the proc Namespace

The **proc** namespace is not accessible by the command line tool **kdb**, as it is unique for each running process using Elektra. So we have to omit an example for this namespace at this point. Elektrified applications can use this namespace to override configuration from other namespaces internally.

### 449.0.5 Add a Key to the spec Namespace

The **spec** namespace is used to store metadata about keys and therefore Elektra handles the **spec** namespace differently to other namespaces. The following part of the tutorial is dedicated to the impact of the **spec** namespace on cascading lookups.

## 449.1 Write Operations and the cascading Namespace

If a value is to be written to a cascading key, i.e., a key starting with '/', only cascading keys that resolve to an existing key will be used.
For example,

```
kdb set /tests/tutorial/cascading/#0/current/cascading_write_test value
# STDERR: Aborting: A cascading write to a non-existent key is ambiguous.
# RET: 12
kdb meta-set /tests/tutorial/cascading/#0/current/cascading_write_test metakey metavalue
# STDERR: Aborting: A cascading write to a non-existent key is ambiguous.
# RET: 12
```

will both fail, as no matching key exists. Since there are multiple hypothetical key names that would match the cascading name (keys of specific namespaces like user:, system:, ...) if they existed, it is not clear what the user intended and thus an error is produced.
To make the previous two operations meaningful, a matching key in the user: namespace is created:

```
kdb set user:/tests/tutorial/cascading/#0/current/cascading_write_test value
#> Create a new key user:/tests/tutorial/cascading/#0/current/cascading_write_test with string "value"
```

Now, the operations operate on a well-defined key and succeed this time:

```
kdb set /tests/tutorial/cascading/#0/current/cascading_write_test value
#> Set string to "value"
#> Using name user:/tests/tutorial/cascading/#0/current/cascading_write_test
kdb meta-set /tests/tutorial/cascading/#0/current/cascading_write_test metakey metavalue
#> Using name user:/tests/tutorial/cascading/#0/current/cascading_write_test
```

## 449.2 Override Links

The `spec` namespace is special as it can completely change how the cascading lookup works.
During a cascading lookup for a specific key, the default Elektra behavior can be changed by a corresponding `spec-key`, i.e. a key in the **spec** namespace **with the same name**.
For example, the metadata `override/#0` of the respective `spec-key` can be specified to use a different key in favor of the key itself. This way, we can implement a redirect or symlink like behavior and therefore even config from current folder (`dir` namespace) can be overwritten.
The cascading lookup will consider the following **metadata keys** of `spec-key`:

1. `override/#n` redirect to one of the specified keys
2. `namespaces/#n` specifies which namespaces will be considered and in which order
3. `fallback/#n` if no key was found these keys will act as a fallback
4. `default` defines a default value for the key if none of the keys was found

**Note:** `override/#n`, `namespaces/#n` and `fallback/#n` are Elektra **array keys**. This means, such keys can exist several times, each with a different number for `n`, e.g. `override/#0`, `override/#1`... This way, we can define multiple values for a specific key with a defined order.
As you can see, override links are considered before everything else, which makes them really powerful.
Consider the following example:
First, we create a target key to demonstrate the override link mechanism:

```
sudo kdb set system:/tests/overrides/test "hello override"
#> Create a new key system:/tests/overrides/test with string "hello override"
```

Override links can be defined by adding them to the `override/#` metadata array key of the corresponding `spec-key`:

```
sudo kdb meta-set spec:/tests/tutorial/cascading/#0/current/test override/#0 /tests/overrides/test
```

Now when doing a cascading lookup, we get the value of our target key instead of the specified one:

```
kdb get /tests/tutorial/cascading/#0/current/test
#> hello override
```

As we used a cascading key for our override link (`/tests/overrides/test`) we can use this to allow users to provide their own `tests/overrides/test` keys. If a user sets the `/tests/overrides/test` key, the **user** namespace will be used (for a non-root user) and therefore the new target for our `/tests/tutorial/cascading/#0/current/test` key will be `user:/tests/overrides/test` instead of `system:/tests/overrides/test`.

```
kdb set user:/tests/overrides/test "hello user"
#> Create a new key user:/tests/overrides/test with string "hello user"
kdb get /tests/tutorial/cascading/#0/current/test
#> hello user
```

Furthermore, you can specify a custom order for the namespaces, set fallback keys and more. For more information, read the `elektra-spec` help page.

## 449.3 Cleanup

As last part in this tutorial we remove the modifications to the database we made previously.

```
kdb rm -r user:/tests/tutorial/
sudo kdb rm -r system:/tests/tutorial
sudo kdb rm -r system:/tests/overrides
kdb import system:/tests/overrides dump < $(kdb get user:/tests/overrides)
rm $(kdb get user:/tests/overrides)
kdb rm user:/tests/overrides
sudo kdb rm -r spec:/tests/tutorial/
rm -r kdbtutorial
```

# Chapter 450

# Changetracking

Elektra provides developers of applications and plugins a way to determine changes made to a `KeySet` relative to the last known state of the key database. This can be useful if you are writing a plugin that works with changes of the configuration.

## 450.1 Basics

The two headers you have to use are `kdbchangetracking.h` and `kdbdiff.h`. Those declare the `elektraChangeTracking*` and `elektraDiff*` functions.
The two main data structures you will encounter are `ChangeTrackingContext` and `ElektraDiff`.

## 450.2 Getting the difference between KeySets

If all you want to do is get the difference between two in-memory `KeySet` objects, use the function `elektraDiffCalculate`.

```
KeySet * originalKeySet;
KeySet * modifiedKeySet;
Key * parentKey;
// Calculate the difference of the keys below and including parentKey
ElektraDiff * diff = elektraDiffCalculate (modifiedKeySet, originalKeySet, parentKey);
// Extract useful information, see section 'Working with the diff'
elektraDiffDel (diff);
```

## 450.3 Getting the changes within a transaction in a plugin

If you are writing a plugin, you can use `elektraChangeTrackingGetContextFromPlugin` to get the current `ChangeTrackingContext`. Then, use `elektraChangeTrackingCalculateDiff` to calculate the changes to the KDB.

```
int myPluginSet (Plugin * handle, KeySet * returned, Key * parentKey)
{
        const ChangeTrackingContext * ctx = elektraChangeTrackingGetContextFromPlugin (handle);
        ElektraDiff * diff = elektraChangeTrackingCalculateDiff (returned, ctx, parentKey);
        // Don't be impatient, we will soon look into what we can do with the diff
        elektraDiffDel (diff);
        return 1;
}
```

## 450.4 Getting the changes as an application developer

If you are elektrifying your app, you can use `elektraChangeTrackingGetContextFromKdb` to get the current `ChangeTrackingContext` for your KDB instance. Then, use `elektraChangeTrackingCalculateDiff` to calculate the changes to the KDB.

```
KDB * kdb;
KeySet * myKeySet;
const ChangeTrackingContext * ctx = elektraChangeTrackingGetContextFromKdb (kdb);
ElektraDiff * diff = elektraChangeTrackingCalculateDiff (myKeySet, ctx, parentKey);
// Don't be impatient, we will soon look into what we can do with the diff
elektraDiffDel (diff);
```

## 450.5 Working with the diff

Congratulations! You've got your diff! Now what? Elektra provides you with different functions to determine added, removed and modified keys, as well as added, removed and modified metadata.
You can use the following methods to reason about changes to the keyset as a whole:

```
// Get keys that are in modifiedKeySet but not in originalKeySet
KeySet * addedKeys = elektraDiffGetAddedKeys (diff);
// Get keys that are in originalKeySet but not in modifiedKeySet
KeySet * removedKeys = elektraDiffGetRemovedKeys (diff);
// Get keys that are in both KeySets, but have either different values or different metadata.
// The returned keys will have the old values and metadata
KeySet * modifiedKeys = elektraDiffGetModifiedKeys (diff);
ksDel (addedKeys);
ksDel (removedKeys);
ksDel (modifiedKeys);
```

Use these methods to get information about single keys:

```
Key * myKey; // A key that we want to check.
// whether the value of myKey has been changed
bool valueChanged = elektraDiffKeyValueChanged (diff, myKey);
// determine if only the metadata of myKey has been changed
bool onlyMetaChanged = elektraDiffKeyOnlyMetaChanged (diff, myKey);
// get all metadata that has been added to myKey
KeySet * addedMeta = elektraDiffGetAddedMetaKeys (diff, myKey);
// get all metadata that has been removed from myKey
KeySet * removedMeta = elektraDiffGetRemovedMetaKeys (diff, myKey);
// get all metadata that has been modified.
// the returned keyset will contain the old metadata
KeySet * modifiedMeta = elektraDiffGetModifiedMetaKeys (diff, myKey);
ksDel (addedMeta);
ksDel (removedMeta);
ksDel (modifiedMeta);
```

# Chapter 451

# How-To: Merging

## 451.1 Introduction

Elektra takes the semantic aspects of configuration files into account when merging them into one result file. Those configuration files are represented as key sets in Elektra. Key sets are collections of name-value pairs.
The `kdb` tool allows users to access and perform functions on the Elektra key database from the command line. We added a new command to this very useful tool, the `cmerge` command. This command allows a user to perform a three-way merge of key sets from the `kdb` tool.
The syntax to use this tool is:
`kdb cmerge [OPTIONS] our their base result`
`our`, `their` and `base` represent the three keys that are used in a three-way merge. As in the `diff3` tool, required are the three files MYFILE (our), OLDFILE (base) and YOURFILE (their) The result off the three-way merge will be stored in `result`.

## 451.2 Simple example

The easiest case is if all three versions contain equal data.

```
kdb set user:/tests/base a
#> Create a new key user:/tests/base with string "a"
kdb set user:/tests/their a
#> Create a new key user:/tests/their with string "a"
kdb set user:/tests/our a
#> Create a new key user:/tests/our with string "a"
kdb cmerge user:/tests/our user:/tests/their user:/tests/base user:/tests/result
kdb get user:/tests/result
#> a
```

We change the key for another example.

```
kdb set user:/tests/our b
#> Set string to "b"
```

Using a `result` path that is not empty gives an error. The option `-f` can be used to override. **Attention!** This deletes existing keys below `result`.

```
kdb cmerge user:/tests/our user:/tests/their user:/tests/base user:/tests/result
# RET: 3
# There are keys in the result path. Use -f to override them.
kdb cmerge -f user:/tests/our user:/tests/their user:/tests/base user:/tests/result
kdb get user:/tests/result
#> b
```

We can use the same key multiple times in a single call to cmerge.

```
kdb set user:/tests/same a
#> Create a new key user:/tests/same with string "a"
kdb cmerge -f user:/tests/same user:/tests/same user:/tests/same user:/tests/result
kdb get user:/tests/result
#> a
```

### 451.2.1 hosts

As a real-world example, we import three different (see the comment) versions of a hosts file.

```
echo "127.0.0.1       localhost
127.0.1.1       computer
# BASE The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
```

```
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters" | kdb import user:/tests/hosts/base hosts
echo "127.0.0.1       localhost
127.0.1.1       computer
# OUR The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters" | kdb import user:/tests/hosts/our hosts
echo "127.0.0.1       localhost
127.0.1.1       computer
# THEIR The following lines are desirable for IPv6 capable hosts
::2     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters" | kdb import user:/tests/hosts/their hosts
kdb cmerge user:/tests/hosts/our user:/tests/hosts/their user:/tests/hosts/base user:/tests/hosts/result
```

The merge notices that only one of the three versions of the key `ip6-localhost` has changed. Assuming that this was an update it puts the new value in the result.

```
kdb get user:/tests/hosts/result/ipv6/ip6-localhost
#> ::2
```

## 451.3 Metadata

Metadata gets merged as well. We do not follow a complicated approach for this topic. When a key-value pair is chosen from the three versions to be present in the result it takes all its metadata with it.

In case that the values of some keys are equal, the `our` version wins and consequently the metadata of the `our` version is used. The reason for this is that users might have used the metadata for personal comments.

To demonstrate this, we continue the hosts example:

```
kdb meta-get user:/tests/hosts/result/ipv6/ip6-localhost comment/#2
#>  THEIR The following lines are desirable for IPv6 capable hosts
```

We set up some keys:

```
kdb set user:/tests/meta/base equal
#> Create a new key user:/tests/meta/base with string "equal"
kdb meta-set user:/tests/meta/base comment/#0 "This is the original inline comment"
kdb meta-set user:/tests/meta/base comment/#1 "This is the first line of the original comment above the key"
kdb meta-set user:/tests/meta/base comment/#2 "This is the second line of the original comment above the
      key"
kdb set user:/tests/meta/their equal
#> Create a new key user:/tests/meta/their with string "equal"
kdb meta-set user:/tests/meta/their comment/#0 "This is their inline comment"
kdb meta-set user:/tests/meta/their comment/#1 "This is the first line of their comment above the key"
kdb meta-set user:/tests/meta/their comment/#2 "This is the second line of their comment above the key"
kdb set user:/tests/meta/our equal
#> Create a new key user:/tests/meta/our with string "equal"
kdb meta-set user:/tests/meta/our comment/#0 "This is your custom inline comment"
kdb meta-set user:/tests/meta/our comment/#1 "This is the first line of your custom comment above the key"
kdb meta-set user:/tests/meta/our comment/#2 "This is the second line of your custom comment above the key"
kdb cmerge user:/tests/meta/our user:/tests/meta/their user:/tests/meta/base user:/tests/meta/metaFromOur
```

Now we can check if the metadata has been merged as expected.

```
kdb meta-get user:/tests/meta/metaFromOur comment/#0
#> This is your custom inline comment
kdb meta-get user:/tests/meta/metaFromOur comment/#1
#> This is the first line of your custom comment above the key
kdb meta-get user:/tests/meta/metaFromOur comment/#2
#> This is the second line of your custom comment above the key
```

If a key is part of the result because its value has changed then the result will also contain the metadata of that key.

```
kdb set user:/tests/meta/their different
#> Set string to "different"
kdb cmerge user:/tests/meta/our user:/tests/meta/their user:/tests/meta/base
      user:/tests/meta/metaFromChanged
```

We can test again if the result meets our expectations.

```
kdb meta-get user:/tests/meta/metaFromChanged comment/#2
#> This is the second line of their comment above the key
```

## 451.4 Arrays

cmerge uses LibGit2 to handle arrays in an efficient manner.

```
echo "one\
two\
three\
four\
five" | kdb import user:/tests/arrays/original line
echo "previous\
```

```
one\
two\
three\
four\
five" | kdb import user:/tests/arrays/changed line
kdb cmerge -f user:/tests/arrays/changed user:/tests/arrays/original user:/tests/arrays/original
      user:/tests/arrays/result
kdb get user:/tests/arrays/result/#0
#> previous
```

## 451.5 Scripts

There are two tools of which cmerge is the central tool:

1. `kdb install-config-file` installs or merges configuration files from the file system into Elektra. There is a tutorial for this tool, too.
2. `kdb cmerge-config-files` performs a three-way merge on three files using Elektra

## 451.6 Calling the API

All the tools that use the merge library rely on the same API. An exemplary call to this API can be found in the `examples folder`.

# Chapter 452

# Code-generator

This guide focuses on writing new templates for the code-generator. For using the code generator take a look at the man-page `kdb-gen(1)`. If you want to use the code-generator specifically for the high-level API, you may want to follow this guide.

## 452.1 Basics

The underlying framework of `kdb gen` is quite flexible. It is based on the `mustache templating system`. Concretely we use `this C++ library` as a basis.
To distinguish the user facing parts of `kdb gen` from the internal layer interfacing `kdb gen` to the mustache library, we will call the internal layer the *framework*.
The file `src/tools/kdb/gen.cpp` implements the command-line interface (CLI) and is of little interest to this guide. Instead we will focus on the framework that is invoked via the CLI. The bulk of the framework is implemented in the classes `GenTemplate and GenTemplateList` (and of course kainjow's mustache library).
First we need to define a few terms:

- *template name*: This is the main identifier of a template. It is used in the command line to choose the template.
- *template base name*: Unlike the template name, this identifier is internal to the framework. It will not be seen by users of `kdb gen`. This is the base name of all mustache files belonging to the template. In most cases this will be the same as the template name.
- *input keyset*: This keyset contains the data with which the template will be instantiated by the code-generator.
- *parent key*: The parent key is given in the command line and defines the input keyset (the keys below the parent key).
- *output name*: The base name for the output files as given in the command line. Suffixes may be appended, if there are multiple output files.
- *parts*: Each template may consist of multiple parts. Each part corresponds to exactly one mustache file and therefore exactly one output file.
- *part suffix*: Parts are identified by there part suffix. This suffix is appended to the template base name to obtain the name of the mustache file corresponding to the part. Similarly it is appended to the output name to obtain the output file for this part. The empty string `""` is a valid part suffix, although it is only recommend for templates with a single output file.

All templates consist of two pieces. The set of mustache files (on for each part) and an accompanying C++ class to supply the mustache rendering engine with data from the input keyset.

## 452.2 Creating a new template

In this guide we will create a basic template, that generates a single file containing a simple list of all keys in our input keyset. An example output from running `kdb gen` with our to-be-developed template would be:

```
user:/sw/myapp/#0/current
```

```
user:/sw/myapp/#0/current/dir0
user:/sw/myapp/#0/current/dir0/subdir0/key0
user:/sw/myapp/#0/current/dir0/subdir0/key1
user:/sw/myapp/#0/current/dir1
```

As you can see, the result matches that of redirecting the stdout of `kdb ls` into a file.

### 452.2.1 Creating the mustache template

The first thing we need to create is a mustache template. You will need (at least) one template file for each output file. If you make use of partials (see below), you of course have to write multiple files.
All template files must be located in the `src/tools/kdb/gen/templates` folder and must be named according to the scheme `<template base name>.<part suffix>.mustache`.
For our simple example we create the file `src/tools/kdb/gen/templates/example.txt.mustache` with the following content:

```
{{# keys }}
{{{ name }}}
{{/ keys }}
```

Note: we will not go into detail on how mustache templates work, for more information see e.g. here. All features supported by the kainjow library should be supported by our framework as well.
Our CMake script will collect all `.mustache` files in `src/tools/kdb/gen/templates` into a header containing a `static const char *` field for each file and a `std::unordered_map` containing references to all the fields. The naming scheme is needed so that the other C++ code can access the files contents via the map. This approach was chosen to allow executing the code-generator without first running the install script.
When you create a new `.mustache` file (either for a new part or a new partial), you need to invoke `cmake` again, so that all the files are collected.

### 452.2.2 Creating the supporting class

Since we need to some way of supplying data to our template, we have to create a subclass of `GenTemplate`.
First we create a new directory for our template class in `src/tools/kdb/gen`, for our template it should be `src/tools/kdb/gen/example`. In this directory we then create `example.cpp` and `example.hpp`. If your template class becomes sufficiently complex, it may make sense to split the code into multiple classes and into multiple files, for this reason we recommend creating a new directory for each template. The CMake script will also automatically recognize your files, if you put them directly into `src/tools/kdb/gen`, but using additional subdirectories (beyond the one matching your template name) like `src/tools/kdb/gen/example/src` would require modifying `src/tools/kdb/CMakeLists.txt`.
In `example.cpp` and `example.hpp` we create our subclass of `GenTemplate`. Therefore `example.hpp` should look like this:

```
#ifndef ELEKTRA_EXAMPLE_HPP
#define ELEKTRA_EXAMPLE_HPP
#include <gen/template.hpp>
class ExampleGenTemplate : public GenTemplate
{
public:
    ExampleGenTemplate () : GenTemplate ("example", { ".txt" }, {}, {})
    {
    }
protected:
    kainjow::mustache::data getTemplateData (const std::string & outputName, const std::string & part, const
       kdb::KeySet & ks,
                                             const std::string & parentKey) const override;
};
#endif // ELEKTRA_EXAMPLE_HPP
```

Apart from the line `ExampleGenTemplate () : GenTemplate ("example", { ".txt" }, {}, {})` everything should be more or less the same for all templates. Let's dissect this line:

- `ExampleGenTemplate ()` we declare a zero argument constructor. All templates *must* have only a single zero argument constructor, otherwise the framework cannot instantiate them.
- `: GenTemplate (` the constructor must invoke the base-class constructor.
- `"example",` we decided to use `example` as the base name for our template files. This base name will be replaced by the `outputName` chosen by the user, when invoking the code-generator.
- `{ ".txt" },` this is the list of part suffixes for our template. We only have a single part (output file) with the suffix `.txt`.

- `{}`, the first empty list would contain all the partials our template uses. We don't use any.
- `{}`, the second empty list, is actually a map. It would contain all parameters and whether they are required or not. We don't have parameters.

As you can see, the header is quite simple. The more important part is actually the source file. It contains the implementation of `getTemplateData`, the function that delivers all the template data to the framework.
For our example we will use this implementation:

```
#include "example.hpp"
kainjow::mustache::data ExampleGenTemplate::getTemplateData (const std::string & outputName, const
       std::string & part,
                                                                const kdb::KeySet & ks, const std::string &
       parentKey) const
{
    using namespace kdb;
    using namespace kainjow::mustache;
    list keyList;
    for (auto it = ks.begin (); it != ks.end (); ++it)
    {
        Key key = *it;
        auto keyObject = object({ { "name", key.getName() } });
        keyList.emplace_back(keyObject);
    }
    auto data = object({ { "keys", keyList } });
    return data;
}
```

The framework will invoke `getTemplateData` for each part of our template with the `outputName` and `parentKey` as given on the command line, as well as the current part suffix (`part`) and the input keyset `ks`. All keys of the input keyset are guaranteed to be below `parentKey`.
The code above simply iterates over the input KeySet and for each key creates an object `{ name: $keyName }`. All those objects are collected into a list, which is then stored under the key `keys` in the global object.

### 452.2.3 Adding the class to <tt>GenTemplateList</tt>

To make the framework aware of our class (and by extension our template), we have to then add it to `GenTemplateList`. This is simply done by adding a line to the implementation of `GenTemplateList::GenTemplateList ()` in `template.cpp`.
For our example that is:

```
addTemplate<ExampleGenTemplate> ("example");
```

We need to specify `"example"` again, because this string defines how our template shall be called, i.e. what we need to specify in the terminal to invoke it.
You also need to add the appropriate `#include` at the top of the file. In our case this is:

```
#include "example/example.hpp"
```

## 452.3 Using the new template

Now you should be able to use the new template by running (after compiling/installing Elektra again):

```
kdb gen example user userkeys
```

This should produce the file `userkeys.txt`. The file should be the same, as if we had called `kdb ls user:/ > userkeys.txt`.

## 452.4 Advanced concepts

Lastly, we will discuss a few more advanced concepts.

### 452.4.1 Switching delimiters

Switching the delimiters is fully supported and already in use in the high-level API template to allow for easier formatting.

### 452.4.2 Parameters

Sometimes you want to incorporate user input beyond the contents of the input keyset into the code-generation. For example, if you generate a web page, you may need to now where it is going to be hosted, so that you can generate

correct links. Another example would be letting the user choose the name of a generated function, when generating C code.
The first case would be required parameter. Without the domain name, we cannot generate the links. The second example meanwhile may be an optional parameter. We could just use a sensible default name for the function, since it doesn't matter much which function the user will call in their own code.
Our framework supports both required and optional parameters. To define the parameters of your template use the `parameters` argument of the `GenTemplate` constructor. This parameter is a map from strings to bools. The keys are the parameter names and the values define, whether the parameter is required (`true` means required).
Our two examples could use `{ { "domain_name", true } }` and `{ { "function_name", false } }` respectively.
To access the parameter value call one of the `getParameter` overloads. One takes a default value, the other takes a map of values and verifies that one of the given values has been chosen. For more information see the relevant code documentation. There is also `getBoolParameter` which a specialised version for boolean parameters. It accepts only `0` and `1` as values.
Calling `kdb gen` for the web page example (called `webpage` below) would then look like this:

```
kdb gen webpage <parentKey> <outputName> domain_name=somedomain.xyz
```

Since `function_name` is optional in our other example (called `ccode` below), both of the following calls are valid:

```
kdb gen ccode <parentKey> <outputName>
kdb gen ccode <parentKey> <outputName> function_name=foo
```

### 452.4.3 Using partials

The use of partials is a bit more involved than in other mustache frameworks. All the partial files for template `X` must be placed in the folder `src/tools/kdb/gen/templates/X` and must use the file extension `.mustache`. Apart from that, the filename can be chosen arbitrarily.
To use the partial named `Y` (i.e. the file `src/tools/kdb/gen/templates/X/Y.mustache`) you must use this mustache command:

```
{{> partial.enum.c }}
```

The prefix `partial.` is required by the framework, if you omit it, there will be an error.

### 452.4.4 Custom escape functions

By default, mustache escapes values for use in HTML (unless `{{{ name }}}` or `{{& name }}` is used). Since most of our templates are not HTML, the escape function can be customised. You simply have to override `GenTemplate::escapeFunction`. For an example see `HighlevelGenTemplate::escapeFunction` in `src/tools/kdb/gen/highlevel/highlevel.hpp`, it is designed for C code instead of HTML.

### 452.4.5 Dynamic list of parts

For some templates it might be necessary to switch which parts are produced based on the given parameters. This can be done by overriding `getParts()` in your template class.
To achieve a dynamic parts list, simply pass *all possible* parts in the constructor invocation. Then in your override of `getParts()` you simply inspect the given parameters and remove any parts that should not be generated.

### 452.4.6 Non-Mustache parts

It may also be useful to generate some output files without a mustache template. You could of course just write a template that renders as its string input data, but that won't work for binary files.
The proper way to achieve non-mustache-based parts is to inspect the `part` value passed to `getTemplateData()`. When you detect a non-mustache-based part you write to the file named `outputName + part` and once you are done you return `kainjow::mustache::data(false)`. This tells the render function to not invoke mustache for this part and instead continue with the next part.

# Chapter 453

# Command-line Options

## 453.1 Introduction

Many applications use command-line options and environment variables as a way to override configuration values. In Elektra this can be automated by providing a specification that maps command-line options and environment variables to keys in the KDB.
The recommended way to do this is via the `gopts` plugin. This plugin internally calls the actual parser `elektraGetOpts`. However, since there are some downsides to calling the parser manually, we cannot generally recommend doing so. If you think you have a use case for calling `elektraGetOpts` directly, take a look at the section `Advanced Use: Calling elektraGetOpts directly` below.
The parser uses a specification together with `argc`/`argv` and a list of environment variables to create keys in the `proc:/` namespace. Because the keys are in the `proc:/` namespace, they will be preferred over all other namespaces in a standard cascading lookup. This allows us to use command-line options or environment variables to override standard configuration keys.

## 453.2 Setup

While you could manually mount and configure `gopts`, it is *not recommended* doing so. Instead, you should use `elektraGOptsContract` to create a contract for use with `kdbOpen()`. This contract ensures that `gopts` is automatically mounted and correctly configured. To use `elektraGOptsContract`, include `kdbgopts.h`. This gives you access to these two functions:

```
int elektraGOptsContract (KeySet * contract, int argc, const char * const * argv, const char * const * envp,
       const Key * parentKey, KeySet * goptsConfig);
int elektraGOptsContractFromStrings (KeySet * contract, size_t argsSize, const char * args, size_t envSize,
       const char * env, const Key * parentKey, KeySet * goptsConfig);
```

Whenever possible, we recommend that you use `elektraGOptsContract` since it has less memory and processing overhead than `elektraGOptsContractFromStrings`. However, to use `elektraGOptsContract` the pointers given for `argv` and `envp` must remain valid until after calling `kdbClose()`, because the `gopts` plugin will directly use these pointers. In the standard use case (i.e. using the `argv` from `main()` and the global `environ` for `envp`), this restriction is not a problem.
An example for using `elektraGOptsContract` could look like this:

```
extern char ** environ;
int main (int argc, char ** argv)
{
  Key * parentKey = keyNew ("/sw/org/example/#0/current", KEY_END);
  KeySet * contract = ksNew (0, KS_END);
  KeySet * goptsConfig = ksNew (0, KS_END);
  // error handling omitted for brevity
  elektraGOptsContract (contract, argc, argv, environ, parentKey, goptsConfig);
  KDB * kdb = kdbOpen (contract, parentKey);
}
```

If you cannot provide pointers that meet the requirements, you may use `elektraGOptsContractFromStrings`. This function copies its arguments `args` and `env` into separate memory, so the pointers need only be valid for the duration of the function call. This is mainly useful for language bindings, since in many programming languages manual memory management is not possible and there is no (easy) way to ensure the `argv` and `envp` pointers meet the necessary requirements.
The `gopts` plugin can also use operating system specific functions to retrieve the command-line arguments and environment variables internally. We recommend that you do not rely on this behavior whenever possible, since it

can be a bit flaky (especially for command-line arguments). However, if for example you are writing a library or for some other reason do not have access to the necessary data, you can use this fallback.
If you pass `argc=0` **and** `argv=NULL` to `elektraGOptsContract` or `argsSize=0` **and** `args=NULL` to `elektraGOptsContractFromStrings`, `gopts` will fallback to the internal lookup of command-line options. Similarly, if you pass `envp=NULL` to `elektraGOptsContract` or `envSize=0` **and** `env=NULL` to `elektraGOptsContractFromStrings`, environment variables will be retrieved internally.
The other parameters are the same for both functions. The contract will be written into the `contract` KeySet. The `parentKey` indicates where to find the specification. The actual namespace that `parentKey` uses is irrelevant (we recommend a cascading key, so that it can be re-used for `kdbGet()`). The parser will use the keys below the `spec:/` namespace key equivalent to `parentKey` as the specification, and it will write keys to the equivalent key in `proc:/`.
The last parameter `goptsConfig` can be used to provide additional configuration values to `gopts`. For example, this can be used to configure the auto-generated help message. The keys that `gopts` accepts in this KeySet will be explained throughout the document . A full list can be found in the `gopts` README.

## 453.3 Specification

This section describes the specification used by the command-line option and environment variable parser.

### 453.3.1 Options

To define a command-line option either set the `opt` metakey to the short option you want to use, or set `opt/long` to the long option you want to use. For short options, only the first character of the given value will be used ('\0' is ignored). Short and long options can be used simultaneously.
Additionally, a key can also be associated with multiple short/long options. To achieve this treat `opt` as an array. For example for two options `-a` and `-b` you would set `opt=#1`, `opt/#0=a` and `opt/#1=b`. If not explicitly stated otherwise, you can replace `opt` with any `opt/#` array element in all meta-keys mentioned in this document. This of course includes long options (i.e. `opt/#0/long`, etc.).
While you can specify multiple options (or environment variables, see below) for a single key, only one of them can be used at the same time. Using two or more options (or variables) that are all linked to the same key, will result in an error.

#### 453.3.1.1 Option Arguments

Per default an option is expected to have an argument. Arguments to short and long options are given in the same way as with `getopt_long(3)` (i.e. `-oarg`, `-o arg`, `--option arg` or `--option=arg`).
To change whether an option expects an argument set `opt/arg` to either `"none"` or `"optional"` (the default is `"required"`).

- If you choose `"none"`, the corresponding key will be set to `"1"`, if the option is used. This value can be changed by setting `opt/flagvalue`.
- An option that is set to `"optional"` is treated the same as with `"none"`, except that you can also set the value with the long option form `--option=value`. This also means that `opt/flagvalue` is used, if no argument is given. Contrary to `getopt_long(3)` options with optional arguments can still have short forms. They just cannot have an argument in this form.

### 453.3.2 Environment Variables

Elektra also supports parsing environment variables in a similar manner. For these there are however, less configuration options. You can simply specify one or more environment variables for a key using the `env` metakey (or `env/#` meta-array for multiple).

### 453.3.3 Arrays

Both, options and environment variables, expose special behavior, if used in combination with arrays.
If an option is specified on a key with basename `#`, the option can be used repeatedly. All occurrences will be collected into the array.

Environment variables obviously cannot be repeated, instead a behavior similar to that used for PATH is adopted. On Windows the variable will be split at each ';' character. On all other systems ':' is used as a separator.

### 453.3.4 Parameter Arguments

All unused elements of `argv` are collected into an array. You can access this array by specifying `args=remaining` on a key with basename `#`. The array will be copied into this key. As is the case with getopt(3) processing of options will stop, if `--` is encountered in `argv`.
If we parse command-line options like the POSIX version of getopt(3) does, then we would also stop processing options at the first non-option argument. This is not the case by default, but we can enable this behavior. To do so, you need to pass a Key `/posixly` with value `1` in the `goptsConfig` KeySet of the `gopts` contract.
Additionally, there is `args=indexed`. If it is specified on a key, the key must also have the metakey `args/index=N` set to an integer `N`. Such a key will be set to the unused element at index `N`. If a key has `args=indexed` and `args/index=N`, then there must also be keys for all integers `0 <= X < N` with `args=indexed` and `args/index=N` set. For example, you cannot use `args/index=0` and `args/index=2` without `args/index=1`.
Combining `args=indexed` and `args=remaining` in the same specification (on different keys) is also possible. The key with `args=remaining` will only contain those elements not used via `args=indexed`. For example, if there are keys with `args/index=0` and `args/index=1` then the `args=remaining` array will start with the third (index 2) parameter argument. Note however, the `args=remaining` array **always** starts with index `#0`, even if it doesn't contain the first parameter argument.

#### 453.3.4.1 Example

```
[from]
args = indexed
args/index = 0
[to]
args = indexed
args/index = 1
[more/#]
args = remaining
```

If an application `app` with the specification above is called as `./app apple banana cherry date`, then the keys will be assigned as follows:

- `from = apple`
- `to = banana`
- `more/#0 = cherry`
- `more/#1 = date`

### 453.3.5 Sub-Commands

The parser also supports sub-commands. Explaining sub-commands is easiest through the help of an example: `add` and `commit` are both sub-commands of `git`, since we can call `git add` and `git commit` and they do entirely different things. The most important impact of using sub-commands is their effect on option arguments. For example calling `git -p add` and `git add -p` result in different behavior, since the `-p` option is interpreted differently. The options that `git` understands are separate from the options that its sub-command `add` knows. However, the option `-p` is understood by both. In `git` it is short for `--paginate` and in `add` it is short for `--patch`.
An important thing to know about sub-commands is that they automatically turn on POSIX mode. This means **all** options for a specific sub-command must be given before any non-option arguments (such as parameters or sub-commands). Otherwise, we couldn't distinguish between `git -p add` and `git add -p`. In other words, an option argument is always assigned to the first sub-command to its left. Any element of `argv` that is not an argument for a sub-command, either switches to a new sub-command or is the start of the parameter arguments.
A sub-command is created by specifying `command` on a key. To enable sub-command processing the parent key of the whole specification must have `command` set to an empty string. All keys marked with `command` **directly** below another key `K` marked with `command` (e.g. the parent key) are sub-commands of `K`. It is an error, if the immediate parent of a key `X` marked with `command` is not marked with `command` and `X` is not the parent of the whole specification.

To inform the application about the invoked sub-commands, the parser sets each `command` key to one of two values:

- The basename of the key, whose command was invoked.
- An empty string otherwise.

For example consider `./app add more`: The parent key will be set to the basename of whatever key `command=add` was specified on, the key for `add` will be set to the basename corresponding to `more` and the key for `more` is set to an empty string, because none of its sub-commands were invoked. A more detailed example is shown below.
Every key considered by the parser is assigned either to the root command, or a single sub-command. Specifically, each key is assigned to the command of its immediate parent. If sub-commands are used and the immediate parent of an `opt` or `args` key has no `command` metadata an error occurs. The value of a key will **only** be set, if the corresponding sub-command was invoked.
Lastly, it is allowed to have keys with `args` and `command` below the same parent. If a matching sub-command is found among the `command` keys, processing will continue there. Otherwise, the `args` keys will be considered. This allows an application to implement dynamic commands (like `git` or `kdb`) by using the `args=remaining` array to invoke another application.
If an unknown sub-command is encountered without an `args` key, an error is returned.
All of this is best understood with an example:

```
[kdb]
command = ""
[kdb/printversion]
description = "print version information and exit (ignoring all other options/commands/parameters)"
opt = v
opt/long = version
opt/arg = none
[kdb/getter]
description = "get a key's value"
command = get
[kdb/getter/verbose]
description = "print additional information about where the value comes from"
opt = v
opt/long = verbose
opt/arg = none
[kdb/getter/keyname]
description = "name of the key to read"
args = indexed
args/index = 0
[kdb/setter]
description = "set a key's value"
command = set
[kdb/setter/verbose]
description = "print additional information about where the value will be stored"
opt = v
opt/long = verbose
opt/arg = none
[kdb/setter/keyname]
description = "name of the key to write"
args = indexed
args/index = 0
[kdb/setter/value]
description = "value to be written"
args = indexed
args/index = 1
[kdb/dynamic/#]
description = "dynamically call a user-supplied command"
args = remaining
```

- If we invoke `kdb -v`, keys below `kdb/getter`, `kdb/setter` and `kdb/dynamic` are not touched. The result is:
  - `kdb = ""`
  - `kdb/printversion = 1`
  - `kdb/getter = ""`
  - `kdb/setter = ""`
- If we invoke `kdb get -v name`, keys below `kdb/setter` and `kdb/dynamic` are not touched. The result is:
  - `kdb = getter`

  - `kdb/getter = ""`
  - `kdb/getter/verbose = 1`
  - `kdb/getter/keyname = name`
  - `kdb/setter = ""`

- If we invoke `kdb -v set -v`, keys below `kdb/getter` and `kdb/dynamic` are not touched. The result is:

  - `kdb = setter`
  - `kdb/printversion = 1`
  - `kdb/setter = ""`
  - `kdb/setter/verbose = 1`
  - `kdb/getter = ""`

- If we invoke `kdb -v custom -v -x z`, keys below `kdb/getter` and `kdb/setter` are not touched. The result is:

  - `kdb = ""`
  - `kdb/printversion = 1`
  - `kdb/getter = ""`
  - `kdb/setter = ""`
  - `kdb/dynamic/#0 = custom`
  - `kdb/dynamic/#1 = -v`
  - `kdb/dynamic/#2 = -x`
  - `kdb/dynamic/#3 = z`

To determine what code to execute, an application would just start with the parent key `kdb` in the example above. It would then repeatedly look at the current key's value, append that to the current key and continue, until the current key's value was the empty string `""`. Each of the examined keys corresponds to one of the sub-commands in the invocation and the keys directly below those, contain the relevant options (and for the last sub-command also parameters).
The C code for this example is located in `examples/optsCommands.c`.

### 453.3.6 Precedence

The order of precedence is simple:

- If a short option for a key is found, it will always be used.
- If none of the short options for a key are found, we look for long options.
- If neither short nor long options are found, environment variables are considered.

### 453.3.7 Limitations

- Both options and environment variables can only be specified on a single key. If you need to have the value of one option/environment variable in multiple keys, you may use `fallback`s.
- `-` cannot be used as a short option, because it would collide with the "option end marker".
- `help` cannot be used as a long option, because it would collide with the help option.

## 453.4 Help Message

When the help option `--help` is encountered in `argv`, the parser only reads the specification, but does not create any keys in the `proc:/` namespace. It will however, generate a standard help message that you can print.
To find you, whether `--help` was encountered, check if the KeySet returned by `kdbGet()` contains the special key `proc:/elektra/gopts/help` with value `1`. If it does, the auto-generated help message is stored in the key `proc:/elektra/gopts/help/message`.

```
int main (int argc, char ** argv)
{
  // setup omitted for brevity
  KeySet * ks = ksNew (0, KS_END);
  // error handling omitted for brevity
  kdbGet (kdb, ks, parentKey);
  Key * help = ksLookupByName (ks, "proc:/elektra/gopts/help");
  if (help != NULL && strcmp (keyString (help), "1") == 0)
  {
    printf ("%s\n", keyString (ksLookupByName (ks, "proc:/elektra/gopts/help/message")));
    // cleanup omitted for brevity
    return 0;
  }
}
```

> **Note**: The key `proc:/elektra/gopts/help` will always be generated. Only if its value is set to `1`, the `--help` option was encountered.

### 453.4.1 Structure of the Help Message

The help message consists of a usage line and an options list. The program name for the usage line is taken from `argv[0]`. If the value contains a slash (`/`) it will be considered a path and only the part after the last slash will be used.
The options list will contain exactly one entry for each key that has at least one option. Each entry has two parts. First all the options for the key are listed and then (possibly on the next line, if there are a lot of options), the description for the key is listed. The description is taken from the `opt/help` or alternatively the `description` metakey.
**Note:** `opt/help` is specified *only once per key*. That means even if the key uses `opt/#0`, `opt/#1`, etc. (unlike most other metadata) the description will always be taken from `opt/help` directly, because there can only be one description. In general, we recommend using `description`, because it is used by other parts of Elektra as well. `opt/help` is intended to provide a less verbose description more suitable for the command-line.

### 453.4.2 Sub-Commands

If sub-commands are in use, the generated help message will apply to the invoked sub-command only. For example `./app --help` generates the general help message for `./app` containing only options valid for the root command. But `./app more --help` generates the help message for the sub-command `more` and contains options valid to this sub-command.

### 453.4.3 Modifying the Help Message

The standard help message can be modified in a few different ways:

- The usage line can be replaced by a custom string (see below).
- A custom string can be inserted between the usage line and the options list (see below).
- An option can be hidden from the help message by setting `opt/hidden` to `"1"`. This hides both the long and short form of the option. If you want to hide just one form, use an array of two options and hide just one index.
- If the option has an `"optional"` or `"required"` argument, the string `ARG` will be used as a placeholder by default. You can change this, by setting `opt/arg/help` for the corresponding option.

#### 453.4.3.1 Custom Usage Line

To use a custom usage line, you can either manually generate the help message or you can pass the key `/help/usage` in the `goptsConfig` KeySet. The value of `/help/usage` will be used to replace the default usage line.

Please note, that this will replace the whole usage line including the program name taken from `argv[0]`.

#### 453.4.3.2 Adding a Prefix Text

If you just want to add information above the options list, but not replace the whole usage line, you can do so by adding a prefix text. This can be done by manually generating the help message or by passing the key `/help/prefix` in the `goptsConfig` KeySet. The value of `/help/prefix` will be inserted between the usage line and the options list.

## 453.5 Examples

The following specification describes a command line interface similar to the one used by `rm`. (It is based on `rm (GNU coreutils) 8.30`).

```
[force]
opt = f
opt/long = force
opt/arg = none
description = ignore nonexistent files and arguments, never prompt
[interactive]
opt = #1
opt/#0 = i
opt/#0/long = interactive
opt/#0/arg = optional
opt/#0/flagvalue = always
opt/#0/arg/help = WHEN
opt/#1 = I
opt/#1/flagvalue = once
opt/#1/arg = none
description = prompt according to WHEN: never, once (-I), or always (-i); without WHEN, prompt always
[singlefs]
opt/long = one-file-system
opt/arg = none
description = when removing a hierarchy recursively, skip any directory that is on a file system different
       from that of the corresponding line argument
[nopreserve]
opt/long = no-preserve-root
opt/arg = none
description = do not treat '/' specially
[preserve]
opt/long = preserve-root
opt/arg = optional
opt/arg/help = all
opt/flagvalue = root
description = do not remove '/' (default); with 'all', reject any command line argument on a separate device
       from its parent
[recursive]
opt = #1
opt/#0 = r
opt/#0/long = recursive
opt/#0/arg = none
opt/#1 = R
opt/#1/arg = none
description = remove directories and their contents recursively
[emptydirs]
opt = d
opt/long = dir
opt/arg = none
description = remove empty directories
[verbose]
opt = v
opt/long = verbose
opt/arg = none
env = VERBOSE
description = explain what is being done
[showversion]
opt/long = version
opt/arg = none
description = output version information and exit
[files/#]
args = remaining
env = FILES
description = the files that shall be deleted
```

If this specification is used in a program called `erm` (for Elektra rm), which is called like this:

```
FILES="one.txt:other.log" VERBOSE=1 erm -fi --recursive
```

The following keys will be created (assuming the specification is mounted at `spec:/sw/org/erm/#0/current`)↩
:

- `proc:/sw/org/erm/#0/current/force = "1"`

- proc:/sw/org/erm/#0/current/interactive = "always"
- proc:/sw/org/erm/#0/current/recursive = "1"
- proc:/sw/org/erm/#0/current/verbose = "1"
- proc:/sw/org/erm/#0/current/files [array] = "#1"
- proc:/sw/org/erm/#0/current/files/#0 = "one.txt"
- proc:/sw/org/erm/#0/current/files/#1 = "other.log"

Calling FILES="abcd.txt" erm 123.txt 456.txt meanwhile will result in:

- proc:/sw/org/erm/#0/current/files [array] = "#1"
- proc:/sw/org/erm/#0/current/files/#0 = "123.txt"
- proc:/sw/org/erm/#0/current/files/#1 = "456.txt"

NOTE: proc:/sw/org/erm/#0/current/files [array] = "#1" means the array metadata of proc:/sw/org/erm/#0/current/files is #1.
You can find a full working example here. However, it uses a hard coded specification which is a bit harder to read.

## 453.6 Advanced Use: Calling <tt>elektraGetOpts</tt> directly

The actual command line parser is implemented in elektraGetOpts.

```
int elektraGetOpts (KeySet * ks, int argc, const char ** argv, const char ** envp, Key * parentKey);
```

To access this function, you first need to link your application against libelektra-opts and then include the header kdbopts.h.

> **Note:** libelektra-opts is an internal library and kdbopts.h an internal header. We do not make and guarantees to the API stability of the functions declared in kdbopts.h. We will do our best to keep the API compatible, but the only way to have guaranteed API stability and backwards compatibility is to only use the parser via gopts.

Calling elektraGetOpts directly has some disadvantages. The main one being that there is no way to validate the values of command-line options. When you use gopts, you can for example add the metadata type=long to a key with a command-line option specification and the type plugin will validate that the value generated by the command-line option parser is actually of type long. But since elektraGetOpts has no way of delegating to plugins (as it is independent of kdbGet), you need to do all validation manually.
So why would you want to call elektraGetOpts directly? The advantage of calling elektraGetOpts directly is that you have more control over where the specification comes from. With gopts we need to mount the specification before starting our application (at least before calling kdbOpen). But if we call elektraGetOpts directly, we can pass whatever KeySet we want. This can be useful, if you are writing a custom configuration tool (like the standard kdb tool, but specific to your use case). Such tools normally don't need (or want) to be configured via persistent config files, but often have an advanced command-line interface. Using the fairly feature-rich parser implemented in elektraGetOpts could be a good option here.

## 453.7 Advanced Use: Manually Generating the Help Message

When using gopts you automatically get a generated help message via the key proc:/elektra/gopts/help/message, whenever the --help option is used. But if you want to show the help message in another case, or you cannot use the auto-generated message for some reason (e.g. you want to add a prefix text, but the content of this text is not known before calling kdbOpen()), then you need to generate the help message manually.
If you are using elektraGetOpts directly, manually generating the help message is the only option.
The help message can be generated with elektraGetOptsHelpMessage.

```
char * elektraGetOptsHelpMessage (Key * helpKey, const char * usage, const char * prefix);
```

To access this function, you first need to link your application against libelektra-opts and then include the header kdbopts.h.

> **Note:** libelektra-opts is an internal library and kdbopts.h an internal header. We do not make and guarantees to the API stability of the functions declared in kdbopts.h. We will do our best to keep the API compatible, but the only way to have guaranteed API stability and backwards compatibility is to only use the parser via gopts.

Calling elektraGetOptsHelpMessage allocates a new string that will contain the generated help message. You need to free the string with elektraFree, once you are done with it.
The parameter helpKey has to be the same key as passed to elektraGetOpts as parentKey, if you called elektraGetOpts directly. If you used gopts, you should pass the key proc:/elektra/gopts/help.
The parameter usage is used to replace the default usage line, if it is not NULL. The string prefix is inserted between the usage line and the options list, if it is not NULL.
With gopts this could look something like this:

```
int main (int argc, char ** argv)
{
  // setup omitted for brevity
  kdbGet (kdb, ks, parentKey);
  if (/* custom condition */)
  {
    Key * help = ksLookupByName (ks, "proc:/elektra/gopts/help", 0);
    char * helpMessage = elektraGetOptsHelpMessage (help, "custom usage", "custom prefix");
    printf ("%s\n", helpMessage);
    elektraFree (helpMessage);
  }
}
```

# Chapter 454

# Compilation Variants

To create different variants of the same feature, but avoid code duplications within plugins, you have multiple options:

- Define a needs clause in a `contract` and reuse another plugin as it is. This should be preferred for filter and validation tasks.
- Have common code together in a helper library (or core library), see the CMake function `add_lib` for creating such a library (in the folder libs). This should be used for rather common functionality that might be useful for many plugins or even applications.
- Have configuration for plugins (See `elektraPluginGetConfig()` and dynamically switch with `if` according to the configuration. This should be preferred when you want to (de)activate some features of a plugin at run-time.
- Or use compilation variants to compile the plugin code multiple times with different `COMPILE_↩ DEFINITIONS` (that are Macro definitions). This should be preferred when different macro definitions lead to different plugins. It should especially be used when the resulting plugins have different dependencies: it is possible to have different `LINK_LIBRARIES`.

The advantage of compilation variants are:

- No run-time overhead
- Can be used during bootstrapping (when no configuration is available)
- Different compilation variants can be built at once (no recompilation with different CMake flags required)
- Different compilation variants can have different dependencies
- Different compilation variants can be mounted without `#refnames`

## 454.1 How to use It

To use compilation variants, add your plugin in the CMake Cache Variable `PLUGINS` multiple times. Then there can be an arbitrary number of variants. As naming convention you should have a base name with an additional variant appended with underscore, e.g.:

```
myplugin_varianta;myplugin_variantb
```

In the CMakeLists.txt of your plugin, you have two options. Option (A): When you can easily enlist every variant you simply list all plugins one after the other (*outside of* `if (DEPENDENCY_PHASE)`):

```
add_plugin(myplugin_varianta
        SOURCES      <your sources for varianta here..>
        COMPILE_DEFINITIONS   VARIANTA ELEKTRA_VARIANT=varianta
        LINK_LIBRARIES <libraries for varianta>
        )
add_plugin(myplugin_variantb
        SOURCES      <your sources for variantb here..>
        COMPILE_DEFINITIONS   VARIANTB  ELEKTRA_VARIANT=variantb
        LINK_LIBRARIES <libraries for variantb>
        )
```

Option (B): If you cannot enlist every possible compilation variant, you can iterate over all PLUGINS and check which names are requested. Then you create a plugin for every name that matches:

```
foreach (plugin ${PLUGINS})
        if (${plugin} MATCHES "myplugin_.*")
                # somehow process the variant names and include
                # or change sources and compile definitions
                # based on that.
                add_plugin(${plugin}
                SOURCES      <your sources here..>
                COMPILE_DEFINITIONS   <definitions here..>
                        ELEKTRA_VARIANT=${plugin without prefix}
        LINK_LIBRARIES <libraries for variantb>
        if (${plugin} MATCHES "ALL")
                # handle categories of plugins
                add_plugin(myplugin_all1, ...)
                add_plugin(myplugin_all2, ...)
```

For the categories such as `ALL`, however, you need to automatically append (using `add_plugin`) a useful set of plugins.
Note that every plugin needs to have `ELEKTRA_VARIANT` differently set in `COMPILE_DEFINITIONS`, otherwise you will get a linker error that `libelektra_<pluginname>_LTX_elektraPluginSymbol` has multiple definitions.
Now every public function of the plugin conflicts with itself. To avoid that, you can use:

- static functions, but they are only visible within one file. This should be preferred, when possible.
- use helper libraries using `add_lib` to share code between compilation variants (only if code is also potentially useful for other plugins/applications)
- Get a unique name for every variant using the macro `ELEKTRA_PLUGIN_FUNCTION(myplugin, open)` where myplugin is the name of the plugin and the second argument is how the function should be called.
- Including a readme for every variant (with `#ifdef` for different text) using the macro `#include ELEKTRA_README(myplugin)`

As a summary, you can have many plugins build out of the same source. Using `pluginname_variantnames` many plugins will be compiled, each with other `SOURCES` or `COMPILE_DEFINITIONS` and even `LINK_↩ LIBRARIES`: If you, e.g. just set the variants name as macro you can use

```
#ifdef varianta
#endif
```

within the code and can have two plugins: one (called `myplugin_varianta`) compiled included the `#ifdef` the other (base variant called `myplugin`) without.
Currently compilation variants are used in `the resolver plugin`.

# Chapter 455

# Introduction

In this tutorial, we will go through the steps necessary to contribute to Elektra to make it easier for you to get started. We will use a Unix based OS like Linux and CLion for development, but depending on where you want to contribute code, other IDEs will also be sufficient. Before you start, please read this to get familiar with the process of contributing to Elektra.

## 455.1 Prerequisites

- `Git` or similar software
- `CLion` or similar software
- `GitHub account`

## 455.2 Forking the Repository

Libelektra is hosted on GitHub. You can find its repository here:

- `https://github.com/ElektraInitiative/libelektra`

To be able to make pull requests, you need a copy of this repository inside your GitHub account. You can find a tutorial about how to do this `here` (remember to sign in to your account first). After this, you should see this copy in the list of your own repositories with a hint of its origin.

## 455.3 Getting the Code

To develop for libelektra, we now have to "download" your copy of its original repository. In Git this process is called "cloning". CLion has built-in Git support which we will use for this tutorial. Once you have opened CLion click on the button *Get from VCS* in the welcome-window.


> Hint for WSL-Users: Cloning the repository into the WSL filesystem will speed up the compilation time!
> i.e. clone into `~/libelektra` and not into `/mnt/c/...`

Now you should log in to your GitHub account from the IDE. Click on *GitHub* on the left and then on *Log In via GitHub...*. A browser window should open where you can enter your GitHub credentials. After successful authentication, the browser window closes and the GitHub information is available within CLion.
A list of repositories should be displayed. Select the entry *libelektra* to use your forked repository. At the bottom of the window, you can select a folder where you want to store your local copy and then click the button *Clone*.

Alternatively you can also clone your repository using the command line. Open a terminal and navigate to the folder you want to save the source code into and type:

```
git clone https://github.com/<yourGitHubUserName>/libelektra.git
```

With the project now locally available we can start developing.

## 455.4 Setting Up the Project

If you run in WSL, see WSL Setup

To import all the project configurations, right-click on the file *CMakeList.txt* in the root directory of the repository and select *Load CMake Project*, then click on *Trust Project*.

If the entry is not visible in the context menu, you can try to repair the IDE via *File --> Repair IDE...*
If you've cloned the project using a terminal, start CLion and once you see the main menu, click *Open* and select the *CMakeLists.txt* file inside the project's root directory. This will import the project accordingly and populate you run configuration with some predefined values. In rare occurrences this won't happen. If that is the case for you, simply restart CLion using:
*File --> Invalidate Caches...*
Now after all processes of CLion have finished, the project should be set up and the run configuration should be populated with entries.
Since some *kdb*-operations require root access and it is not recommended to start programs (like CLion) with root access if they usually don't require it, we have to change our CMake configurations to get rid of this requirement. To do that, open:
*File --> Settings --> Build, Execution, Deployment --> CMake*
There you can edit your CMake profiles. To get rid of the root requirement we'll add the following CMake options to our "Debug" profile:

```
-DKDB_DB_HOME="~/.config/kdb/[xyz]/home"
-DKDB_DB_SYSTEM="~/.config/kdb/[xyz]/system"
-DKDB_DB_SPEC="~/.config/kdb/[xyz]/spec"
-DKDB_DB_USER=".config/kdb/[xyz]/user"
-DCMAKE_INSTALL_PREFIX="install"
```

where "[xyz]" can be replaced by any unique identifier so that different profiles won't clash with each other. This configuration also isolates your build of Elektra from any existing Elektra installation on your system. Note the missing `~/` from the argument to `-DKDB_DB_USER`, as libelektra internally already adds the home directory path. An additional `~/` would lead to a folder named `~` in your home directory. For debugging purposes we also recommend adding the following CMake options for debug builds to enable further logging and checks:

```
-DENABLE_DEBUG="ON"
-DENABLE_LOGGER="ON"
```

The following options can be added to build additional bindings, tools and plugins:

```
-DBINDINGS="ALL;-DEPRECATED"
-DPLUGINS="ALL;-DEPRECATED"
-DTOOLS="ALL"
```

Another interesting option is to treat all warnings as errors:

```
-DCOMMON_FLAGS="-Werror"
```

The final configuration should look like this:

To increase the build speed you can also change the *Build options* to e.g.

```
-j 12
```

which, in this case, starts 12 build jobs in parallel. For optimal performance this value should represent the number of available cores of your CPU + few extra jobs.
Take care to enter the parameter in the correct format, prefixed with "--" as shown here:


It remains to be noted that CLion maintains all CMake profiles in parallel. If some CMake file changes, CLion executes `cmake` for each profile which can put a lot of strain on your system.
Finally check if `ClangFormat` is enabled for the project to automatically adhere to the formatting guidelines of the project when formatting the code. You can find the settings here:
*File --> Settings... --> Editor --> Code Style*
Make sure the selected *Scheme* is *Project*.

### 455.4.1 WSL Setup

At this point we assume you have cloned the repository into the WSL filesystem, as stated earlier.
Now we have to make sure CLion will use the WSL compiler executables or WSL toolchain.

1. Press `Ctrl + Alt + S` or go to *File --> Settings... --> Build, Execution, Deployment --> Toolchains*
2. If not already present, add the WSL Toolchain

(a) Click on the + in the top left corner

(b) Select WSL

(c) Wait for CLion to detect all executables

3. Move the WSL toolchain to the top using the up-arrow in the toolbar or `Alt + Up` to set it as default
4. Click `OK`

The WSL toolchain is now configured as the default. You can reload the CMake project by right-clicking on the root `CMakeLists.txt` and selecting `Load CMake Project` or `Reload CMake Project`
If you need further help with setting up CLion and WSL, visit the `official Tutorial`.

## 455.5 Development

Usually the folders you have to work in to add functionality or documentation are as follows:

- **doc**
  This folder contains mainly all documentation of the project, including almost all pages of the `homepage` of this project. One important note is, that all Markdown-pages can also be used for testing using `Markdown to Shell Recorder` (you can find an example on how to do this here).
- **src**
  Almost all functionality-code resides here.
  - **bindings**
    Here is all the code of available Bindings of libelektra.
  - **plugins**
    You can find all developed Plugins here. If you want to fix a bug for an existing plugin or add another one, this is the directory you have to work in. For further information on how to develop your own plugin, please visit this tutorial.
  - **tools**
    In this folder the source of all tools for interacting with Elektra's global key database, e.g. *kdb*, can be found.
- **tests**
  Here you can find all sorts of tests (excluding tests created with the *Markdown to Shell Recorder*-tool).

## 455.6 Testing

The most thorough way to test your changes is to run all tests. Therefore, navigate to your run-configurations (*Run --> Edit Configurations...*) and look for the entry *CTest Application --> All CTest*. The configuration should look like this:


You can easily run the tests by clicking on the icon right to the selected configuration:



Now you can execute this run configuration, which will run all enabled tests. Alternatively you can also run all tests using the terminal by executing `make run_all` inside your build folder (e.g. /cmake-build-debug).
You can also run other specific tests by setting `Executable` to any of the `testmod_*` or `testkdb_*`targets. Additionally, all tests using *Google Test* (e.g. tests/kdb/*) can be run directly using CLion by opening their source code and clicking on the green icon next to the class name.
If you want to test various *kdb* methods separately, you can create your own run configurations. Add a new one by clicking on the "+"-sign on the top left of the "Edit Configurations..." dialog and name it. Here *Target* should be `all` and *Executable* should have "kdb" selected. If you for example want to test `kdb plugin-info dump`, write "plugin-info dump" next to *Program arguments*. That's it, now you can just test this part of *kdb*.

For further information please read this.
Another option to easily run all tests is via Docker. A tutorial about how to do this and with further information is available here. This is also recommended before creating a pull-request. You get feedback promptly and reduce load on the CI build servers.

## 455.7 Committing Your Changes

Once you are satisfied with your changes, you have to commit them to your forked repository. By convention, such commits shouldn't be too large, otherwise it will become difficult to revert some small changes if they are not working as intended. If you use the terminal for your Git operations, to be able to commit code to your repository, you have to configure your local Git installation to use your GitHub credentials. You can find information about how to do this here. After you've set up your Git configuration, you can continue with uploading your changes.
By default, you are in the *master* branch of your repository. First, you should never directly work on the *master* branch, since only working code is expected to be there. This means, we now create a branch in our repository where our code changes will be published into. On the bottom right of your CLion window you can find the button *Git: <branchname>*. Click it and select *+ New Branch*. Type in the name of your new branch (e.g. "testbranch") and keep *Checkout branch* checked to automatically switch to it as your working branch.

Alternatively open a terminal, navigate to the root directory of your local code and type:
```
git branch testbranch
git checkout testbranch
```
After you have changed some files, it is time to publish them to your repository. To do that, select:
*Git --> Commit...*
In the dialog you have opened you can now select the files you want to include in your commit. When you double-click on a file, you can view the changes that are going to be committed. Make sure to write a meaningful commit message. The first line should have the following syntax:
```
module: short statement
```

If you fixed a bug in `kdb cp` the first line of your commit message could be `KDB: Fixed cp not copying value`. Your commit message should also include a reference to the issue you have fixed so that the issue can be closed automatically once your code change gets included to the official repository (e.g. `Closes #1234`). Before committing your changes please make sure that *Reformat code* and *Rearrange code* are disabled in the commit dialog. Otherwise, Clions formatter might produce files that don't adhere to our formatting guidelines.

If you installed the `pre-commit-check-formatting` pre-commit-hook from the `scripts` directory ensure that *Run Git hooks* is enabled in the commit dialog.
Alternatively, you can run the formatting and fix-spelling scripts inside Docker. Further information about this option can be found here.
Finally you can commit your changes by clicking the *Commit* button and navigate to:
*Git --> Push*
To do that in one step, you can also click on the button *Commit and Push...* next to the *Commit* button.
Using the terminal you first have to add all files you've changed and also want in your commit to the *stage*. Suppose we've changed how `kdb cp` works, we now have to add it to our files we want to commit (you can add several files for a commit too):
```
git add ./src/tools/kdb/cp.cpp
```
Now we've staged our modified file for our commit. The final step is to actually commit it to your online repository. Therefore, type:
```
git commit -m "<commit message>"
git push origin testbranch
```
With this, you've published your changes to your remote branch of your repository. The next step is merging this change into the original repository.

## 455.8 Creating a Pull Request

This step is most easily done using a browser. Open the web page of the Git repository of libelektra ( https://github.com/ElektraInitiative/libelektra) and log in to your account if you have not already done so.


Navigate to "Pull requests", there you can find a green button called "New pull request".

By clicking it ( `shortcut`), you can now create a pull request referencing your forked repository and the branch,the modified code resides in. Click on "compare across forks" so that you can find and select your branch.

Choose "<username>/libelektra" as the head repository and "testbranch" as the compare-branch. Now the green button "Create pull requests" should be enabled.

By clicking it you can define the title of your pull request and write a description of the work you have done. Please read the template in this form and include the information stated there if possible. Finally, by clicking "Create pull request" you've successfully created a pull request to merge your changes into the official repository! Now maintainers of libelektra will review your code and, if everything is fine, merge your changes into the official repository. Otherwise, they'll comment on this pull request if further changes are needed. To include additional changes in this pull request, just commit new code changes to the same repository and branch you've referenced for this pull request, they will be added automatically to it. In any case check the output of the automated tests!
If you want to use CLion for creating Pull Request, please check out `this` link for further information.

## 455.9 Troubleshooting

### 455.9.1 Resolving Missing ∗.so Library Error In Debug Mode

In case you fail to run Elektra with the message like this one `Reason: of module: libelektra-plugin-resolver.so, because: libelektra-plugin-resolver.so: cannot open shared object file: No such file or directory` you can solve it by defining the LD_LIBRARY_PATH variable directly in CLion. Click on the debug configurations dropdown in the upper right corner and choose 'Edit Configurations...'. Then find 'Environmental Variables' field and add the following: LD_LIBRARY_PATH=PATH_TO_YOUR_LIB_DIRECTORY
Example:
LD_LIBRARY_PATH=/home/username/TU/libelektra/cmake-build-debug/lib
If you want to run built `kdb` outside of CLion, the recommended way is to run this script from your build directory. The script resides in you original directory with project sources.
Example:

```
. /PATH/TO/YOUR/PROJECT/scripts/dev/run_env
```

Please keep in mind it sets the variables only in the currently opened shell window/session.
Please refer to this tutorial to fix the problem permanently.

## 455.10 Hints

- Especially for not that well-versed programmers don't forget that when debugging e.g. `KeySet`, which can contain many `Key` objects, you can only see that there may be more than one key stored in this set by its *size* attribute since it's keys are referenced by a pointer. Just the first `Key` (referenced by the pointer) will be visible directly using the debugger. To analyse it's stored `Key` objects you can either add a `watch` or use the GDB debug console (next to the "variables" tab inside the debugger view).
- Sometimes CLion does not rebuild changed modules accordingly by just running the test. If you think, that the modules should have been rebuilt (e.g. if running `testmod_abc` doesn't rebuild abc), please report the issue under `https://issues.libelektra.org`, so we can fix the CMake dependencies. As temporary fix you can try to rebuild the project and restart the test.
- Please add a line of changelog to this file and add it to your pull request, otherwise some test will fail in any case.

# Chapter 456

# Contributing with Windows

## 456.1 Introduction

In this tutorial, you will learn how to set up a development environment for Elektra using Windows Subsystem for Linux (WSL) and Visual Studio 2019.
Visual Studio 2022 (VS2022) Community is free but requires a registration after a couple of days. Download it and install the Linux development with C++ workload.
Elektra development mainly happens on GNU/Linux distributions. Therefore, it makes sense to use WSL if you have Windows 10 installed on your PC. For the beginning, it does not matter if you choose WSL 1 or WSL 2. It is possible to convert them later.

## 456.2 Download and Installation

We assume you use Ubuntu as WSL distribution. Install the packages required for building from VS2022:
```
apt install g++ gdb make ninja-build rsync zip
```
Then clone the libelektra Git repository:
```
git clone git@github.com:ElektraInitiative/libelektra.git
```
and open it in VS2022.

> **Note:** This command should work both in a Windows Terminal and in a WSL Shell. If you do encounter problems (especially related to line endings), please report the issue. Running the command from a WSL Shell should always work, because that uses the Linux version of git.

It will complain that IntelliSense is out of date. In addition, the `Error List` at the bottom will show CMake errors about missing compilers. To solve this, you have to configure a Linux CMake project.

## 456.3 Configure a Linux CMake project

Microsoft has even more information about this task.
In the drop-down menu that says `x64-Debug (Default)` click `Manage Configurations...`. Add `WSL-GCC-Debug` and save the configuration.

> You can remove `x64-Debug (Default)`.

CMake generation will automatically restart. When it has completed, you can click the drop-down `Select Startup Item...` (by the green *Play* button).
A long list of targets should appear. Choose `hello (bin\hello)` and click the green *Play* button. You should now see the output `Hello world` in the Linux Console Window at the bottom.

## 456.4 Why choose VS2022

One advantage of using VS2022 is the graphical debugger. Search for `hello.c` in the Solution Explorer to the right. Create a breakpoint in the main function and run the program again using the green *Play* button.
On the bottom-left you will see the windows `Autos`, `Locals` and `Watch 1`. Click on `Locals` to inspect the `Key * k`. Right-click `keyNew` to access functions like `Peek Definition`.

## 456.5 Disadvantages of VS2022

One problem with VS2022 is that it currently doesn't support opening project that are stored in the WSL filesystem. Accessing files on the Windows filesystem via WSL (required for compiling) is much slower. This means that compiling Elektra this way may take longer than expected.
As an alternative you may want to consider using `Visual Studio Code` (VSCode). VSCode doesn't have as much graphical IDE features, but it has good C/C++ support and also includes a graphical debugger. Most importantly, developing in the WSL (including within the WSL filesystem) is explicitly supported via the `WSL Remote Extension`.

## 456.6 Choosing your WSL version

It makes sense to compare WSL 1 and WSL 2 and make an informed decision for one of them. Your setup may require adaptions, such as installing an SSH server in your Linux distribution.
Good sources of information are:

- the article `Comparing WSL 1 and WSL 2` from the official WSL documentation
- the blog post `C++ with Visual Studio and WSL2`
- the official `Linux with Visual Studio C++ documentation`

Commands to change your WSL version can be found, for example, in the blog post `WSL 2 is now available in Windows Insiders`
Like Microsoft, we generally recommend you start with WSL2 (the default). It should provide better performance and more complete support for many use cases.

## 456.7 Troubleshooting

For further information, consider the official `Linux with Visual Studio C++ documentation`.
If you choose to work with source code residing on a Windows filesystem mounted in WSL (e.g. `/mnt/c/...`), please enable the `metadata` option to prevent problems due to permissions not being persisted:

- Create or modify `/etc/wsl.conf` to contain the following section and setting:

```
[automount]
options = "metadata"
```

- Restart the WSL VM by issuing `wsl --shutdown` in your window console.

Building from sources residing on a mounted Windows file system may incur a significant performance penalty. Consider using your WSL home directory (e.g. `~/`) as your location for the Elektra sources.

# Chapter 457

# Cryptographic Methods in Elektra

Elektra can protect the following aspects of your configuration:

1. confidentiality (i.e. protection against unauthorized access), and
2. integrity (i.e. protection against unauthorized modification).

Elektra provides two plugins to achieve this protection:

1. `crypto`, and
2. `fcrypt`.

## 457.1 Prerequisites - GnuPG

For the rest of this tutorial we assume that you are somewhat familiar with GnuPG (GPG). The documentation of GnuPG can be found here.
In order to find your GPG private key(s) you can use:

```
gpg2 --list-secret-keys
```

If GPG private keys are available, you see an output, that looks similar to this:

```
sec   rsa1024 2016-08-20 [SC]
          DDEBEF9EE2DC931701338212DAF635B17F230E8D
uid           [ultimate] Elektra Unit Tests (DO NOT USE IN PRODUCTION) <unit-tests@libelektra.org>
ssb   rsa1024 2016-08-20 [E]
```

The GPG key we use in this tutorial has the ID `DDEBEF9EE2DC931701338212DAF635B17F230E8D`.
A GPG private key is mandatory for the plugins to work. If you have no GPG private key available, you can generate one by entering the following command:

```
gpg2 --generate-key
```

The `fcrypt` plugin and the `crypto` plugin support both versions (version 1 and version 2) of GPG.

## 457.2 Introduction

In this tutorial we explain the use of the `crypto` plugin and the `fcrypt` plugin by a simple example: We want to protect a password that is contained in an INI-file.
The following example demonstrates how the INI-file is mounted without encryption enabled. We create the password at `user:/tests/password` and display the contents of `test.ini`.
*Step 1:* Mount `test.ini`

```
kdb mount test.ini user:/tests ini
```

*Step 2:* Set the password at `user:/tests/password` and display the contents of `test.ini`

```
kdb set user:/tests/password 1234
kdb file user:/tests/password | xargs cat | tail -n1
#> password = 1234
```

*Step 3:* (Optional) Cleanup

```
kdb rm user:/tests/password
kdb umount user:/tests
```

As you can see the password is stored in plain text. In this tutorial we demonstrate two different approaches towards confidentiality:

1. with the `fcrypt` plugin, which encrypts the entire INI-file, and

2. with the `crypto` plugin, which allows the encryption of specific key values only.

We also show how to approach integrity with the signature features of the `fcrypt` plugin.

## 457.3 Configuration File Encryption/Decryption

The `fcrypt` plugin enables the encryption and decryption of entire configuration files, thus protecting the confidentiality of the configuration keys and values. `fcrypt` utilizes GPG for all cryptographic operations. The GPG key, which is used for encryption and decryption, is specified in the backend configuration under `encrypt/key`.

```
sudo kdb mount test.ini user:/tests fcrypt "encrypt/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D" ini
```

If the above command fails, please take a look at the ReadMe of the fcrypt plugin.

As a result the file `test.ini` is encrypted using GnuPG. `fcrypt` will call the `gpg2` or `gpg` binary as follows:

```
gpg2 -o test.ini -a -r DDEBEF9EE2DC931701338212DAF635B17F230E8D -e test.ini.tmp
```

Note that `test.ini` can not only be decrypted by Elektra, but it is also possible to decrypt it with GnuPG directly. You can try to decrypt `test.ini` with GPG:

```
gpg2 -d test.ini
```

The complete procedure looks like this:

```
kdb mount test.ini user:/tests fcrypt "encrypt/key=$(kdb gen-gpg-testkey)" ini
kdb set user:/tests/password 1234
kdb file user:/tests/password | xargs cat
```

To clean up the environment we run:

```
kdb rm user:/tests/password
kdb umount user:/tests
```

## 457.4 Configuration File Signatures

`fcrypt` also offers the option to sign and verify configuration files, thus protecting the integrity of the configuration values. If `sign/key` is specified in the backend configuration, `fcrypt` will forward the key ID for signing the configuration file.

An example backend configuration is given as follows:

```
sudo kdb mount test.ini user:/tests fcrypt "sign/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D" ini
```

As a result the file `test.ini` will be signed using GPG. `fcrypt` will call the `gpg2` or `gpg` binary as follows:

```
gpg2 -o test.ini -a -u DDEBEF9EE2DC931701338212DAF635B17F230E8D -r DDEBEF9EE2DC931701338212DAF635B17F230E8D
      -s test.ini.tmp
```

If `test.ini` is modified, all following calls of `kdb get` will fail with an error message stating that the signature of the file could not be verified.

The complete example looks like this:

```
kdb mount test.ini user:/tests fcrypt "sign/key=$(kdb gen-gpg-testkey)" ini
kdb set user:/tests/password 1234
kdb file user:/tests/password | xargs cat
```

To clean up the environment we run:

```
kdb rm user:/tests/password
kdb umount user:/tests
```

### 457.4.1 Combining Signatures and Encryption

The options `sign/key` and `encrypt/key` can be combined, resulting in configuration files, that are signed and encrypted.

Mounting `test.ini` with signatures and encryption enabled can be done like this:

```
sudo kdb mount test.ini user:/tests fcrypt
      "sign/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D,encrypt/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D"
      ini
```

The complete example looks like this:

```
kdb mount test.ini user:/tests fcrypt "sign/key=$(kdb gen-gpg-testkey),encrypt/key=$(kdb gen-gpg-testkey)"
      ini
kdb set user:/tests/password 1234
kdb file user:/tests/password | xargs cat
```

To clean up the environment we run:

```
kdb rm user:/tests/password
kdb umount user:/tests
```

## 457.5 Configuration Value Encryption/Decryption

So far we learned how to encrypt and decrypt entry configuration files. Sometimes we only want to protect a smaller subset of configuration values in a bigger configuration setting. For this reason the `crypto` plugin was developed.

The crypto plugin uses libgcrypt as provider of cryptographic functions.
The crypto plugin provides the option to encrypt and decrypt single configuration values (Keys) in a Keyset. GPG is required for the key-handling.
To follow our example of an encrypted password in test.ini, we first mount the INI-file with the crypto plugin enabled, like this:

```
sudo kdb mount test.ini user:/tests crypto "crypto/key=DDEBEF9EE2DC931701338212DAF635B17F230E8D" base64 ini
```

We recommend adding the base64 plugin to the backend, because crypto will output binary data. Having binary data in configuration files is hardly ever feasible. base64 encodes all binary values within a configuration file and transforms them into Base64 strings.

### 457.5.1 Marking Keys For Encryption

To tell the crypto plugin which Keys it should process, the metakey crypto/encrypt is used. The crypto plugin searches for the metakey crypto/encrypt. If the value is equal to 1, the value of the Key will be encrypted.
We want to protect the password, that is stored under user:/test/password. So we set the metakey as follows:

```
kdb meta-set user:/tests/password crypto/encrypt 1
```

Now we are safe to set the actual password:

```
kdb set user:/tests/password "1234"
```

The resulting INI-file contains the following data:

```
#@META crypto/encrypt = 1
password=@BASE64IyFjcnlwdG8wMBEAAADwPI+lqp+X2b6BIfLdRYgwxmAhVUPurqkQVAI78Pn4OYONbei4NfykMPvx9C9w91KT
```

You can access the password as usual with kdb get:

```
kdb get user:/tests/password
```

As a result you get "1234".

### 457.5.2 Disabling Encryption

You can disable the encryption by setting crypto/encrypt to a value other than 1, for example:

```
kdb meta-set user:/tests/password crypto/encrypt 0
```

### 457.5.3 Complete Example

The complete example looks like this:

```
kdb mount test.ini user:/tests crypto "crypto/key=$(kdb gen-gpg-testkey)" base64 ini
kdb meta-set user:/tests/password crypto/encrypt 1
kdb set user:/tests/password 1234
kdb set user:/tests/unencrypted "I am not encrypted"
kdb file user:/tests/password | xargs cat
```

To disable encryption on user:/tests/password, we can run:

```
kdb meta-set user:/tests/password crypto/encrypt 0
kdb file user:/tests/password | xargs cat
```

To clean up the environment we run:

```
kdb rm user:/tests/unencrypted
kdb rm user:/tests/password
kdb umount user:/tests
```

To shut down the gpg-agent we run:

```
gpg-connect-agent --quiet KILLAGENT /bye
```

The shutdown of gpg-agent is optional.

# Chapter 458

# How to Write a Specification in Elektra for dockerd

## 458.1 Overview

### 458.1.1 Introduction

In this tutorial you will learn how to interactively use the `SpecElektra` specification language and `kdb` to write a configuration specification for `dockerd`.

### 458.1.2 What you should already know

- Already know how to write a specification

### 458.1.3 What you'll learn

- how to create and mount a specification using `kdb`
- how to add keys with different types, defaults, enums, array specifications, wildcard specifications and examples to your specification and how to validate them

### 458.1.4 What you'll do

- use `kdb` to create and mount a specification for `dockerd`
- define defaults, array / wildcard specifications, examples and checks for keys in the validation
- use the specification as a starting point for customizing the configuration of installed applications

### 458.1.5 Scope

In this tutorial we will introduce a possible specification for `dockerd`. Using `kdb` we will configure the specification.

> NOTE: As the specification for dockerd is quite big we will only present a sample for the above mentioned metakeys and link to a full specification `dockerd-full-spec`.

## 458.2 Getting Started

Before we start just an overview of the structure:

- Specification file location: `/docker/daemon.json`
- Parent specification key: `spec:/sw/dockerd/dockerd/#0/current`

## 458.3 Specification Types (values)

`Elektra` supports multiple types which leads to a more flexible specification. See type plugin for information about all the types that are supported.

## 458.4 Mount Setup

We will be mounting an existing example `dockerd-spec`.

### 458.4.1 Step 1: Mount dockerd specification

First you need to mount a specification file, in this case `dockerd.ini` to the `spec:/` namespace. You can define the path inside the `spec:/` namespace as `/sw/docker/dockerd/#0/current`, refer to `the documentation` to find out more about constructing the name.

```
sudo kdb mount "$PWD/examples/spec/dockerd.ini" spec:/sw/docker/dockerd/#0/current ni
# RET: 0
```

> NOTE: If you encounter any error saying that you have already mounted some specification with the same name you can run `sudo kdb umount spec:/sw/docker/dockerd/#0/current` and rerun the above command.
>
> Note: ni is the format which is used for the specification in the file. You can also choose to use json, then you need use yajl instead of ni.

### 458.4.2 Step 2: Define a mountpoint

Next you can define, that this specification uses a specific mountpoint for a concrete application configuration. So you can say the concrete configuration should be written to `dockerd.ini`.

```
kdb meta-set spec:/sw/dockerd/dockerd/#0/current mountpoint /docker/daemon.json
# RET: 0
```

Your `dockerd.ini` file should now contain the `mountpoint` metakey:

> NOTE: Excerpt of `cat $(kdb file spec:/sw/docker/dockerd/#0/current)`.

```
# ;Ni1
# ; Generated by the ni plugin using Elektra (see libelektra.org).
# =
# []
#  meta:/mountpoint = /dockerd/daemon.json
```

### 458.4.3 Step 3: Define <tt>json</tt> as plugin

Next we will define that our configuration should be written `json`.
We can do this by running:

```
kdb meta-set spec:/sw/dockerd/dockerd/#0/current infos/plugin "yajl"
# RET: 0
```

### 458.4.4 Step 4: Do a specification mount

```
sudo kdb spec-mount "/sw/docker/dockerd/#0/current" ni
# RET: 0
```

This specification mount makes sure that the paths where the concrete configuration should be (`daemon.json`) are ready to fulfill our specification (`dockerd.ini`). Be aware that different files get mounted for different namespaces. You've a specification file (`dockerd.ini`) for the `spec`-namespace and three files (`daemon.json`) on different locations for the `dir`- `user`- and `system`-namespaces.
You can see the files by providing the namespace as prefix to the `kdb file` command (each shows a different path):

```
kdb file system:/sw/docker/dockerd/#0/current
# /dockerd/daemon.json
kdb file user:/sw/docker/dockerd/#0/current
# STDOUT-REGEX: /dockerd/daemon.json
kdb file dir:/sw/docker/dockerd/#0/current
# STDOUT-REGEX: /dockerd/daemon.json
```

> NOTE: The $PWD should equal the PWD where you run `sudo kdb mount "$PWD/examples/spec/dockerd.↩ ini" spec:/sw/docker/dockerd/#0/current ni`.

**_Note_**: The files only exist, when configuration values are stored there, i.e. they are created on the first `kdb set` and removed with the last `kdb rm`.

For more information about namespaces in Elektra please see here, a tutorial about the topic is available here.

## 458.5 Writing specification for keys (manually)

NOTE: All output we display for `cat $(kdb file spec:/sw/docker/dockerd/#0/current)` is an excerpt of the whole file output.

In this example for `dockerd` we will be using 3 types of specifications:

- Simple specification (type and description)
- Enum specifications (for keys were only a set of possible options can be used)
- Array and wildcard specifications (for keys where a list of possible options can be used)

As the `dockerd` specification is big we will just present one of each of the above mentioned specification types.

NOTE: In Elektra we use `/` instead of `-` to seperate key names. This results in a hierarchical structure of key names. This commands will automatically store the specification in the `dockerd-daemon.ni` specification file.

### 458.5.1 Array specification

```
[data/root]
meta:/type = string
meta:/description = Root directory of persistent Docker state
meta:/default = /var/lib/docker
```

In order to get the above specification we will need the following commands:

```
kdb meta-set spec:/sw/docker/dockerd/#0/current/data/root type "string"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/data/root description "Root directory of persistent Docker
       state"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/data/root default "/var/lib/docker"
# RET: 0
```

In case no `data/root` key gets configured the value `/var/lib/docker` is used.
Let us verify that the metakeys have been set correctly:

NOTE: Excerpt of `cat $(kdb file spec:/sw/docker/dockerd/#0/current)`.

```
cat $(kdb file spec:/sw/docker/dockerd/#0/current)
# [data/root]
#  meta:/type = string
#  meta:/description = Root directory of persistent Docker state
#  meta:/default = /var/lib/docker
```

### 458.5.2 Enum specification (for keys were only a set of possible options can be used)

```
[default/cgroupns/mode]
meta:/description = Default mode for containers cgroup namespace
meta:/default = private
meta:/check/enum = #1
meta:/check/enum/#0 = host
meta:/check/enum/#1 = private
```

In order to get the above specification we will need the following commands:

```
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/cgroupns/mode description "Default mode for
       containers cgroup namespace"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/cgroupns/mode default "private"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/cgroupns/mode check/enum "#1"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/cgroupns/mode check/enum/#0 "host"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/cgroupns/mode check/enum/#1 "private"
# RET: 0
```

With this configuration we have managed to allow two possible values for `default/cgroupns/mode`. The values are `host` and `private`. The default value if we do not set any configuration is `private`.
Let us verify that the metakeys have been set correctly:

NOTE: Excerpt of `cat $(kdb file spec:/sw/docker/dockerd/#0/current)`.

```
cat $(kdb file spec:/sw/docker/dockerd/#0/current)
# [default/cgroupns/mode]
#  meta:/check/enum/#0 = host
#  meta:/check/enum/#1 = private
#  meta:/description = Default mode for containers cgroup namespace
#  meta:/default = private
#  meta:/check/enum = #1
# [data/root]
#  meta:/type = string
#  meta:/description = Root directory of persistent Docker state
#  meta:/default = /var/lib/docker
```

### 458.5.3 Wildcard specifications (for keys where a list of possible options can be used)

```
[default/ulimits/_]
meta:/type = long
meta:/description = Default ulimits for containers
meta:/example = 64000
```

For this specification we want to allow an arbitrary number of `default ulimits`. The name of the `ulimits` does not matter but all should have the same metakeys.
In order to get the above specification we will need following commands:

```
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/ulimits/_ type "long"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/ulimits/_ description "Default ulimits for
       containers"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/ulimits/_ example "64000"
# RET: 0
```

The above specification will allow us to create any name below the `default/ulimits` key. The value needs to be a string. The sample value is `64000`.
Let us verify that the metakeys have been set correctly:

NOTE: Excerpt of `cat $(kdb file spec:/sw/docker/dockerd/#0/current)`.

```
cat $(kdb file spec:/sw/docker/dockerd/#0/current)
# [default/ulimits/_]
#  meta:/type = long
#  meta:/example = 64000
#  meta:/description = Default ulimits for containers
# [default/cgroupns/mode]
#  meta:/check/enum/#0 = host
#  meta:/check/enum/#1 = private
#  meta:/description = Default mode for containers cgroup namespace
#  meta:/default = private
#  meta:/check/enum = #1
# [data/root]
#  meta:/type = string
#  meta:/description = Root directory of persistent Docker state
#  meta:/default = /var/lib/docker
```

### 458.5.4 Array specifications (for keys where a list of possible options can be used)

```
[default/address/pools]
meta:/array/min = 0
meta:/description = Default address pools for node specific local networks (list)
[default/address/pools/#/base]
meta:/type = string
meta:/description = Ip address (ipv4) + subnet
meta:/example = 172.30.0.0/16
[default/address/pools/#/size]
meta:/type = short
meta:/description = Number of ip addresses in this pool with base
meta:/example = 24
```

The specification above shows the use of an array specification with the `#` character. We define the array to have a minimum value of `0` and arbitrary max length. We use `type`, `description` and `example` as metakeys on the keys beneath each array element.
This configuration above assures that we can configure `pools` with `base` and `size`.
It prevents a configuration like:

```
default/address/pools/#0/size = "test"
```

It will fail as `default/address/pools/#/size` is required to be of type `short` when set.
In order to get the above specification we will need following commands:

```
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools array/min "0"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools description "Default address pools for
       node specific local networks (list)"
```

```
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools/#/base type "string"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools/#/base description "Ip address (ipv4)
       + subnet"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools/#/base example "172.30.0.0/16"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools/#/size type "short"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools/#/size description "Number of ip
       addresses in this pool with base"
# RET: 0
kdb meta-set spec:/sw/docker/dockerd/#0/current/default/address/pools/#/size example "24"
# RET: 0
```

The above specification defines that we can create array elements and each can have `base` or `size`.

## 458.6 Final specification code

Your specification should be complete now! After adding all the keys that are necessary for our application, you can verify that all specification keys are contained by running:

```
cat $(kdb file spec:/sw/docker/dockerd/#0/current)
```

> NOTE: We want display the output because it is too long to display (~ 400-500 lines).

## 458.7 Adding full example specification (with kdb import)

The above tutorial has given a good overview of how to write a specification. You might want to add a full example specification for `dockerd` using `kdb import`. To do so, follow the next steps.
To make sure we don't run into errors we will clean up everything we have done by now.

1. `sudo kdb rm -r spec:/sw/docker/dockerd/#0/current`
2. `sudo kdb umount spec:/sw/docker/dockerd/#0/current`
3. `rm -rf $PWD/dockerd` (make sure that you are in the same PWD as when you run the `sudo kdb mount`)

Now we are going to add an example of `dockerd-full-spec`.
Make sure you are in the root of the cloned `libelektra` repository:

1. `sudo kdb mount "$PWD/dockerd/dockerd-daemon.ni" spec:/sw/docker/dockerd/#0/current ni`
2. `kdb meta-set spec:/sw/dockerd/dockerd/#0/current mountpoint /dockerd/daemon.ni`
3. `kdb meta-set spec:/sw/dockerd/dockerd/#0/current infos/plugin "yajl"`
4. `sudo kdb spec-mount "/sw/docker/dockerd/#0/current"`
5. `sudo kdb import spec:/sw/docker/dockerd/#0/current ni < ./examples/spec/dockerd.ini`

To verify that everything was created successfully, run:

```
cat $(kdb file spec:/sw/docker/dockerd/#0/current)
```

> NOTE: We want display the output because it is too long to display (~ 400-500 lines).

## 458.8 Appendix (full specification)

The full specification can be viewed at `dockerd-full-spec`.

# Chapter 459

# How-To: kdb export

## 459.1 Introduction

The kdb tool allows users to interact with Elektra's Key Database via the command line. This tutorial explains the export function of kdb. This command lets you export Keys from the Elektra Key Database.
The command to use kdb export is:

```
kdb export [options] source [format]
```

In this command, source is the root key of which Keys should be exported. For instance, `kdb export system:/export` would export all the keys below `system:/export`. Additionally, this command exports keys under the `system:/elektra` directory by default. It does this so that information about the keys stored under this directory will be included if the Keys are later imported into an Elektra Key Database. This command exports keys to `stdout` to store them into the Elektra Key Database. Typically, the export command is used with redirection to write the Keys to a file.

### 459.1.1 Format

The format argument can be a very powerful option to use with kdb export. The format argument allows a user to specify which plugin is used to export the keys from the key database. The user can specify any storage plugin to serve as the format for the exported Keys. For instance, if a user mounted their hosts file to `system:/hosts` using `kdb mount /etc/hosts system:/hosts hosts`, they would be able to export these keys using the hosts format by using the command `kdb export system:/hosts hosts > hosts.ecf`. This command would essentially create a backup of their current `/etc/hosts` file in a valid format for `/etc/hosts`.
If no format is specified, the format `dump` will be used instead. The dump format is the standard way of expressing keys and all their relevant information. This format is intended to be used only within Elektra. The dump format is a good means of backing up Keys from the key database for use with Elektra later such as reimporting them later. As of this writing, `dump` is the only way to fully preserve all parts of the `KeySet`.

## 459.2 Options

The kdb export command takes one special option: `-E` or alternatively `--without-elektra`, which tells `kdb` to omit the `system:/elektra` directory of keys.

## 459.3 Example

```
kdb export system:/backup > backup.ecf
```

This command would export all keys stored under `system:/backup`, along with relevant keys in `system:/elektra`, into a file called `backup.ecf`.

# Chapter 460

# Hello, Elektra

This basic tutorial shows you how to compile and run a very basic Elektra application. For this tutorial we assume that you installed Elektra and `CMake` on your machine. We also assume that you work a Unix based OS like Linux or macOS.

1. Create a folder called `Hello` somewhere on your disk
2. Copy the file `examples/helloElektra.c` to the folder `Hello` you just created
3. Save a file with the following content

```
cmake_minimum_required(VERSION 3.0)
find_package(Elektra REQUIRED)
if (ELEKTRA_FOUND)
    message (STATUS "Elektra ${ELEKTRA_VERSION} found")
    include_directories (${ELEKTRA_INCLUDE_DIR})
    add_executable (hello helloElektra.c)
    target_link_libraries (hello ${ELEKTRA_LIBRARIES})
else (ELEKTRA_FOUND)
    message (FATAL_ERROR "Elektra not found")
endif (ELEKTRA_FOUND)
```

as `CMakeLists.txt` in the folder `Hello`.

1. Open a shell and change into the directory `Hello`
2. Create a build directory inside `Hello`, change into the build directory, and run Cmake:

```
mkdir build
cd build
cmake ..
```

. If everything worked until now, then CMake should print messages that look something like this:

```
-- The C compiler identification is Clang 13.0.1
-- The CXX compiler identification is Clang 13.0.1
-- Check for working C compiler: usr/bin/cc
-- Check for working C compiler: usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: usr/bin/c++
-- Check for working CXX compiler: usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Elektra 0.11.0 found
-- Configuring done
-- Generating done
-- Build files have been written to: Hello/build
```

1. Now it's time to build your application. For that step run `make` inside the folder `Hello/build`:

```
make
```

. If the last step completed successfully, then the build directory now contains the application `hello`.

1. You can now run your Elektra application by calling `./hello` inside the build directory. The output of the application should look something like this:

```
Open key database
Retrieve key set
Number of key-value pairs: 0
Add key user:/test/hello
Number of key-value pairs: 1
hello, elektra
Delete key-value pairs inside memory
Close key database
```

1. You can now change the content of `helloElektra.c`. If you want to compile and execute the updated code, then repeat steps 6 and 7.

# Chapter 461

# Bindings for the High-level API

This document describes how and when to write a language binding for the high-level API.
Writing bindings for high-level API is different from writing bindings for other parts of Elektra. This is mainly because the high-level API has different goals.

## 461.1 Goals

The goals of any high-level API (or binding to the C high-level API) for Elektra should be:

- **Type Safety:** The API should use Elektra's type system as dictated by the `type` plugin. The various types shall be mapped to native types of the target language. All API calls interacting with keys should reflect the type of the key. Type mismatches should produce errors.
- **Easy to Use:** The API should be easy to use and abstract as much of the low-level API as possible. The main part of the API should not consist of more than an initialization method, typed `get` and `set` calls and if required by the language a method freeing the acquired resources.
- **No Errors in Getters:** It is a stated goal of our high-level API to make `get` calls not able to fail. This means `get` calls *cannot* return an error under normal conditions. This can be ensured via checks during initialization or via code-generation. Both are based on the specification. Without a specification it is not possible to prevent errors because of missing keys, since we have no way of knowing which keys should exist.

  There is an exception to this rule. Some target languages have a standard error concept, which not only forces the user to handle arising errors, but also does so in a simple and concise way. An example of this is Rust. Its `Result` type forces the user to handle errors and the `?` operator allows to do this in a concise way.
  However, we still recommend avoiding errors as far as possible.
- **Idiomatic Code:** If the target language has a concept of "idiomatic code", the API should fall into that category.

## 461.2 When to write Bindings

Based on the goals above, you should decide, whether it is possible to write a binding for the C high-level API while preserving the goals.
If you decide to write a binding, proceed with this tutorial.
If not, designing an API to meet our goals is up to you. While designing the API you should keep in mind that you can always use code-generator (`kdb gen`) templates like the C API does.

## 461.3 How to write Bindings

Since the C high-level API consists of a shared library API and a code-generated part, your binding will also have these two parts.

### 461.3.1 Bindings for the <tt>highlevel</tt> Library

Writing the binding for the `highlevel` library works the same way as writing a language binding for any part of Elektra. The only additional challenge is that the binding should still meet the same goals and (as far as possible) fulfill the same guarantees as the C API.
Some languages like to split bindings into two parts. One version that maps the C API one-to-one and another part that builds on the first one. The second part then uses the languages additional features. Since our API isn't intended to be used directly, but through generated code, it might not be necessary to write the second part. It may be sufficient to write (or generate) a one-to-one mirror of the C API and use that in the code-generator template.
The generated code should still be understandable. So if writing a more idiomatic API on top of the direct mapping, significantly simplifies the generated code (and maybe also the template), you should write such an API.

### 461.3.2 Bindings for the <tt>lowlevel</tt> Library

Your programming language of choice must provide a way to call into C code (like `cgo`).
In general we prefer (in this order):

1. Automatically and statically generated bindings (like `swig`).
2. Manually written bindings that directly call to library without any statically build part (like JNI and GI). They are a bit slower but can directly access libelektra.so.
3. Manually written bindings that are built statically.

If you want to manually write a binding make sure you have a good understanding of the possible limitations the interop layer can have (e.g. variadic functions, freeing of resources, ...).
What you will also need is to set up the compiler + linker flags. For this we recommend `pkg-config`, because Elektra already provides `.pc` (and `cmake`) files.
For garbage collected (GC) languages freeing memory by hand is not something you usually do and since the GC has no knowledge of memory allocated in C we have two options:

- Forcing the user to call the appropriate functions like `keyDel()` themselves, which is very developer unfriendly and error-prone or
- Using language features like Java / C++ Destructors or Go's `runtime.SetFinalizer()` function to automatically and reliably release the memory as soon as the native objects are garbage collected.

If you decide on mapping the functionality of kdb.h 1:1 it is pretty straightforward - whereas if you want to adapt or enhance some APIs to leverage language features like iterators or operator overloading feel free to do so. The less “alien” the binding feels to its users the better.
Remember that Elektra has internal iterators (for metadata+keysets) but in general we prefer external iterators by either copying the KeySet per iterator or using `ksAtCursor`.

### 461.3.3 Variadic Functions

Some languages for example cannot call variadic functions because in C the amount of parameters has to be known at compile-time. In Go for example this is not the case since it supports variable length arguments at runtime with the `...` operator.
This is unfortunate because the low-level bindings rely heavily on variadic functions. It is possible to work around this problem by either

1. Writing helper functions in C that call these variadic functions with a fixed amount of parameters or
2. Imitating the behavior of a function e.g. `keyNew()` by calling multiple functions: `keyNew()` and `keySetMeta()` for every metakey/value that was passed.

### 461.3.4 Creating the Code-Generator Template

How to create a template for `kdb gen` is detailed in this tutorial.
The created template should support the same input keysets (and parent keys) as the `highlevel` template. If and how exactly you implement the advanced features (structs, unions, ...) is up to you. Note: enums should always be supported (if your language has them) as they are one of the `type` plugins types.

For example, in C++ the generated code could consist of nested structs with overloaded operators. In that case the structs features doesn't really make sense, since everything is already structs. Of course you could reuse some of the `gen/*` metadata to allow some cross-compatibility.

# Chapter 462

# High-level API (with code-generation)

Most applications don't need the flexibility of the low-level API (with `kdbGet`, `kdbSet`, etc.). In most cases an easy and safe way to access the configuration values is preferred. This is why we created the high-level API.
There are two different ways of using the high-level API:

1. directly
2. via code-generation

The recommended way is via code-generation (which will be explained below). If you want to use the high-level API directly take a look at its documentation. Please note, however, that certain features are only available through code-generation.

## 462.1 Overview

The code-generation API builds on Elektra's specifications. Instead of just using specifications at runtime, the code-generator parses them when invoked (ideally right before compiling) and utilizes the specification when generating configuration accessor functions.

## 462.2 Writing a specification

The process of writing a full specification for your application is beyond the scope of this guide. We will just focus on the parts that are necessary for using the code-generation API.
We will use this specification in the format of the `ni` plugin:

```
[]
mountpoint = myapp.ini
[mydouble]
type = double
default = 0.0
[myfloatarray/#]
type = float
default = 1.1
```

In Elektra a specification is defined through the metadata of keys in the `spec` namespace. The specification above contains metadata for three keys:

1. the parent key (`@`)
2. `@/mydouble`
3. `@/myfloatarray/#` (The `#` at the end of `myfloatarray/#` indicates that it is an array)

The `mountpoint` metadata on the parent key sets the name of our application's config file (the location is defined by Elektra), it should be unique.
The `type` metadata specifies the type of key. The available types can be found in the high-level API Readme under "Data Types". It is important to set the `type`, because the code-generator will ignore all keys that don't have a `type`.
Because we want our getters to be unable to fail (makes error handling trivial), we need to provide a `default` value as well. Note that `default`s for array keys like `myfloatarray/#` only work via the `spec` plugin. If you didn't mount everything correctly, you will get an error.

That's it. The code-generator just requires that each key (that you want to access) has a `type` and a `default` value.
*Note:* You can also mark keys with the `require` metadata, if there is no reasonable default value. This is only recommended as a last resort, but still preserves the guarantee that `elektraGet*` calls won't fail. If a `required` key is missing, the initialization of the `Elektra` handle will fail.

## 462.3 Invoking the code-generator

The code-generator is a very powerful and flexible tool and has many options to tweak its output. If you want to know more about how to set up everything just the way you want to, take a look at the man-pages `kdb-gen(1)` and `kdb-gen-highlevel(1)`.
To get started the basic invocation of the code-generator should be enough:

```
kdb gen -F ni=spec.ini highlevel "/sw/example/myapp/#0/current" conf
```

This tells the code-generator that your application uses the parent key `/sw/example/myapp/#0/current` and that the output files should be called `conf.*`. The argument `highlevel` just specifies which template to use and the option `-F ni=spec.ini` indicates that the file `spec.ini` (in the `ni` plugin's format) contains the specification. While the code-generator can read a specification from the KDB, we recommend you use the `-F` option. It keeps the KDB clean and can avoid troubles later on, when installing your application.

## 462.4 Using the generated code

You can now take a look at `conf.h` and `conf.c` (the files generated by the compiler). Depending on the specification you used, these files may be very long. They might also be formatted strangely, because of limitations in the code-generator. Feel free to reformat them with your tool of choice, before inspecting them.
To explain how to use the generated code, lets take a look at some of `conf.h`. For brevity's sake some parts of the file have been replaced by placeholder comments.

```
/* file header ... */
#ifndef CONF_H
#define CONF_H
#ifdef __cplusplus
extern "C" {
#endif
/* includes ... */
/* helper macros ... */
```

First there is some boilerplate, including the copyright header, include statements and some helper macros.

```
#define ELEKTRA_TAG_MYDOUBLE Mydouble
#define ELEKTRA_TAG_MYFLOATARRAY Myfloatarray
```

Next we see all the 'tag macros' used to refer to config values. These are essentially just aliases, but they allow for some flexibility in how we generate the names of the `static inline` functions further down. You should always refer to your config values via these macros, even if they are just aliases. This is because we might have to change the naming scheme for the functions, but we will try to keep the tag macros unchanged.
Additionally, the comments for these macros contain the documentation on what arguments are needed for accessing the tag in question. For example to access the elements of the array `myfloatarray/#`, we obviously need to provide an index.

```
#define elektra_len19(x) ((x) < 10000000000000000000ULL ? 19 : 20)
/* local macros ... */
#define elektra_len(x) elektra_len00 (x)
```

Then we see some local helper macros only used in this file.

```
static inline kdb_double_t ELEKTRA_GET (Mydouble) (Elektra * elektra) { /* ... */ }
static inline void ELEKTRA_SET (Mydouble) (Elektra * elektra, kdb_double_t value, ElektraError ** error) {
       /* ... */ }
static inline kdb_float_t ELEKTRA_GET (Myfloatarray) (Elektra * elektra, kdb_long_long_t index1) {  /* ...
       */ }
static inline void ELEKTRA_SET (Myfloatarray) (Elektra * elektra, kdb_float_t value, kdb_long_long_t index1,
       ElektraError ** error) { /* ... */ }
```

This is the most important part of the header. It is what makes the API work.
For each config value we generate an `ELEKTRA_GET(*)` and an `ELEKTRA_SET(*)` accessor function. All these functions are `static inline`, because they just call other getter/setter functions with partially fixed arguments. In fact many of these functions will only be a single line.

```
#undef elektra_len19
/* local macros ... */
#undef elektra_len
int loadConfiguration (Elektra ** elektra, int argc, const char * const * argv, const char * const * envp,
       ElektraError ** error);
void printHelpMessage (Elektra * elektra, const char * usage, const char * prefix);
void exitForSpecload (int argc, const char ** argv);
```

Then we undefine the local macros we defined before and declare the three initialization functions `loadConfiguration`, `printHelpMessage` and `exitForSpecload`.

```
/* elektra* macros ... */
#ifdef __cplusplus
}
#endif
#endif // CONF_H
```

At the end of the file you will find the `elektra*` convenience macros. These macros can be used to make accessing config values look more like normal function calls and avoid the ugly double parentheses in e.g. `ELEKTRA_GET(...)` `(...)`.

### 462.4.1 Obtaining an <tt>Elektra</tt> handle

We start at the bottom of our `conf.h` excerpt. `exitForSpecload` is used to initiate specload mode, if needed. This mode makes your application provide its specification to Elektra. How this works exactly is not so important (see specload plugin). You only need to know, that `exitForSpecload` should be called immediately at the start of your `main` function and that it only returns, when your application is not in specload mode.

```
int main (int argc, const char * const * argv, const char * const * envp) {
    exitForSpecload (argc, argv);
    // ...
}
```

To access your configuration, you first need to call `loadConfiguration` to get an `Elektra` handle for your application. This is done via a snippet that is more or less the same for all applications:

```
ElektraError * error = NULL;
Elektra * elektra = NULL;
int rc = loadConfiguration (&elektra, argc, argv, envp, &error);
if (rc == -1)
{
    fprintf (stderr, "An error occurred while opening Elektra: %s", elektraErrorDescription (error));
    elektraErrorReset (&error);
    exit (EXIT_FAILURE);
}
if (rc == 1)
{
    // help mode - application was called with '--help'
    // for more information see "Command line options" below
    printHelpMessage (elektra, NULL, NULL);
    elektraClose (elektra);
    exit (EXIT_SUCCESS);
}
```

Next it is recommended, you change the default handler for fatal errors. By default, we just call `exit (EXIT_FAILURE)`, since we don't know how you log your errors and what cleanup may be needed.

```
elektraFatalErrorHandler (elektra, onFatalError);
```

`onFatalError` will receive the fatal `ElektraError *`. It must at least call `elektraErrorReset` on the error and then call `exit()`.

If you want to try out the application immediately, skip down to the section about compiling. You may also have to follow the section on running your application to get everything up and running.

### 462.4.2 Reading config values

Once you have your `Elektra` instance, reading config values is easy. You just call one of the getter functions.

```
kdb_double_t mydouble = elektraGet (elektra, ELEKTRA_TAG_MYDOUBLE);
```

Here we used the convenience macro `elektraGet`. You could also invoke the `static inline` accessor function directly:

```
kdb_double_t mydouble = ELEKTRA_GET (ELEKTRA_TAG_MYDOUBLE) (elektra);
```

No error handling is required, because getter functions are designed to not fail. In a correct setup, either the initialization fails and getters are never called, or getter calls always succeed.

To access config values that don't have a static key name, like arrays, you have to supply additional arguments (and use `elektraGetV`):

```
kdb_float_t myfloat0 = elektraGetV (elektra, ELEKTRA_TAG_MYFLOATARRAY, 0);
kdb_float_t myfloat1 = elektraGetV (elektra, ELEKTRA_TAG_MYFLOATARRAY, 1);
// or
kdb_float_t myfloat0 = ELEKTRA_GET (ELEKTRA_TAG_MYFLOATARRAY) (elektra, 0);
float myfloat1 = ELEKTRA_GET (ELEKTRA_TAG_MYFLOATARRAY) (elektra, 1);
```

Of course, we also need to know how big the `myfloatarray/#` array actually is. To that end we can use `ELEKTRA_SIZE` or `elektraSize`:

```
kdb_long_long_t myfloat_size = elektraSize (elektra, ELEKTRA_TAG_MYFLOATARRAY);
// or
kdb_long_long_t myfloat_size = ELEKTRA_SIZE (ELEKTRA_TAG_MYFLOATARRAY) (elektra);
```

`ELEKTRA_SIZE` functions like their `ELEKTRA_GET` counterparts are designed to not fail.

Please note that even if you set up everything correctly, calling a getter on a non-existent, wrongly typed or otherwise inconvertible key is a fatal error. All fatal errors result in a call to the fatal error handler and therefore will exit the application.

### 462.4.3 Writing config values

Writing config values is not quite as easy as reading, but it is still quite simple:

```
ElektraError * error = NULL;
ELEKTRA_SET (ELEKTRA_TAG_MYDOUBLE) (elektra, 3.141593, &error);
if (error == NULL) {
    // handle error
    elektraErrorReset (&error);
}
```

As you can see the complexity stems from the necessary error handling. Because setting values involves IO and other uncontrollable factors, setter calls cannot be designed to not fail. This is why they accept an additional `ElektraError **` argument. It is important to call `elektraErrorReset`, if an error was set. Calling a setter with a non-null `ElektraError **` parameter is a fatal error.
Of course, you can also use `elektraSet` (error handling omitted):

```
elektraSet (elektra, ELEKTRA_TAG_MYDOUBLE, 3.141593, &error);
elektraSetV (elektra, ELEKTRA_TAG_MYFLOATARRAY, 2.718282f, &error, 2);
```

Note that `elektraSetV` takes the `ElektraError` argument before the variable arguments, while in `ELEKTA↩_SET` the error is always the last argument. This is because of limitations in the C macro system.
There is no setter for array sizes. Since Elektra's low-level part supports discontinuous arrays, we simply change the array size whenever necessary, if an array element setter is called. However, the high-level API has no support for discontinuous arrays, so take care not to create holes in your arrays, if you want to iterate over them. Remember, accessing non-existent keys (and this includes array elements) is a fatal error.

### 462.4.4 Command-line options

The generated `loadConfiguration` function automatically mounts the `gopts` plugin. This means that command-line options (as described here) are parsed and their values are set on the corresponding keys. You don't have to do anything, apart from setting the `opt` metadata. The only exception to that is the *help mode*.
When your application is called with `-h` or `--help`, we enter help mode. This is indicated by the return value `1` of `loadConfiguration`. As you can see in the example above, you should call `printHelpMessage` to print an appropriate help message to `stdout` and then close the allocated `Elektra` instance and `exit`. **Beware** an `Elektra` instance created in help mode may not be fully functional, which is why you should immediately close it once you called `printHelpMessage`.

### 462.4.5 Advanced concepts

The code-generator has some more advanced features that are supported out of the box. For example, you can read multiple config values at once by utilizing structs. The use of structs also allows for recursive configurations like menus (a menu can have submenus).
For more information take a look at the man-page `kdb-gen-highlevel(1)`.

## 462.5 Compiling your application

Once you've written your application, you will want to compile it. This requires linking some libraries and adding to your include path. The easiest way is to use CMake or pkg-config to find the needed compiler options. Examples on how set this up can be found in here and here.
The compiler invocation should look something like this:

```
cc myapp.c conf.c `pkg-config --cflags --libs elektra-codegen` -I. -o myapp -Wl,-rpath `pkg-config
      --variable=libdir elektra-codegen`
```

Note: At least C99 is required, so if your compiler defaults to an older version you'll need to add `-std=c99`.

## 462.6 Running your application

Running your application is easy, just run the executable (e.g. `myapp`). While this might work out of the box, you will just get the default configuration. To change the configuration you need to use `kdb`, which doesn't know about your

specification yet. This means you would need to set the `type` metadata and all the other stuff that your application expects by hand. For every single key. Obviously this is not the right solution.

### 462.6.1 Mounting the specification

A better solution is to inform Elektra (and `kdb`) about our specification. Then Elektra automatically copies metadata to where it should be.
First you need to mount your specification itself into the KDB. Mounting is basically the process of informing Elektra about a new part of the KDB, similar to how mounting an external hard drive informs the OS about a new part of the file system.

```
sudo kdb mount -R noresolver /etc/myapp_spec.eqd "spec:/sw/example/myapp/#0/current" specload
       app="$PWD/myapp"
```

The command above assumes that you also used the `kdb gen` command from above and that the `myapp` executable is located in `$PWD`.

> **Note:** Because of a limitation in `specload`, we have to use the `noresolver` resolver. This also means that the path to the config file (here `/etc/myapp_spec.eqd`) has to be absolute. Otherwise, it will always be relative to the current working directory in which `kdb` or your application was executed. The file *should not* exist when calling `kdb mount`. `specload` works different to other plugins. The given config file is only used, if the user makes changes to the specification via `kdb set`.

Now that Elektra knows about your specification, calling your application might work better since metadata should now be copied when you set a config value via `kdb set`. However, there won't be any type checking. For that we need to enable the `type` plugin. While this could be done manually, we can just let Elektra figure out which plugins we need and activate all of them.
This can be done with the `spec-mount` command:

```
sudo kdb spec-mount "/sw/example/myapp/#0/current"
```

Now finally your application is all setup.

### 462.6.2 Configuring your application

To configure your application you can use `kdb`:

```
kdb set "/sw/example/myapp/#0/current/mydouble" 15.4
```

If you want to set a value system-wide (not just for your user) you can use the system namespace:

```
kdb set "system:/sw/example/myapp/#0/current/mydouble" 15.4
```

Always use the cascading version of `kdb set` (i.e. the keyname begins with a slash `/`), otherwise type checking and other plugins might not be called correctly.

# Chapter 463

# How-To: kdb import

## 463.1 Introduction

The kdb tool allows users to interact with Elektra's Key Database via the command line. This tutorial explains the import function of kdb. This command lets you import Keys from the Elektra Key Database.
The command to use kdb import is:

```
kdb import [options] destination [format]
```

In this command, `destination` is where the imported Keys should be stored below. For instance, `kdb import system:/imported` would store all the keys below `system:/imported`. This command takes Keys from `stdin` to store them into the Elektra Key Database. Typically, it is used with a pipe to read in the Keys from a file.

### 463.1.1 Format

The format argument can be a very powerful option to use with kdb import. The format argument allows a user to specify which plugin is used to import the Keys into the Key Database. The user can specify any storage plugin to serve as the format for the Keys to be imported. For instance, if a user wanted to import a `/etc/hosts` file into KDB without mounting it, they could use the command `cat /etc/hosts | kdb import system:/hosts hosts`. This command would essentially copy the current hosts file into KDB, like mounting it. Unlike mounting it, changes to the Keys would not be reflected in the hosts file and vise versa.

#### 463.1.1.1 Dump Format

The dump does not rename keys by design. If a user exports a KeySet using dump using a command such as `kdb export system:/backup > backup.ecf`, they can only import that keyset back into `system:/backup` using a command like `cat backup.ecf | kdb import system:/backup`.

## 463.2 Options

The kdb import command only takes one special option `-s <name>` or alternatively `--strategy <name>`, which is used to specify a strategy.

## 463.3 Example

```
cat backup.ecf | kdb import system:/backup
```

This command would import all keys stored in the file `backup.ecf` into the Key Database under `system:/backup`.
In this example, `backup.ecf` was exported from the KeySet using the dump format by using the command:

```
kdb export system:/backup > backup.ecf
```

`backup.ecf` contains all the information about the keys below `system:/backup`:

```
cat backup.ecf
#> kdbOpen 1
#> ksNew 3
#> keyNew 19 0
#> system:/backup/key1
#> keyMeta 7 1
#> binary
```

```
#> keyEnd
#> keyNew 19 0
#> system:/backup/key2
#> keyMeta 7 1
#> binary
#> keyEnd
#> keyNew 19 0
#> system:/backup/key3
#> keyMeta 7 1
#> binary
#> keyEnd
#> ksEnd
```

Before the import command, `system:/backup` does not exists and no keys are contained there. After the import command, running the command `kdb ls system:/backup` prints:

```
system:/backup/key1
system:/backup/key2
system:/backup/key3
```

# Chapter 464

# How-To: install configuration files

The `install-config-file` tool makes using Elektra for a configuration file easier. Please refer to its man page for details on its syntax.
It is especially useful in the context of package upgrades. In this case

- `their` is the current version of the maintainer's copy of a configuration file,
- `base` is the previous version of the maintainer's copy of the configuration file.
- `our` is the user's copy of the configuration file, derived from `base`

First of all, we create a small example configuration file. To do so, we first create a temporary file and store its location in Elektra.

```
kdb set user:/tests/tempfiles/firstFile $(mktemp)
echo -e "keyA=a\nkeyB=b\nkeyC=c" > `kdb get user:/tests/tempfiles/firstFile`
```

The following call to `kdb install-config-file` will mount it at `system:/tests/installing` and additionally create a copy of it. The copy will be required for a three-way merge later on.

```
kdb install-config-file system:/tests/installing `kdb get user:/tests/tempfiles/firstFile` mini
```

We can now safely make changes to our configuration.

```
kdb set system:/tests/installing/keyB X
```

Let's assume that we've downloaded a different version of this file from the internet and placed it into the directory `/tmp/new`. At this point we can use `kdb install-config-file` to merge the changes of this downloaded version into the configuration that we currently store in Elektra.
Note the slash `/` in the beginning of the second parameter of the call to `kdb install-config-file`. This tool uses `kdb mount` and in consequence also the resolver. You have to enter the paths accordingly. Read the tutorials on mounting and namespaces if you are not sure what this means.

```
kdb set user:/tests/tempfiles/secondFile $(echo $(mktemp -d)/$(basename $(kdb get
      user:/tests/tempfiles/firstFile)))
echo -e "keyA=a\nkeyB=b\nkeyC=Y" > `kdb get user:/tests/tempfiles/secondFile`
kdb install-config-file system:/tests/installing $(kdb get user:/tests/tempfiles/secondFile) mini
```

We can check that this worked by calling

```
kdb get system:/tests/installing/keyB
#> X
kdb get system:/tests/installing/keyC
#> Y
```

Finally, we use the following commands to clean up our tutorial.

```
kdb umount system:/tests/installing
rm -rf $(kdb get user:/tests/tempfiles/firstFile)
rm -rf $(kdb get user:/tests/tempfiles/secondFile)
kdb rm -rf user:/tests/tempfiles
kdb rm -rf user:/elektra/merge/preserve
```

# Chapter 465

# install-webui

- infos/maintainer = Tomislav Makar `tmakar23@gmail.com`

## 465.1 elektra-web

*an API and web user interface to remotely manage Elektra instances*
The configuration view of elektra-web is similar to the tree view of the `qt-gui`, but with dynamic fields rendered via key metadata.

### 465.1.1 Dependencies

Elektra-web requires:

- `Elektra` with the `yajl plugin` installed
- A recent `node.js` installation (at least 6.x)
- `Go` with version > 1.13

### 465.1.2 Building with elektra-web Tool

To build Elektra with the elektra-web tool:

- Install Node.js, Go and dependencies for `yajl` plugin (see links above)
- Configure libelektra build with the elektra-web tool, e.g. `cmake .. -DTOOLS="kdb;web"`
- Build libelektra: `make`
- Install libelektra: `sudo make install`

### 465.1.3 Getting Started (docker)

- Create and run a new docker container: `docker run -d -it -p 33333:33333 -p 33334:33334 elektra/web`
- You can now access the client on: `http://localhost:33334`
- You can also build it yourself in `scripts/docker/webui/` (see `Building Docker Image`)

### 465.1.4 Running from source

- Install dependencies (see above)
- Clone libelektra repo and `cd libelektra/src/tools/web`
- Install and start an elektrad instance:

  - `cd elektrad`
  - `go build`
  - `./elektrad` (replaces `kdb run-elektrad`)

- Install and start the client (connects to the elektrad instance):
  - `cd webui`
  - `npm install`
  - `npm start` (replaces `kdb run-webd`)

- You can now access the client on: `http://localhost:33334`

### 465.1.5 Use-cases

#### 465.1.5.1 Running elektra-web on a Single Instance

If you do not want to configure multiple instances, you can set the `INSTANCE` environment variable to the server you want to configure. You can also set `user:/sw/elektra/web/#0/current/instance` to the host. Make sure to enter a full HTTP URL, e.g. `http://localhost:33333`.
If this configuration option is set, elektra-web will load the configuration page for that instance instead of the main overview page.
If you want to host elektra-web with the client and elektrad on the same instance, you must first start elektrad via `kdb run-elektrad`. Afterwards, you can run the client with:

```
export INSTANCE="http://localhost:33333" && npm start
```

It is also possible to set visibility by prefixing the host with `VISIBILITY@`.
For example (`advanced` visibility, `user` is default):

```
export INSTANCE="advanced@http://localhost:33333" && npm start
```

Now, when you open `http://localhost:33334` in your browser, the configuration page for the instance will be opened immediately.

### 465.1.6 Overview

Elektra web consists of multiple components:

- (multiple) servers running an elektra daemon (`elektrad`)
- a single server to communicate with the elektra daemons and serve the client (`webd`)
- a web browser that accesses the client (Web UI) on the `webd` server (`client`)

### 465.1.7 API

`API blueprints` are available for both APIs:

- `elektrad`, documentation: `https://elektrad.docs.apiary.io/`
- `webd`, documentation: `https://elektrawebd.docs.apiary.io/`

### 465.1.8 Test REST API on localhost

In order to test the API on localhost, you have to start an elektrad instance. This is possible in two ways:

- manually (recommended if you would like to control elektrad manually or the eletrad-web tool is not installed)

```
cd libelektra/src/tools/web
cd elektrad
go build
./elektrad
```

- by installing elektrad tool together with Elektra (see section `Building with elektra-web Tool`)

Now the server is running on http://localhost:33333. After that you can test the API with help of Postman or similar tools that allow sending REST API requests.
**Additional note**: It is recommended to install the elektrad tool rather than starting the server manually. When Elektra is installed, the kdb command together with its tools is installed globally. For instance, whenever you would like to write a shell script which has to start a REST API server, you can just add kdb run-elektrad and execute it locally.
Examples:
First create a new key-value pair user:/test and set its value to 5. This can be done

- using the command line
  ```
  kdb set user:/test 5
  ```
- through the rest API using curl
  ```
  curl -X PUT -H "Content-Type: text/plain" --data "5" http://localhost:33333/kdb/user:/test
  ```

The output of the commandline tool will be Set string to "5" if the key did not exist before. If the specified key didn't exist before, then the output will be Create a new key user:/test with string "5". Elektrad will respond with code 200.
The command
```
curl http://localhost:33333/kdb/user:/test
#> {"exists":true,"name":"test","path":"user:/test","ls":["user:/test"],"value":"5","meta":""}
```
will now return the value of the specified key user:/test, which is stored in the database.
```
{
    "exists": true,
    "name": "test",
    "path": "user:/test",
    "ls": [
        "user:/test"
    ],
    "value": "5",
    "meta": ""
}
```
The command
```
curl -X POST -H "Content-Type: application/json" -d '{"meta": [{"key": "metakey1", "value":
      "value1"},{"key": "metakey2", "value": "value2"}]}' http://localhost:33333/kdbMetaBulk/user:/test
```
will now create multiple metakeys at once. In this case, it will create two (metakey1 and metakey2).
The command
```
curl http://localhost:33333/kdb/user:/test
#>
      {"exists":true,"name":"test","path":"user:/test","ls":["user:/test"],"value":"1","meta":{"metakey1":"value1","metakey2"
```
will now also return the two metakeys.
```
{
    "exists": true,
    "name": "test",
    "path": "user:/test",
    "ls": [
        "user:/test"
    ],
    "value": "5",
    "meta": {
        "metakey1": "value1",
        "metakey2": "value2"
    }
}
```

### 465.1.9 Auth

Currently, webd does not support authentication. The best way to work around this is to use a reverse proxy (e.g. nginx reverse proxy).
Once you set up a reverse proxy on your web server, you can use it to authenticate users, e.g. by username/password auth

### 465.1.10 Code Structure

elektrad/ - contains the daemon to interact with a single elektra instance
webd/ - contains a daemon to serve the client and interact with multiple elektra instances
webui/ - contains the elektra-web client (Web UI)

- src/actions/ - Redux actions to access the KDB or display notifications in the UI
- src/components/ - React components

  - pages/ - pages in the app
    * Home.jsx - the main page (overview of all instances)
    * Configuration.jsx - configuration page (single instance)
  - TreeItem/ - contains all UI components related to a single item in the tree view
    * dialogs/ - these dialogs are opened when certain actions are pressed (icons next to the tree items)
      · AddDialog.jsx - dialog to create a new (sub-)key
      · DuplicateDialog.jsx - dialog to duplicate a key
      · EditDialog.jsx - dialog to edit a key value
      · RemoveDialog.jsx - dialog to confirm the removal of a key
      · SettingsDialog.jsx - dialog to edit metadata (new metadata can be implemented here)
      · *SubDialog.jsx - sub-dialogs of the SettingsDialog
    * fields/ - special input fields to display various values
  - App.jsx - defines app structure and routes
- src/index.js - main entry point of the app (fetches instances and renders UI)
- src/containers/ - contains components that are connected to Redux
- src/css/ - contains CSS styles
- src/reducers/ - contains Redux reducers (used to process actions)

## 465.1.11 Development Guides

### 465.1.11.1 Updating Dependencies

Lockfiles (package-lock.json) can be updated by simply deleting the current lock file and running npm install, which creates a new lock file.
Check for outdated dependencies via npm outdated. Dependencies can then be updated by running npm update.

### 465.1.11.2 Building Docker Image

Run the following command in the scripts/docker/web/ directory, replacing 1.5.0 with the latest version:
```
docker build -t elektra/web:1.5.0 -t elektra/web:latest .
```
Test the image:
```
docker run -d -it -p 33333:33333 -p 33334:33334 elektra/web:1.5.0
```
Publish it to the docker registry:
```
docker push elektra/web:1.5.0
```

### 465.1.11.3 Adding Support for New Metadata

- Create a new sub dialog by, for example, copying the NumberSubDialog.jsx file (or similar) to a new file in the webui/src/components/TreeItem/dialogs folder.
- Include the sub dialog by adding it to the SettingsDialog.jsx file in the same folder. For example, it could be added before the AdditionalMetakeysSubDialog at the end of the file:

```
+      <NewSubDialog
+        onChange={this.handleEdit('check/something')}
+        value={this.getMeta('check/something', '')}
+        saved={this.getSaved('check/something')}
+      />
       <AdditionalMetakeysSubDialog
         handleEdit={this.handleEdit.bind(this)}
         getMeta={this.getMeta.bind(this)}
         getSaved={this.getSaved.bind(this)}
         meta={this.props.meta}
         deleteMeta={this.props.deleteMeta}
       />
     </FocusTrapDialog>
```

- Mark the meta keys as handled by adding them to the HANDLED_METADATA array in webui/src/components/↩ TreeItem/dialogs/utils.js:

```
export const HANDLED_METADATA = [
  ...,
  'visibility',
  'binary',
+ 'check/something',
]
```

- Validation can then be added by handling metadata in the `webui/src/components/TreeItem/fields/validateType.js` file to the `validateType` function.
- Rendering fields in a special way when certain metakeys are present can be done by adjusting the `renderSpecialValue` function in the `webui/src/components/TreeItem/index.js` file.

# Chapter 466

# How-To: Java kdb

## 466.1 Introduction

When programming in Java it is possible to access the key database, changing values of existing keys or adding new ones and a few other things. It is also possible to write plugins for Elektra in Java but we will focus on using the Java binding in this tutorial.

## 466.2 First Steps

In order to use `kdb` you need to include the dependency in your project. Here you can find a detailed tutorial on how to do that.
After that you can start loading a `KDB` object as follows:

```
try (KDB kdb = KDB.open()) {
    // code to manipulate keys
} catch (KDBException e) {
    e.printStackTrace();
}
```

Note that `KDB` implements `AutoClosable` which allows `try-with-resouces`.
If an error occurs, detailed information can be obtained from the thrown `KDBException`.

## 466.3 A word about releasing native resources

There are 3 kinds of native resources having to be cleaned up properly to prevent memory leaks: `KeySet`, `Key` and `KDB` or rather their backing native key set, key or KDB session.
Fortunately the only resource you are strictly required to release is `KDB` via either `KDB::close` or `try-with-resouces`. For `KeySet` and `Key` the garbage collector cleans them up automatically using a `Cleaner`

## 466.4 Fetching keys

First let's retrieve a key which is already part of the key database. The first thing we need to do is to create a `KeySet` your keys are going to be stored in:

```
var keySet = KeySet.create();
```

Now we load all keys located below a specific parent key:

```
var parentKey = Key.create("user:/");
kdb.get(keySet, parentKey);
```

Note on `KDB::get`: The resulting key set may contain more keys than requested.
Now we can simply fetch the desired key's value as follows:

```
String value = keySet.lookup("user:/my/presaved/key").map(Key::getString).orElseThrow();
```

So for example if you had executed the command below via shell, before starting the application:

```
kdb set user:/my/presaved/key it_works!
```

`value` would equals `it_works!`.

## 466.5 Saving Keys

Next let's save a new key to the key database. Again, first we need to create an empty `KeySet`. We also **need to fetch** all keys for the namespace before we will be able to save a new key.

```
var keyNamespace = Key.create("user:/");                // create key representing the namespace to fetch
var keySet = kdb.get(keyNamespace);                     // fetch all keys for the namespace into a new key
    set
var keyToStore = Key.create("user:/somekey", "myValue"); // create key with value to store
keySet.append(keyToStore);
kdb.set(keySet, keyToStore);
```

If you try to save a key without fetching it beforehand, a `KDBException` will be thrown, telling you to call get before set.

The *user* namespace is accessible without special rights, but if you try to write to *system* you will need to have root privileges. Take a look at TESTING.md to see how to access the system namespace as non-root user. This should only be done in testing environments though as it is not intended for productive systems.

## 466.6 Examples

### 466.6.1 Traversing Keys in a <tt>KeySet</tt>

First we create a new `KDB` handle and fetch all keys for the desired namespace, in this example the whole `user` namespace. Since all keys are put in the passed `keySet` variable we can then iterate through it. The `at(int)` method returns a new `Key` object for the native key with the corresponding position within the `keySet`.

```
var keyNamespace = Key.create("user:/");           // select a namespace from which all keys should be
    fetched
try (KDB kdb = KDB.open()) {
    var keySet = kdb.get(keyNamespace);            // fetch all keys into a new key set
    for (int i = 0; i < keySet.length(); i++) {    // traverse the set
        var currentKey = keySet.at(i);
        System.out.println(String.format("%s: %s",
                currentKey.getName(),              // fetch the key's name
                currentKey.getString()));          // fetch the key's value
    }
} catch (KDBException e) {
    e.printStackTrace();
}
```

`KeySet` also provides an iterator implementation:

```
var keyNamespace = Key.create("user:/");           // select a namespace from which all keys should be
    fetched
try (KDB kdb = KDB.open()) {
    var iter = kdb.get(keyNamespace).iterator();   // fetch all keys into a new key set
    while(iter.hasNext()) {                        // traverse the set
        var currentKey = iter.next();
        System.out.println(String.format("%s: %s",
                currentKey.getName(),              // fetch the key's name
                currentKey.getString()));          // fetch the key's value
    }
} catch (KDBException e) {
    e.printStackTrace();
}
```

Of course, alternatively a for each loop can be used:

```
var keyNamespace = Key.create("user:/");           // select a namespace from which all keys should be
    fetched
try (KDB kdb = KDB.open()) {
    var keySet = kdb.get(keyNamespace);            // fetch all keys into a new key set
    for (var currentKey : keySet) {                // traverse the set
        System.out.println(String.format("%s: %s",
                currentKey.getName(),              // fetch the key's name
                currentKey.getString()));          // fetch the key's value
    }
} catch (KDBException e) {
    e.printStackTrace();
}
```

Another way to traverse is to use the Stream API which was introduced with Java 8:

```
var keyNamespace = Key.create("user:/");           // select a namespace from which all keys should be
    fetched
try (KDB kdb = KDB.open()) {
    var keySet = kdb.get(keyNamespace);            // fetch all keys into a new key set
    keySet.forEach(key -> System.out.printf("%s: %s%n", key.getName(), key.getString()));
                // directly format-print all key value pairs using foreach
    // or
    keySet.stream().map(key -> String.format("%s: %s", key.getName(),
       key.getString())).forEach(System.out::println);  // map the key value paris to the desired format
       first and then print them using foreach and a function reference
} catch (KDBException e) {
    e.printStackTrace();
}
```

As the `KeySet` implements the `SortedSet<Key>` interface, all its methods can be used for (not only) a traversal:

```
var keyNamespace = Key.create("user:/");           // select a namespace from which all keys should be
    fetched
try (KDB kdb = KDB.open()) {
    var keySet = kdb.get(keyNamespace);            // fetch all keys into a new key set
    keySet.subSet(Key.create("user:/b"), Key.create("user:/k"))                                  // only
    select the keys starting with "user:/a" through "user:/k" (excluded).
     .forEach(key -> System.out.printf("%s: %s%n", key.getName(), key.getString()));          // directly
    format-print all key value pairs using foreach
} catch (KDBException e) {
    e.printStackTrace();
}
```

Another way to traverse is to use the Stream API which was introduced with Java 8:

```
var keyNamespace = Key.create("user:/");           // select a namespace from which all keys should be
    fetched
try (KDB kdb = KDB.open()) {
    var keySet = kdb.get(keyNamespace);            // fetch all keys into a new key set
    keySet.forEach(key -> System.out.printf("%s: %s%n", key.getName(), key.getString()));
                // directly format-print all key value pairs using foreach
    // or
    keySet.stream().map(key -> String.format("%s: %s", key.getName(),
    key.getString())).forEach(System.out::println);  // map the key value paris to the desired format
    first and then print them using foreach and a function reference
} catch (KDBException e) {
    e.printStackTrace();
}
```

### 466.6.2 Read Multiple Keys From KDB

This example shows how to read multiple keys. It provides comments for further clarification. Further information can be found here.

## 466.7 Java Plugin Tutorial

For the tutorial on how to write java plugins, please check out this page.

# Chapter 467

# How-To: Write a Java Plugin

This file serves as a tutorial on how to get started with writing a Java plugin.

## 467.1 Basics

If you want to know more about plugins in general, please check out the How-To: Write a Plugin page. If you need a tutorial for using Key and KeySet in Java, please check out the How-To: Java kdb page.

## 467.2 Two Technologies used in Java Plugins

Before we will take a look into how to write a plugin in Java, it is important to note, that there are two technologies needed:

- `process` plugin
- JNA binding

### 467.2.1 <tt>process</tt> Plugin

The `process` plugin is a special plugin, which allows using an external application as the implementation of a plugin.
To achieve this, the `process` plugin spawns a child process for the external executable and then uses a simple protocol to relay any requested operations to this child process. The details of how this protocol works are not important for writing a Java plugin. All the details of the protocol are abstracted via the `PluginProcess` class and the `Plugin` interface.
If you do want to know the details of the `process` protocol, take a look at the README of the `process` plugin.

### 467.2.2 JNA Binding

Java Native Access is Java technology (library), which like JNI allows a developer to run native code using only Java by providing access to native shared libraries. In order to use the JNA binding, it should be installed first. You can find more information on `JNA's GitHub page`.
We provide a Java binding for Elektra's API via JNA. For more general information about the binding take a look at this document.

### 467.2.3 Writing a Plugin

To write a plugin, you need to create an implementation of the `org.libelektra.Plugin` interface. This is very similar writing any other plugin, except that you use Java instead of C.
The standard API functions of the plugin API all have corresponding methods in the `Plugin` interface:

- `int open(KeySet config, Key errorKey)` is the Java version of `elektraPluginOpen`
- `int get(KeySet keySet, Key parentKey)` is the Java version of `elektraPluginGet`

- `int set(KeySet keySet, Key parentKey)` is the Java version of `elektraPluginSet`
- `int error(KeySet keySet, Key parentKey)` is the Java version of `elektraPluginError`
- `int close(Key parentKey)` is the Java version of `elektraPluginClose`

Additionally, there is a `@Nonnull String getName()` method. This method must return the unique name of the plugin. In the C API this would be taken from the contract, but the `process` protocol requires this separately, so we need a separate method for it.
Otherwise, there are a few differences between implementing a plugin in C and in Java:

1. In C it is possible to implement only `elektraPluginGet` and leave the other functions unimplemented. Because of interface inheritance in Java, it is required to implement all 5 method `open`, `get`, `set`, `error` and `close`. Whether a method is actually supported, must be communicated via the contract.
2. In C the parent key of the contract depends on the plugins name. For example, the contract for `dump` can be found under `system:/elektra/modules/dump` and the `dump` plugin returns it as such. However, in Java the parent key for the contract is always `system:/elektra/modules/java` (you may use the constant `Plugin.PROCESS_CONTRACT_ROOT`). The keys will be transformed via the `process` protocol and plugin to match the normal expectations.
3. In C all functions a plugin exports (including `open`, `get`, `set`, `error`, `close`, but also additional ones) are registered in the contract under `system:/elektra/modules/<plugin>/exports/<function>` with a function pointer key. Because we cannot provide a C function pointer to a Java function and because `process` uses a child process for the Java code, we cannot export functions like that. This means a Java plugin cannot export additional functions. However, we must still define which functions are supported by the plugin. To this end, a Java plugin must set `system:/elektra/modules/java/exports/has/<function> = 1` (where `<function>` is one of `open`, `get`, `set`, `error`, `close`) for all supported functions.
4. Methods that are not supported, should simply be implemented as
   ```
   throw new UnsupportedOperationException();
   ```
   This is safe, because the method will not be called if the other steps are followed correctly. The exception here is the `get` method. It must still be implemented and return the contract, when the parent is the same as or below `system:/elektra/modules/java`. Otherwise, it is safe to throw `UnsupportedOperationException`.

Otherwise, the rules for return values and plugin behavior are the same as for a C plugin.
You can find a few examples for Java plugins in the folder `src/bindings/jna/plugins/`. We use separate Gradle projects for every plugin, but that is not a requirement. But creating separate JAR files for each plugin, keeps down the size of the binaries that need to be loaded to mount a plugin.

### 467.2.4 Usage of Plugin

See man page of `kdb-mount-java(1)`.

# Chapter 468

# Language Bindings

## 468.1 Introduction

In this section, we will explain the how to write a language binding for Elektra.

### 468.1.1 High-level API

Writing bindings for the high-level API is described in a different document, since it is a bit less straightforward and needs additional considerations.

## 468.2 TODO

1. which parts of Elektra bindings make sense (application, plugin, tools, ...)
2. how to integrate bindings into CMake (if possible and useful)
3. which parts of the bindings can and should differ for every language. This includes:
   (a) iterators
   (b) conversion to native types (strings, int, ...)
   (c) operator overloading (if available)
   (d) other programming language integrations (streams, hash-codes, identity, ...)
   (e) returned errors from kdb functions (what this issue here is about)

## 468.3 CMake Integration

### 468.3.1 Building

To add the subdirectory containing our binding to the build, we have to modify `src/bindings/CMakeLists.txt`.

```
check_binding_included ("our_binding" IS_INCLUDED)
if (IS_INCLUDED)
    add_subdirectory (our_binding_directory)
endif ()
```

At first we want to make sure that the build tools and compilers we need for the binding are installed. We can use `find_program (BUILD_TOOL_EXECUTABLE build_tool)` to find our `build_tool` program. The result of the search will be stored in `BUILD_TOOL_EXECUTABLE`, so now we can use an if block to include the bindings in the build, if the program exists or exclude it, if it doesn't. To do that, we use `add_binding` which adds ours to the list of bindings that will be built. For more provided functions, see here.
If, for example, our bindings only support linking against a dynamic library we can express that, by using the `BUILD_*` variables in if blocks or by passing `ONLY_SHARED` to `add_binding`. You can read more in the compile doc.

```
if (BUILD_TOOL_EXECUTABLE)
    add_binding (our_binding ONLY_SHARED)
else ()
```

```
    exclude_binding (our_binding, "build_tool not found")
    return ()
endif ()
```

Elektra uses out-of-source builds, so we have to copy all the needed files over to the build directory. The `${CMAKE_CURRENT_SOURCE_DIR}` variable refers to the source directory, while `${CMAKE_CURRENT_BINARY_DIR}` refers to the build directory. The copy is as simple as

```
# Inside the previous if block
file (COPY "${CMAKE_CURRENT_SOURCE_DIR}/" DESTINATION "${CMAKE_CURRENT_BINARY_DIR}")
```

However for some files we may want to use some CMakes variables. Say we're writing a build script for our project and want to include the version number and the directory that `libelektra.so` resides in, so our build tool can find and link against it. We create our script named `build.script.in`. It looks like this

```
version = "@KDB_VERSION@"
link-search-path = "@CMAKE_BINARY_DIR@/lib"
```

Back in our CMake script, we tell CMake to replace the variables with their associated values.

```
configure_file ("${CMAKE_CURRENT_SOURCE_DIR}/build.script.in" "${CMAKE_CURRENT_BINARY_DIR}/build.script"
       @ONLY)
```

Note how we leave off the `.in` ending on the target file.

Since we're building a foreign language project, it will most likely have its own build tool or compiler. So we have to tell CMake how to invoke it, in order to build the project. First we specify what file we expect to be generated by that command. In this example it's a `.lib` file that is generated in some target directory. We then call our build tool using the variable `BUILD_TOOL_EXECUTABLE` we created earlier with the `build` subcommand and the `--release` option. We can also specify one or multiple files that this command depends on, such that CMake can make sure they are built or generated before. Finally, we add a custom target that depends on the `.lib` file. To built this target, CMake will invoke our custom command and build the specified file.

```
add_custom_command (OUTPUT "${CMAKE_CURRENT_BINARY_DIR}/target/release/libelektra.lib"
            COMMAND ${BUILD_TOOL_EXECUTABLE} build --release
            WORKING_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}
            DEPENDS "${CMAKE_CURRENT_BINARY_DIR}/build.script"
       "${CMAKE_CURRENT_BINARY_DIR}/other-dependency.file")
add_custom_target (our_binding ALL DEPENDS "${CMAKE_CURRENT_BINARY_DIR}/target/release/libelektra.lib")
```

We can then explicitly include the bindings using `cmake -DBINDINGS="our_binding" ..` in the build directory and follow the further steps for compilation.

### 468.3.2 Testing

To invoke our tests through CMake, we have to follow similar steps as in the build. We add a test by specifying a command that runs our tests. In our case, we're calling the same program for testing as in the building step.

```
add_test (NAME test_our_binding COMMAND ${BUILD_TOOL_EXECUTABLE} test WORKING_DIRECTORY
       "${CMAKE_CURRENT_BINARY_DIR}")
```

We may have to specify an additional environment variable to tell the test command, where `libelektra.so` resides, so that the dynamic linker can find it.

```
set_property (TEST test_our_binding PROPERTY ENVIRONMENT "LD_LIBRARY_PATH=${CMAKE_BINARY_DIR}/lib")
```

Now our bindings can be tested through `ctest` alongside all other tests.

See the Java binding for examples.

## 468.4 Error Handling

Since v0.9.0, Elektra has a new error code system. You might want to take a look in the design decision first to understand the concept of the error codes. These codes are hierarchically structured and are therefore perfectly suitable for inheritance if the language supports it.

Some error codes like the `Permanent Errors` are generalizations and used for developers who want to catch all specific types of errors (e.g., it does not matter if it is a Resource or Installation Error but the developer wants to check for both). Such errors should not be able to "instantiate" or emitted back to Elektra as we want to force developers to take a more specific category. In case of Java for example the `Permanent Error` is an abstract class. Which errors are instantiable or not can be seen in the error-categorization guideline in the respective title saying either `abstract` or `concrete`. Here is an example of how Java has implemented it:

```
public abstract class PermanentException extends Exception {...}
    public class ResourceException extends PermanentException {...}
        public class MemoryAllocationException extends ResourceException {...}
    public class InstallationException extends PermanentException {...}
...
```

All error codes as well as the hierarchy itself is depicted in the design decision.

If you have a language which does not support inheritance this way like GoLang, you can still use the error code itself since the hierarchy is integrated in it. For example you can check if the code starts with `C01...` to catch all `Permanent Errors`.

### 468.4.1 Error Message

In Elektra every error has a predefined format. You can take a look at the related design decision to see how it looks like.
Every Exception/Error struct/etc. should have separate accessors to individual parts of the message. These include:

1. Module (getModule())
2. Error Code (getErrorCode())
3. Reason (getReason())
4. Configfile (getConfigFile())
5. Mountpoint (getMountpoint())
6. Debuginformation (text looks like "At: file:line") (getDebugInformation())

In case of an error at least the following part has to be returned:

```
Sorry, module getModule() issued error getErrorCode():
getReason()
```

Please also keep the wording identical for consistency. Take a look how the Java binding implemented it in the KDBException

# Chapter 469

# How-To: Logging

## 469.1 Quickstart

Enable logging by compiling with the CMake option `ENABLE_LOGGER=ON` (e.g. `cmake -DENABLE_LOGGER=ON`).
By default errors and warnings are logged to stderr and syslog, while notice and info message are only logged to syslog. Debug level message are not logged by default, but will be logged to syslog, if `ENABLE_DEBUG=ON` is set in CMake.
The file sink behaves like syslog, if it has been enabled (see below).

## 469.2 Step by Step Guide

### 469.2.1 Preparation

1. Import the logging library in your code:

```
#include <kdblogger.h>
```

1. Insert logging statements:

```
ELEKTRA_LOG("Hello, %s!", "world");
```

#### 469.2.1.1 Log Everything

If Elektra should display all log messages, then please follow the steps below.

1. Uncomment the line:

```
// #define NO_FILTER
```

in the file `src/libs/elektra/log.c`. (Alternatively you may define this macro via CMake.)

1. Optional: The logging levels are set in `src/include/kdblogger.h`. For example, if you want to see everything (including debug messages) on stderr, then change the line

```
static const int ELEKTRA_LOG_LEVEL_STDERR = ELEKTRA_LOG_LEVEL_WARNING;
```

to

```
static const int ELEKTRA_LOG_LEVEL_STDERR = ELEKTRA_LOG_LEVEL_DEBUG;
```

#### 469.2.1.2 File Specific Logging

If you want to only log messages below a specific directory prefix, then please follow the steps below.

1. Search for the code:

```
#ifndef NO_FILTER
     // XXX Filter level ...
#endif
```

in the file `src/libs/elektra/log.c`.

1. Replace the code with something like:

```
#ifndef NO_FILTER
     if (strncmp (file, "src/postfix/", sizeof ("src/postfix"))) return -1;
#endif
```

, where `src/postfix` contains all source files with logging statements that Elektra should log. For example, if you want to log everything from the `yamlcpp` plugin, then use the following code.

```
#ifndef NO_FILTER
     if (strncmp (file, "src/plugins/yamlcpp/", sizeof ("src/plugins/yamlcpp"))) return -1;
#endif
```

. To log messages from multiple source you can use the operator `&&` to chain multiple calls to `strncmp`. For example, to log messages from the `directoryvalue` and `yamlcpp` plugin use the code:

```
#ifndef NO_FILTER
     if (strncmp (file, "src/plugins/directoryvalue/", sizeof ("src/plugins/directoryvalue")) &&
         strncmp (file, "src/plugins/yamlcpp/", sizeof ("src/plugins/yamlcpp")))
             return -1;
#endif
```

### 469.2.2 Enabling and Disabling Sinks

The logging framework has 3 sinks: stderr, syslog and file.
The first to are enabled by default, while file is disabled. To enable it uncomment the line

```
// #define USE_FILE_SINK
```

in `src/libs/elektra/log.c`. The file that log messages are written to is defined in this line

```
elektraLoggerFileHandle = fopen ("elektra.log", "a");
```

### 469.2.3 Compilation

1. Enable the logger: e.g. run `cmake` with the switch `-DENABLE_LOGGER=ON`
2. Build Elektra

## 469.3 Log Levels

There are four log levels (ERROR is reserved for aborts within `ELEKTRA_ASSERT`):

- `ELEKTRA_LOG_WARNING`, something critical that should be shown to the user (e.g. API misuse), see ELEKTRA_LOG_LEVEL_WARNING
- `ELEKTRA_LOG_NOTICE`, something important developers are likely interested in, see ELEKTRA_LOG_LEVEL_NOTICE
- `ELEKTRA_LOG`, standard level gives information what the code is doing without flooding the log, see ELEKTRA_LOG_LEVEL_INFO
- `ELEKTRA_LOG_DEBUG`, for less important logs, see ELEKTRA_LOG_LEVEL_DEBUG

# Chapter 470

# How-To: kdb merge

## 470.1 Introduction

The kdb tool allows users to access and perform functions on the Elektra Key Database from the command line. We added a new command to this very useful tool, the `merge` command. This command allows a user to perform a three-way merge of KeySets from the `kdb` tool.
The command to use this tool is:

```
kdb merge [options] ourpath theirpath basepath resultpath
# RET: 7
```

The standard naming scheme for a three-way merge consists of `ours`, `theirs`, and `base`:

- `ours` refers to the local copy of a file
- `theirs` refers to a remote copy
- `base` refers to their common ancestor.

This works very similarly for KeySets, especially ones that consist of mounted configuration files.
For mounted configuration files:

- `ours` should be the user's copy
- `theirs` would be the maintainers copy,
- `base` would be the previous version of the maintainer's copy.

If the user is just trying to accomplish a three-way merge using any two arbitrary keysets that share a base, it doesn't matter which ones are defined as `ours` or `theirs` as long as they use the correct base KeySet. In `kdb merge`, `ourpath`, `theirpath`, and `basepath` work just like `ours`, `theirs`, and `base` except each one represents the root of a KeySet. The argument `resultpath` is pretty self-explanatory, it is just where you want the result of the merge to be saved under. It's worth noting, `resultpath` should be empty before attempting a merge, otherwise there can be unintended consequences.

## 470.2 Options

As for the options, there are two basic options:

- `-i`, `--interactive`: which attempts the merge in an interactive way
- `-f`, `--force`: which overwrites any Keys in `resultpath`

### 470.2.1 Strategies

Additionally, there is an option to specify a merge strategy, which is very important.
The option for strategy is:

- `-s <name>`, `--strategy <name>`: which is used to specify a strategy to use in case of a conflict

The current list of strategies are:

- `preserve`: the merge will fail if a conflict is detected
- `ours`: the merge will use our version during a conflict
- `theirs`: the merge will use their version during a conflict
- `cut`: Removes existing keys below the resultpath and replaces them with the merged keyset.
- `import`: (DEPRECATED, avoid using it!) Preserves existing keys in the resultpath if they do not exist in the merged keyset. If the key does exist in the merged keyset, it will be overwritten.

If no strategy is specified, the merge will default to the preserve strategy as to not risk making the wrong decision. If any of the other strategies are specified, when a conflict is detected, merge will use the Key specified by the strategy (`ours`, `theirs`, `cut` or `import`) for the resulting Key.

## 470.3 Basic Example

Basic Usage:

```
kdb merge system:/hosts/ours system:/hosts/theirs system:/hosts/base system:/hosts/result
```

## 470.4 Examples Using Strategies

Here are examples of the same KeySets being merged using different strategies. The KeySets are mounted using a property format, the left side of '=' is the name of the Key, the right side is its string value.
We start with the base KeySet, `system:/base`:

```
key1=1
key2=2
key3=3
key4=4
key5=5
```

Here is our KeySet, `system:/ours`:

```
key1=apple
key2=2
key3=3
key5=fish
```

Here is their KeySet, `system:/theirs`:

```
key1=1
key2=pie
key4=banana
key5=5
```

To add the keys to the key database, execute the following commands:

```
kdb set user:/tests/base/key1 1
#> Create a new key user:/tests/base/key1 with string "1"
kdb set user:/tests/base/key2 2
#> Create a new key user:/tests/base/key2 with string "2"
kdb set user:/tests/base/key3 3
#> Create a new key user:/tests/base/key3 with string "3"
kdb set user:/tests/base/key4 4
#> Create a new key user:/tests/base/key4 with string "4"
kdb set user:/tests/base/key5 5
#> Create a new key user:/tests/base/key5 with string "5"
kdb set user:/tests/ours/key1 apple
#> Create a new key user:/tests/ours/key1 with string "apple"
kdb set user:/tests/ours/key2 2
#> Create a new key user:/tests/ours/key2 with string "2"
kdb set user:/tests/ours/key3 3
#> Create a new key user:/tests/ours/key3 with string "3"
kdb set user:/tests/ours/key5 fish
#> Create a new key user:/tests/ours/key5 with string "fish"
kdb set user:/tests/theirs/key1 1
#> Create a new key user:/tests/theirs/key1 with string "1"
kdb set user:/tests/theirs/key2 pie
#> Create a new key user:/tests/theirs/key2 with string "pie"
kdb set user:/tests/theirs/key4 banana
#> Create a new key user:/tests/theirs/key4 with string "banana"
kdb set user:/tests/theirs/key5 5
#> Create a new key user:/tests/theirs/key5 with string "5"
```

Now we will examine the result KeySet with the different strategies.

### 470.4.1 Preserve

```
kdb merge -s preserve user:/tests/ours user:/tests/theirs user:/tests/base user:/tests/result
# RET: 11
# STDERR: 1 conflicts were detected that could not be resolved automatically:↩user:/tests/result/key4↩ours:
       CONFLICT_DELETE, theirs: CONFLICT_MODIFY↩↩Merge unsuccessful.
```

The merge will fail because of a conflict for `key4` since `key4` was deleted in our KeySet and edited in their KeySet. Since we used preserve, the merge fails and the result KeySet is not saved.

### 470.4.2 Ours

```
kdb merge -s ours user:/tests/ours user:/tests/theirs user:/tests/base user:/tests/result
```

The result KeySet, `user:/tests/result` will be:

```
kdb ls user:/tests/result
#> user:/tests/result/key1
#> user:/tests/result/key2
#> user:/tests/result/key5
```

The values of the keys are:

```
kdb get user:/tests/result/key1
#> apple
kdb get user:/tests/result/key2
#> pie
kdb get user:/tests/result/key5
#> fish
```

The conflict of `key4` (it was deleted in `ours` but changed in `theirs`) is solved by using our copy, thus deleting the key.

Now we delete the result keys and try the next merging strategy.

```
kdb rm user:/tests/result/key1
kdb rm user:/tests/result/key2
kdb rm user:/tests/result/key5
```

### 470.4.3 Theirs

```
kdb merge -s theirs user:/tests/ours user:/tests/theirs user:/tests/base user:/tests/result
```

The result KeySet, `user:/tests/result` will be:

```
kdb ls user:/tests/result
#> user:/tests/result/key1
#> user:/tests/result/key2
#> user:/tests/result/key4
#> user:/tests/result/key5
```

The values of the keys are:

```
kdb get user:/tests/result/key1
#> apple
kdb get user:/tests/result/key2
#> pie
kdb get user:/tests/result/key4
#> banana
kdb get user:/tests/result/key5
#> fish
```

Here, the conflict of `key4` is solved by using their copy, thus `key4=banana`.

We delete the result keys again and finally try the `cut` merging strategy.

```
kdb rm user:/tests/result/key1
kdb rm user:/tests/result/key2
kdb rm user:/tests/result/key4
kdb rm user:/tests/result/key5
```

### 470.4.4 Cut

```
kdb merge -s cut user:/tests/ours user:/tests/theirs user:/tests/base user:/tests/result
```

The result KeySet, `user:/tests/result` will be:

```
kdb ls user:/tests/result
#> user:/tests/result/key1
#> user:/tests/result/key2
#> user:/tests/result/key4
#> user:/tests/result/key5
```

The values of the keys are:

```
kdb get user:/tests/result/key1
#> 1
kdb get user:/tests/result/key2
#> pie
kdb get user:/tests/result/key4
#> banana
kdb get user:/tests/result/key5
#> 5
```

Here the state of theirs is simply copied to the `resultpath`.

## 470.5 SEE ALSO

- kdb-merge(1)
- elektra-merge-strategy(7)

# Chapter 471

# Mounting

In this tutorial we will go over mounting configuration files to enable non-elektrified applications to be configured through Elektra.
The best way of integrating Elektra into applications is to elektrify them.
To use Elektra with non-elektrified applications we can also let Elektra sync configuration files on the file system and Elektra's key-value storage. Applications can then read and write the configuration files and changes in the key database will be picked up by applications and vice-versa.
The heart of the approach is the so-called *mounting* of configuration files into the key database.
Let us start with a motivating example first:

## 471.1 Mount the Lookup Table for Hostnames

We mount the lookup table with the following command:
```
sudo kdb mount --with-recommends /etc/hosts system:/hosts hosts
```

1. `/etc/hosts` is the configuration file we want to mount
2. `system:/hosts` is the path it should have in the key database, also known as **mount point**
3. `hosts` is the *storage plugin* that can read and write this configuration format.

   Consider using mount with the option `--with-recommends`, which loads all plugins recommended by the *hosts* plugin. You can see the recommended plugins of *hosts* if you look at the output of `kdb plugin-info hosts`. Hosts recommends the *glob*, *network* and *error* plugins. Using `--with-recommends`, more validation is done when modifying keys in `system:/hosts`.

Now we use `kdb file`, to verify that all configuration below `system:/hosts` is stored in `/etc/hosts`:
```
kdb file system:/hosts
#> /etc/hosts
```
After mounting a file, we can modify keys below `system:/hosts`. We need to be root, because we modify `/etc/hosts`.
```
sudo kdb set system:/hosts/ipv4/mylocalhost 127.0.0.33
```
These changes are reflected in `/etc/hosts` instantly:
```
cat /etc/hosts | grep mylocalhost
#> 127.0.0.33   mylocalhost
```
Applications will now pick up these changes:
```
ping -c 1 mylocalhost
# RET:0
```
We are also safe against wrong changes:
```
sudo kdb set system:/hosts/ipv4/mylocalhost ::1
# RET:5
# ERROR:C03200
sudo kdb set system:/hosts/ipv4/mylocalhost 300.0.0.1
# RET:5
# ERROR:C03200
```
We can undo these changes with:
```
# remove the key ...
sudo kdb rm system:/hosts/ipv4/mylocalhost
# ... and unmount
sudo kdb umount system:/hosts
```

**Why do you Need Superuser Privileges to Mount Files?** Elektra manages its mount points in configuration below **system:/elektra/mountpoints**. The file that holds this configuration is, in the same way as `/etc/hosts` before, only writable by administrators:

```
kdb file system:/elektra/mountpoints
#> /etc/kdb/elektra.ecf
```

Because of that only root can mount files.

## 471.2 Resolver

The configuration file path you supplied to `kdb mount` above is actually not an absolute or relative path in your file system, but gets resolved to one by Elektra. The plugin that is responsible for this is the _Resolver_.
When you mount a configuration file the resolver first looks at the namespace of your mount point. Based on that namespace and if the supplied path was relative or absolute the resolver then resolves the supplied path to a path in the file system. The resolving happens dynamically for every `kdb` invocation.
You can display the mounted configuration files with `kdb mount`. Also here you only see the unresolved paths.
If you supplied an absolute path (e.g. `/example.ini`) it gets resolved to this:

| namespace | resolved path |
|---|---|
| spec | /example.ini |
| dir | ${PWD}/example.ini |
| user | ${HOME}/example.ini |
| system | /example.ini |

If you supplied a relative path (e.g. `example.ini`) it gets resolved to this:

| namespace | resolved path |
|---|---|
| spec | /usr/share/elektra/specification/example.ini |
| dir | ${PWD}/.dir/example.ini |
| user | ${HOME}/.config/example.ini |
| system | /etc/kdb/example.ini |

If this differs on your system, the resolver has a different configuration. Type `kdb plugin-info resolver` for more information about the resolvers.
There are different resolvers. For instance on non-POSIX systems paths must be resolved differently. In this case one might want to use the wresolver plugin. Another useful resolver is the blockresolver, which integrates only a block of a configuration file into Elektra.
But resolvers are not the only plugins Elektra uses:

## 471.3 Plugins

Configuration files can have many different formats (`ini`, `json`, `yaml`, `xml`, `csv`, ... to name but a few).
One of the goals of Elektra is to provide users with a unified interface to all those formats. Elektra accomplishes this task with *storage plugins*.

> In Elektra Plugins are the units that encapsulate functionality. There are not only plugins that handle storage of data, but also plugins that modify your values (iconv). Furthermore there are plugins that validate your values (validation, mathcheck, ...), log changes in the key set (logchange) or do things like executing commands on the shell (shell). You can get a complete list of all available plugins with `kdb plugin-list`. Although an individual plugin does not provide much functionality, plugins are powerful because they are designed to be used together.

When you mount a file you can tell Elektra which plugins it should use for reading and writing to configuration files.

#### 471.3.0.1 Metadata

Elektra is able to store metadata of keys. The ni plugin and the dump plugin, among others, support this feature.

Metadata comes in handy if we use other plugins, than just the ones that store and retrieve data. I chose the `ni` plugin for this demonstration, because it supports metadata and is human-readable. So let us have a look at the type and mathcheck plugins.

```
# mount the backend with the plugins ...
sudo kdb mount example.ni user:/example ni type
# ... and set a value for the demonstration
kdb set user:/example/enumtest/fruit apple
#> Create a new key user:/example/enumtest/fruit with string "apple"
```

By entering `kdb plugin-info type` in the commandline, we can find out how to use this plugin. It turns out that this plugin allows us to define a list of valid values for our keys via the metavalue `check/enum`.

```
kdb meta-set user:/example/enumtest/fruit check/enum "#2"
kdb meta-set user:/example/enumtest/fruit check/enum/#0 apple
kdb meta-set user:/example/enumtest/fruit check/enum/#1 banana
kdb meta-set user:/example/enumtest/fruit check/enum/#2 grape
kdb meta-set user:/example/enumtest/fruit check/type enum
kdb set user:/example/enumtest/fruit tomato
# RET:5
# this fails because tomato is not in the list of valid values
```

You can have a look or even edit the configuration file with `kdb editor user:/example ni` to see how the value and metadata is stored:

```
enumtest/fruit = apple
[enumtest/fruit]
check/type = enum
check/enum = #2
check/enum/#0 apple
check/enum/#1 banana
check/enum/#2 grape
```

The example shows an important problem: the configuration file is now changed in ways that might not be acceptable for applications. We have at least two ways to avoid that:

1. Encode metadata as comments
2. Encode metadata in its own `spec` namespace, completely separate to the configuration files the application will see

If you want to find out more about validation I recommend reading this tutorial next.

#### 471.3.0.2 Backends

The plugins together with the configuration file form a *backend*. The backend determines how Elektra stores data below a mount point. You can examine every mount points backend by looking at the configuration below `system:/elektra/mountpoints/<mount point>/`.

## 471.4 Limitations

One drawback of this approach is, that an application can bypass Elektra and change configuration files directly. If for example Elektra is configured to validate new configuration values before updating them, this is something you do not want to happen.
Another drawback is that mounting is static. In a previous example we mounted the `/.git/config` file into `dir:/git`. Now the `dir` namespace of every directory stores the configuration below `dir:/git` in this directories `/.git/config` file. And this mount point is the same for all users and all directories. So you can't have different configuration files for the same mount points in other directories. Because of the same reason you cannot have different configuration file names or syntax for the same mount point in the `user` namespace.
This is one of the reasons why Elektra promotes this naming convention for keys:

> Key names of software-applications should always start with: `/<type>/<org>/<name>/<version>/<profile>`
>
> - **type** can be `sw` (software), `hw` (hardware) or `elektra` (for internal configuration)
>
> - **org** is an URL/organization name. E.g. `kde`
> - **name** the name of the component that has this configuration
> - **version** is the major version number. E.g. If your version is 6.3.8 than this would be `#6`
> - **profile** is the name of the profile to be used. E.g.: `production`, `development`, `testing`, ...

Furthermore, one cannot simply change the configuration file format, because it must be one the application understands. Thus one loses quite some flexibility (for instance if this file format doesn't support meta keys, as already mentioned).
These limitations are the reasons why elektrifing applications provides even better integration. Go on reading how to elektrify your application.

# Chapter 472

# Understanding Namespaces

## 472.1 Structure of the Key Database

The *key database* of Elektra is *hierarchically structured*. This means that keys are organized similar to directories in a file system.
Let us add some keys to the database. To add a key we can use `kdb`, the *key database access tool*:

```
kdb set <key> <value>
```

Now add the key **user:/tests/a** with the Value **Value 1** and the key **user:/tests/b/c** with the Value **Value 2**

```
kdb set user:/tests/a 'Value 1'
#> Create a new key user:/tests/a with string "Value 1"
kdb set user:/tests/b/c 'Value 2'
#> Create a new key user:/tests/b/c with string "Value 2"
```

The database has an hierarchical structure. For instance the key **user:/tests/b/c** has the path **user:/** -> **user:/tests** -> **user:/tests/b** -> **user:/tests/b/c**.
Note how the name of the key determines the path to its value.
You can use the file system analogy as a mnemonic to remember these commands (like the file system commands in your favorite operating system):

- `kdb ls <path>` lists keys below *path*
- `kdb rm <key>` removes a *key*
- `kdb cp <source> <dest>` copies a key to another path
- `kdb get <key>` gets the value of *key*

For example `kdb get user:/tests/b/c` should return `Value 2` now, if you set the values before.

## 472.2 Namespaces

Now we abandon the file system analogy and introduce the concept of *namespaces*.
Above all of the keys have been prefixed with `user:/`. The user namespace is one of several in Elektra.
Every key in Elektra belongs to one of these namespaces:

- **spec** for specification of other keys
- **proc** for in-memory keys (e.g. command-line)
- **dir** for dir keys in current working directory
- **user** for user keys in home directory
- **system** for system keys in `/etc` or `/`
- **default** for keys which are created with default value (if specification requires the key to exist, has default and key was missing)

All namespaces save their keys in a *separate hierarchical structure* from the other namespaces.
But when we set the keys ∗∗/a∗∗ and ∗∗/b/c∗∗ before we didn't provide a namespace. So I hear you asking, "if every key has to belong to a namespace, where are the keys?" They are in the *user* namespace, as you can verify with:

```
kdb ls user:/ | grep -E '(tests/a|tests/b/c)'
#> user:/tests/a
#> user:/tests/b/c
```

At this point the key database should have this structure:

### 472.2.1 Cascading Keys

Another question you may ask yourself now is, what happens if we look up a key without providing a namespace. So let us retrieve the key ∗∗/b/c∗∗ with the -v flag in order to make *kdb* more talkative.

```
kdb get -v /tests/b/c
# STDOUT-REGEX: got [[:digit:]]+ keys↩searching spec:/tests/b/c, found: <nothing>, options: KDB_O_CALLBACK↩
          searching proc:/tests/b/c, found: <nothing>↩    searching dir:/tests/b/c, found: <nothing>↩
       searching user:/tests/b/c, found: user:/tests/b/c↩The resulting keyname is user:/tests/b/c↩The
       resulting value size is 8↩Value 2
```

Here you see how Elektra searches all namespaces for matching keys in this order: **spec**, **proc**, **dir**, **user** and finally **system**
If a key is found in a namespace, it masks the key in all subsequent namespaces, which is the reason why the system namespace isn't searched. Finally, the virtual key ∗∗/b/c∗∗ gets resolved to the real key **user:/b/c**. Because of the way a key without a namespace is retrieved, we call keys, that start with '∗∗/∗∗', **cascading keys**. You can find out more about cascading lookups in the cascading tutorial.
Having namespaces enables both admins and users to set specific parts of the application's configuration, as you will see in the following example.

## 472.3 How it Works on the Command Line (kdb)

We will provide an example of how you can configure elektrified applications.
Our exemplary application will be the key database access tool `kdb` as this should already be installed on your system.
`kdb` can be configured by the following configuration data:

- _/sw/elektra/kdb/#∗∗X∗∗/∗∗PROFILE∗∗/verbose_ - sets the verbosity of kdb
- _/sw/elektra/kdb/#∗∗X∗∗/∗∗PROFILE∗∗/quiet_ - if kdb should suppress non-error messages
- _/sw/elektra/kdb/#∗∗X∗∗/∗∗PROFILE∗∗/namespace_ - specifies the default namespace used, when setting a cascading name

**X** is a placeholder for the *major version number* and **PROFILE** stands for the name of a *profile* to which this configuration applies. If we want to set configuration for the default profile we can set **PROFILE** to %. The name of the key follows the convention described here.
Say we want to set `kdb` to be more verbose when it is used in the current directory. In this case we have to set *verbose* to 1 in the *dir* namespace of the current directory.

```
kdb set "dir:/sw/elektra/kdb/#0/%/verbose" 1
#> Create a new key dir:/sw/elektra/kdb/#0/%/verbose with string "1"
```

> The configuration for a directory is actually stored in this directory. By default the configuration is contained in a folder named `.dir`, as you can verify with `kdb file dir:/` (*kdb file* tells you the file where a key is stored in).
>
> For the purpose of demonstration we chose to only manipulate the verbosity of kdb. Note that setting `dir:/sw/elektra/kdb/#0/%/namespace` to `dir` can be handy if you want to work with configuration of an application in a certain directory.

If we now search for some key, `kdb` will behave just as if we have called it with the `-v` option.

```
kdb get /some/key
# RET: 10
#> got 4 keys
#> searching spec:/some/key, found: <nothing>, options: KDB_O_CALLBACK
#>     searching proc:/some/key, found: <nothing>
#>     searching dir:/some/key, found: <nothing>
#>     searching user:/some/key, found: <nothing>
#>     searching system:/some/key, found: <nothing>
```

```
#>     searching default:/some/key, found: <nothing>
#>     searching default of spec/some/key, found: <nothing>, options: KDB_O_NOCASCADING
# STDERR: Did not find key '/some/key'
```

Verbosity is not always useful because it distracts from the essential. So we may decide that we want `kdb` to be only verbose if we are debugging it. So let us move the default configuration to another profile:

```
kdb mv -r "dir:/sw/elektra/kdb/#0/%" "dir:/sw/elektra/kdb/#0/debug"
#> using common basename: dir:/sw/elektra/kdb/#0
#> key: dir:/sw/elektra/kdb/#0/%/verbose will be renamed to: dir:/sw/elektra/kdb/#0/debug/verbose
#> Will write out:
#> dir:/sw/elektra/kdb/#0/debug/verbose
```

If we now call `kdb get /some/key` it will behave non-verbose, but if we call it with the *debug* profile `kdb get -p debug /some/key` the configuration under ∗∗/sw/elektra/kdb/#0/debug∗∗ applies.

We configured kdb only for the current directory. If we like this configuration we could move it to the system namespace, so that every user can enjoy a preconfigured *debug* profile.

```
sudo kdb mv -r "dir:/sw/elektra/kdb" "system:/sw/elektra/kdb"
#> using common basename: /sw/elektra/kdb
#> key: dir:/sw/elektra/kdb/#0/%/verbose will be renamed to: system:/sw/elektra/kdb/#0/%/verbose
#> Will write out:
#> system:/sw/elektra/kdb/#0/%/verbose
```

Now every user could use the *debug* profile with kdb.

*Cascading keys* are keys that start with ∗∗/∗∗ and are a way of making key lookups much easier. Let's say you want to see the configuration from the example above. You do not need to search every namespace by yourself. Just make a lookup for ∗∗/sw/elektra/kdb/#0/debug/verbose∗∗, like this:

```
kdb get "/sw/elektra/kdb/#0/debug/verbose"
#> 1
```

When using cascading key the best key will be searched at run-time. If you are only interested in the system key, you would use:

```
kdb get "system:/sw/elektra/kdb/#0/debug/verbose"
#> 0
```

Because of *cascading keys* a user can override the behavior of the *debug* profile by setting the corresponding keys in their *user* namespace (as we discussed `before`). If a user sets *verbose* in their user namespace to 0 they override the default behavior from the *system* namespace.

Assuming that **sytem:/sw/elektra/kdb/#0/debug/verbose** is set to **1**, a user could override this by setting

```
kdb set "user:/sw/elektra/kdb/#0/debug/verbose" 0
#> Create a new key user:/sw/elektra/kdb/#0/debug/verbose with string "0"
kdb get "/sw/elektra/kdb/#0/debug/verbose"
#> 0
```

Now `kdb get -p debug /some/key` is not verbose anymore for this user.

# Chapter 473

# Notification Tutorial

## 473.1 Preface

**The features described in this document are experimental.**
This document explains how notifications are implemented in Elektra and how they can be used by application developers.

## 473.2 Notifications - Overview & Concept

Elektra's notification feature consists of several components. While sending and receiving notifications is implemented by plugins, applications use the notification API in order to use different plugins.
The `notification API` implemented by the `elektra-notification` library allows receiving and handling notifications. An `I/O abstraction layer` allows asynchronous notification processing by compatible plugins. The abstraction layer consists of an *interface* used by transport plugins and different implementations of that interface called *I/O bindings*. An I/O binding implements the actual I/O management functions for a specific event loop API. Applications typically use one I/O binding but can also use none or multiple I/O bindings. For more on I/O bindings see the `API documentation`.
Transport plugins exchange notifications via different protocols like D-Bus or ZeroMQ. For each type of transport there are typically two types of plugins: One for sending and one for receiving notifications. Developers do not interact with those plugins directly. The underlying transports are transparent to them. The "internalnotification" plugin implements notification handling functions and feeds back configuration changes from within the application. It is only used internally by the `elektra-notification` library.
When a configuration key is changed Elektra can generate change notifications that allow applications to process those changes. Developers can choose whether and how they want to receive and handle those notifications but not whether notifications are sent or which transport is used. How notifications are sent is specified in the *notification configuration* by the system operator.

## 473.3 Notification Configuration

System operators can mount the desired transport plugins and configure them (e.g. set channel, host, port and credentials) globally.
They need to mount both sending and receiving plugins in order to use transport.

```
kdb global-mount dbus announce=once dbusrecv
```

Plugins usable as transport plugin are marked with `notification` on the `plugin page`.

## 473.4 How to integrate an I/O binding and send notifications asynchronously

Developers do not need to change their programs in order to start sending notifications. However, without the integration of an I/O binding notifications *may* be sent synchronously which would block normal program execution. For programs without time constraints (e.g. CLI programs) this may not be important, but for GUIs or network services this will have negative impact.

The "zeromqsend" and "dbus" plugins do not block program execution for sending as sending is handled asynchronously be the underlying libraries.
The following is a basic example of an application using Elektra extended by the initialization of an I/O binding.

```
#include <elektra/kdb.h>
#include <elektra/kdbio.h>
#include <elektra/kdbio/uv.h>
#include <uv.h>
int main (void)
{
        // Create libuv event loop
        uv_loop_t * loop = uv_default_loop ();
        // Initialize I/O binding tied to event loop
        ElektraIoInterface * binding = elektraIoUvNew (loop);
        // Create contract that tells Elektra to use the I/O binding
        KeySet * contract = ksNew (0, KS_END);
        elektraIoContract (contract, binding);
        // Open KDB (with contract)
        Key * key = keyNew ("/sw/myorg/myapp/#0/current", KEY_END);
        KDB * kdb = kdbOpen (contract, key);
        // Normal application setup code ...
        // Start the event loop
        uv_run (loop, UV_RUN_DEFAULT);
        // Cleanup
        ksDel (contract);
        kdbClose (kdb, key);
        elektraIoBindingCleanup (binding);
        uv_loop_close (loop);
}
```

Make sure to compile/link with `pkg-config --libs --cflags elektra-io-uv`.

## 473.5 How to receive notifications

Since there are many different I/O management libraries (e.g. libuv, glib or libev), the transport plugins use the I/O interface for their I/O operations. Each I/O management library needs its own I/O binding. Developers can also create their own I/O binding for the I/O management library of their choice. This is described in the last section.
Each I/O binding has its own initialization function that creates a new I/O binding and connects it to the I/O management library. For this tutorial we will assume that libuv 1.x is used. For details on how to use a specific binding please look at available I/O bindings on the bindings page.
In order to handle change notifications a developer can either register a variable or a callback.

### 473.5.1 Register a variable

Values of registered variables are automatically updated when the value of the assigned key has changed. In the following example we will register an integer variable.
The following examples are shortened for tangibility. The complete code is available in "notification↩Async" example.

```
#include <elektra/kdb.h>
#include <elektra/kdbio.h>
#include <elektra/kdbio/uv.h>
#include <elektra/kdbnotification.h>
#include <uv.h>
static void printVariable (ElektraIoTimerOperation * timerOp)
{
        int value = *(int *) elektraIoTimerGetData (timerOp);
        printf ("\nMy integer value is %d\n", value);
}
int main (void)
{
        // Create libuv event loop
        uv_loop_t * loop = uv_default_loop ();
        // Initialize I/O binding tied to event loop
        ElektraIoInterface * binding = elektraIoUvNew (loop);
        // Create contract that tells Elektra to use the I/O binding
        KeySet * contract = ksNew (0, KS_END);
        elektraIoContract (contract, binding);
        // Add notifications to the contract
        elektraNotificationContract (contract);
        // Open KDB
        Key * key = keyNew ("/sw/myorg/myapp/#0/current", KEY_END);
        KDB * kdb = kdbOpen (contract, key);
        // Register "value" for updates
        Key * registeredKey = keyNew ("/sw/myorg/myapp/#0/current/value", KEY_END);
        int value;
        elektraNotificationRegisterInt (kdb, registeredKey, &value);
        // Create a timer to repeatedly print "value"
        ElektraIoTimerOperation * timer = elektraIoNewTimerOperation (2000, 1, printVariable, &value);
```

```
        elektraIoBindingAddTimer (binding, timer);
        // Get configuration
        KeySet * config = ksNew(0, KS_END);
        kdbGet (kdb, config, key);
        printVariable (timer);   // "value" was automatically updated
        // Start the event loop
        uv_run (loop, UV_RUN_DEFAULT);
        // Cleanup
        kdbClose (kdb, key);
        elektraIoBindingRemoveTimer (timer);
        elektraIoBindingCleanup (binding);
        uv_loop_close (loop);
}
```

After calling `elektraNotificationRegisterInt()` the variable `value` will be automatically updated if the key in the program above is changed by another program (e.g. by using the `kdb` CLI command). For automatic updates to work transport plugins have to be mounted globally.

### 473.5.2 Callbacks

Registering a variable is suitable for programs where the key's value is simply displayed or used repeatedly (e.g. by a timer or in a loop). If an initialization code needs to be redone after configuration changes (e.g. a value sets the number of worker threads) updating a registered variable will not suffice. For these situations a callback should be used.

The following snippet shows how a callback can be used if the value of the changed key needs further processing.

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
// from https://en.wikipedia.org/wiki/ANSI_escape_code#Colors
#define ANSI_COLOR_RED                  "\x1b[31m"
#define ANSI_COLOR_GREEN                "\x1b[32m"
void setTerminalColor (Key * color, void * context ELEKTRA_UNUSED)
{
        // context contains whatever was passed as 4th parameter
        // to elektraNotificationRegisterCallback()
        char * value = keyString (color);
        if (strcmp (value, "red") == 0)
        {
                printf (ANSI_COLOR_RED);
        }
        if (strcmp (value, "green") == 0)
        {
                printf (ANSI_COLOR_GREEN);
        }
}
int main (void)
{
        KDB * repo;
        // ... initialization of KDB, I/O binding and notifications
        Key * color = keyNew ("/sw/myorg/myapp/#0/current/color", KEY_END);
        // Re-Initialize on key changes
        elektraNotificationRegisterCallback(repo, color, &setTerminalColor, NULL);
        // ... start loop, etc.
}
```

### 473.5.3 How-To: Reload KDB when Elektra's configuration has changed

This section shows how the notification feature is used to reload an application's KDB instance when Elektra's configuration has changed. This enables applications to apply changes to mount points or globally mounted plugins without restarting.

**Step 1: Register for changes to Elektra's configuration**

To achieve reloading on Elektra configuration changes we register for changes below the key `/elektra` using `elektraNotificationRegisterCallbackSameOrBelow()`.

```
Key * elektraKey = keyNew ("/elektra", KEY_END);
elektraNotificationRegisterCallbackSameOrBelow (kdb, elektraKey, elektraChangedCallback, NULL))
keyDel (elektraKey);
```

**Step 2: Create a function for reloading KDB**

Since our application needs to repeatedly initialize KDB on configuration changes we need to create a function which cleans up and re-initializes KDB.

```
void initKdb (ElektraIoTimerOperation * timerOp ELEKTRA_UNUSED)
{
        if (kdb != NULL)
        {
                // Cleanup and close KDB
                kdbClose (kdb, parentKey);
        }
        KeySet * contract = ksNew (0, KS_END);
```

```
        elektraIoContract (contract, binding);
        elektraNotificationContract (contract);
        kdb = kdbOpen (parentKey);
        // Code for registration from snippet before
        Key * elektraKey = keyNew ("/elektra", KEY_END);
        elektraNotificationRegisterCallbackSameOrBelow (kdb, elektraKey, elektraChangedCallback, NULL);
        keyDel (elektraKey);
        // TODO: add application specific registrations
        // Get configuration
        kdbGet (kdb, config, parentKey);
}
```

**Step 3: Handle configuration changes**

The last step is to connect the registration for changes to the (re-)initialization function. Directly calling the function is discouraged due to tight coupling (see guidelines in the next section) and also results in an application crash since the notification API is closed while processing notification callbacks. Therefore, we suggest to add a timer operation with a sufficiently small interval which is enabled only on configuration changes. This timer will then call the initialization function.

First, we create the timer in the main loop setup of the application.

```
// (global declaration)
ElektraIoTimerOperation * reload;
// main loop setup (e.g. main())
// the timer operation is disabled for now and
// will reload KDB after 100 milliseconds
reload = elektraIoNewTimerOperation (100, 0, initKdb, NULL);
elektraIoBindingAddTimer (binding, reload);
```

Now we add the callback function for the registration from step 1:

```
void elektraChangedCallback (Key * changedKey, void * context)
{
        // Enable operation to reload KDB as soon as possible
        elektraIoTimerSetEnabled (reload, 1);
        elektraIoBindingUpdateTimer (reload);
}
```

Finally, we disable the timer in the initialization function:

```
void initKdb (void)
{
        // Stop reload task
        elektraIoTimerSetEnabled (reload, 0);
        elektraIoBindingUpdateTimer (reload);
        if (kdb != NULL)
        {
                // Cleanup notifications and close KDB
                kdbClose (kdb, parentKey);
        }
        // ...
}
```

By correct application of these three steps any application can react to changes to Elektra's configuration. The snippets above omit error handling for brevity. The complete code including error handling is available in the `"notification reload" example`. This example also omits globals by passing them as user data using the `elektraIo*GetData()` functions.

## 473.6 Emergent Behavior Guidelines

When applications react to configuration changes made by other applications this can lead to *emergent behavior*. We speak of emergent behavior when the parts of a system are functioning as designed but an unintended, unexpected or unanticipated behavior at system level occurs.

For example, take the following sequence of events:

1. application `A` changes its configuration
2. application `B` receives a notification about the change from `A` and updates its configuration

Given these two steps, the sequence could be a case of *wanted* emergent behavior: Maybe application `B` keeps track of the number of global configuration changes. Now consider adding the following events to the sequence:

1. application `A` receives a notification about the change from `B` and changes its configuration
2. *continue at step 2*

The additional step causes an infinite cycle of configuration updates which introduces *unwanted* behavior.

When designing a system it is desirable to use components with predictable and well-defined behavior. As a system grows larger and gets more *complex* unpredictable behavior emerges that was neither intended by the system

designer nor by the designer of the components. This system behavior is called *emergent behavior* if it cannot be explained from its components but only from analysis of the whole system.
Emergent behavior can be beneficial for a system. An example from nature is useful cooperation in an ant colony. Nevertheless, it also has disadvantages. Systems that bear *unwanted* emergent behavior are difficult to manage and experience failures in the worst case. This kind of unwanted emergent behavior is called *emergent misbehavior*. Examples of emergent misbehavior are traffic jams or the Millennium Footbridge incident in London (see scientific paper or news article).
An evaluation of Elektra's notification feature shows that it can exhibit the following symptoms:

- **Synchronization** occurs due to the shared notification medium. Multiple applications receive a notification at the same time and execute their configuration update logic. In turn shared resources like hard disk or CPU become overutilized.
- **Oscillation** occurs when applications are reacting to configuration changes by other applications.
- In the worst case oscillation results in **livelock** when the frequency of configuration updates becomes so high that applications are only executing configuration update logic.
- **Phase change** is a sudden change of the system behavior in reaction to a minor change (e.g. incremental change of a configuration setting). Phase change is introduced by application logic.

Building on these findings we will now present guidelines for preventing emergent misbehavior when using the notification API and callbacks in particular.

### 473.6.1 Guideline 1: Avoid callbacks

> Most of the guidelines are related to callbacks. With normal use of the notification API emergent behavior should not occur.

Callbacks couple an application temporally to configuration changes of other applications or instances of the same application. This observation is the basis for Guidelines 1, 2 and 3. While it is possible with registered variables to check for configuration changes at regular time intervals and react to changes the coupling is not as tight as with callbacks.

### 473.6.2 Guideline 2: Wait before reacting to changes

> Waiting after receiving a notification decouples an application from changes and reduces the risk for unwanted ∗∗_synchronization_∗∗.

In applications where applying changes has impact on resource usage (e.g. CPU or disk) applying a time delay as suggested by this Guideline is a sensible choice. But this guideline is not only limited to these applications.
Generally waiting before reacting to changes reduces the risk for unwanted synchronization by decoupling the application temporally. Waiting can be implemented using random time delays which further promotes decoupling since applications react at different points in time to changes. Waiting can also be implemented using a flag : Callbacks set the flag and when the control flow is in the main loop again, the pending updates are applied and the flag is cleared.

### 473.6.3 Guideline 3: Avoid updates as reaction to change

> Avoid changing the configuration as reaction to a change. This reduces the risk for unwanted ∗∗_oscillation_∗∗.

While this guideline does not forbid updating the key database using `kdbSet()` in a callback it advises to avoid it. If we recall the example from before we see how updating as reaction to change leads to unwanted oscillation. If necessary, the function `kdbSet()` should be temporally decoupled as suggested in Guideline 2.
This guideline applies especially to callbacks but is also relevant when variables are polled for changes by the application.

### 473.6.4 Guideline 4: Do not use notifications for synchronization

> Applications should not use notifications for synchronization as this can lead to ∗∗_phase change_∗∗.

This guideline limits the use of the notification API to notifications about configuration changes. There are better suited techniques for different use cases. Applications should not keep track of changes and change their behavior on certain conditions.
For example, this happens when applications synchronize themselves at startup by incrementing a counter in the key database. When a certain limit of application instances is reached the applications proceed with different behavior. If this behavior affects other applications phase change has occurred.

### 473.6.5 Guideline 5: Apply changes immediately

> Call `kdbSet()` to save updated configuration immediately after a change occurred. This reduces conflicting changes in the key database.

When a configuration setting is updated within an application this guideline suggests to write the change immediately to the key database using `kdbSet()`. This ensures that other applications have the same view of the key database and operate on current settings.

### 473.6.6 Guideline 6: Be careful on what to call inside callbacks

> Notification callbacks are called from within Elektra. Calling `kdbClose()` in a callback will lead to undefined behavior or an application crash.

Closing and cleaning up the KDB handle will cause an application crash because the control flow returns from the callback to now removed code. While this can be considered an implementation detail it aligns with `Guideline 2` since reinitialization of KDB uses more resources than other operations like `kdbGet()` or `kdbSet()`.

## 473.7 Logging

In order to analyze application behavior the `logging plugins` ( `syslog`, `journald` or `logchange`) can be used. In order to log not only `kdbSet()` but also `kdbGet()`, the option `get=on` should be used when mounting these plugins.

```
kdb global-mount syslog get=on
```

Now both reading from and writing to Elektra's key database is logged.

## 473.8 How to create your own I/O Binding

Developers can create their own bindings if the I/O management library of their choice is not supported by an existing I/O binding.
For details on see the example doc" binding" or the `API documentation`. Existing I/O bindings provide a good inspiration on how to implement a custom binding. Since a binding is generic and not application specific it is much appreciated if you contribute your I/O binding back to the Elektra Initiative.

# Chapter 474

# The Elektra ODBC Backend

This tutorial describes how to set up unixODBC on Linux and use Elektra to retrieve configuration data from an *SQLite* or *PostgreSQL* database.

> Currently, the *backend_odbc* plugin is marked as EXPERIMENTAL and only data sources that define a table for *metadata* are supported. In the future, we plan to also support data sources without metadata tables.
>
> Please be aware that for using metadata, *outer joins* have to be supported by the ODBC driver.
>
> The ODBC backend plugin was tested with unixODBC, but should also work with iODBC and Microsoft ODBC (on Windows). If you are using such an environment, feel free to share you experiences at `https://issues.libelektra.org` or extend this tutorial.
>
> This tutorial uses SQLite- and PostgreSQL databases as data sources, but ODBC drivers for other data sources are also supported.

## 474.1 Overview

The tutorial covers the following steps:

1. Introduction
2. Setting up the Databases for Configuration Data
3. Installing unixODBC
4. Installing the SQLite and PostgreSQL ODBC Drivers
5. Creating the ODBC data sources for the Databases
6. Compiling Elektra with support for the ODBC Backend
7. Mounting the data sources into the global Key Database (KDB) of Elektra

## 474.2 1. Introduction

### 474.2.1 Database scheme

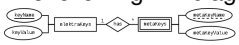
The basic database scheme is quite simple. You just need a table that contains two columns which can store a text value. Usually, the SQL data types `TEXT`, `CHAR` and `VARCHAR` are used for that purpose. They must contain valid UTF-8 sequences. One column is used for storing the **names** of the keys, the other one for storing their **values**. The column where the key-names are stored should be defined as the *primary key* (PK) for that table.
Next, we have to create a table for storing metadata. The table for the metadata needs at least three columns:

- A column with the *key-name* which should be defined as a *foreign key* (FK) to the PK of the table where the keys are stored.

- A column for the *name* of the metakeys.
- A column for the *value* of the metakeys.

  The column for the **key-name** and the column for the **metakey-name** together should form the PK of that table.

Again, all these columns must be defined to store an UTF-8 compatible text. It is allowed that the tables contain more columns, these additional columns are not processed by Elektra, *must not* be part of a PK and *must* support NULL- or DEFAULT-values (if you want to add new keys via Elektra).
The following ER-diagram shows the described scheme:



## 474.3 2. Setting up the Databases for Configuration Data

Before we are getting started with setting up unixODBC, we create the databases and store some configuration data in them.

> This tutorial is neither an introduction to SQL nor to SQLite or PostgreSQL. If you need more information about these topics, there are plenty of resources available.
>
> For downloads and documentation, please visit the respective websites:
>
> - https://www.sqlite.org
> - https://www.postgresql.org
>
> Usually, both database management systems (DBMS) can be installed by the package manager of your operating system.
>
> A pre-configured example SQLite database and the SQL-script that was used to create the tables and fill them with some test data is available at /src/plugins/backend_odbc/sampleDb.

### 474.3.1 Preparing the SQLite Database

As SQLite as a file-based DBMS, we first create a file for the new database. Afterward, we execute the SQL statements to create the tables and insert some tuples. Please note that the command `sqlite3` may be named differently on your system, especially if you use another version of SQLite.

```
sqlite3 ~/elektraOdbc.db
```

Then, the SQLite command prompt is started, where you can enter your SQL statements. We use the statements as defined in /src/plugins/backend_odbc/sampleDb/prepareDB.sql.

```
CREATE TABLE elektraKeys (
    keyName TEXT PRIMARY KEY NOT NULL,
    keyValue TEXT DEFAULT NULL
);
CREATE TABLE metaKeys (
    keyName TEXT NOT NULL,
    metaKeyName TEXT NOT NULL,
    metaKeyValue TEXT DEFAULT NULL,
    CONSTRAINT fk_metakeys FOREIGN KEY (keyName) REFERENCES elektraKeys (keyName),
    CONSTRAINT pk_metaKeys PRIMARY KEY (keyName, metaKeyName)
);
INSERT INTO elektraKeys (keyName, keyValue) VALUES ('sqliteapp1/key1', 'sqlite val 1.1');
INSERT INTO elektraKeys (keyName, keyValue) VALUES ('sqliteapp1/key2', 'sqlite val 1.2');
INSERT INTO elektraKeys (keyName, keyValue) VALUES ('sqliteapp2/key1', 'sqlite val 2.1');
INSERT INTO elektraKeys (keyName, keyValue) VALUES ('sqliteapp2/key2', 'sqlite val 2.2');
INSERT INTO elektraKeys (keyName, keyValue) VALUES ('sqliteapp2/key3', 'sqlite val 2.3');
INSERT INTO elektraKeys (keyName, keyValue) VALUES ('sqliteapp3/key1', 'sqlite val 3.1');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp1/key1', 'metakey 1.1.1',
       'metaval 1.1.1');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp1/key1', 'metakey 1.1.2',
       'metaval 1.1.2');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp1/key1', 'metakey 1.1.3',
       'metaval 1.1.3');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp1/key1', 'metakey 1.1.4',
       'metaval 1.1.4');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp2/key2', 'metakey 2.2.1',
       'metaval 2.2.1');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp2/key2', 'metakey 2.2.2',
       'metaval 2.2.2');
INSERT INTO metaKeys (keyName, metaKeyName, metaKeyValue) VALUES ('sqliteapp2/key3', 'metakey 2.3.1',
       'metaval 2.3.1');
```

With typing `.exit`, you can leave the SQLite command prompt and return to your shell.

### 474.3.2 Preparing the PostgreSQL Database

If you want to use PostgreSQL, a bit more initial work is necessary. It is recommended to set up a user account, create a database and then create the tables using similar SQL statements as given above. You can run the PostgreSQL instance locally on the same computer where you use Elektra, but also on another node in the network. With this approach, it is possible to provide a centralized configuration storage for a whole network. As PostgreSQL natively supports transactions and multiple clients, it is also possible to share the same tables between multiple Elektra clients. Another option is to save the configurations for different users in separate tables. When a user logs in on any client PC in the network, the table with the matching configuration data can then, with some scripting, be mounted via Elektra.
If you need detailed information about how to set up and use a PostgreSQL DBMS instance, there is excellent documentation, including tutorials for beginners, available at the `PostgreSQL website`.
However, for this tutorial, we present a single-user scenario on a local PC.

## 474.4 3. Installing unixODBC

The easiest way to install unixODBC, is to use the package manager of your operating system. For example, in Debian and Ubuntu you can install the package with the following command:
`sudo apt-get install unixodbc`
Alternatively, you can also compile the unixODBC package from source. The download is available at `https://www.unixodbc.org`. The website also offers some other useful information, like various manuals and a list of supported ODBC drivers. Using drivers not mentioned in that list is in many cases also possible.

## 474.5 4. Installing the SQLite and PostgreSQL ODBC Drivers

The ODBC backend was tested with the following ODBC drivers for unixODBC:

- SQLite: `http://www.ch-werner.de/sqliteodbc`
- PostgreSQL: `https://odbc.postgresql.org`

The installation of the drivers should be straightforward. Please refer to the instructions on the websites and the downloaded drivers for how to install these ODBC drivers on your system. Maybe the drivers are also offered as packages by the package manager of your OS.

## 474.6 5. Creating the ODBC data sources for the Databases

Now we have to create or edit two configuration files for unixODBC:

- **odbcinst.ini:** for letting unixODBC know about the drivers and where it can find them
- **odbc.ini:** for configuring the actual data sources, which can then be mounted into Elektra's KDB

On most systems, these files should be stored at `/etc/unixODBC/`. The content of the `/etc/unixODBC/odbcinst.ini` file should look like this:

```
[SQLite]
Description=SQLite ODBC Driver
Driver=/usr/local/lib/libsqlite3odbc.so
Setup=/usr/local/lib/libsqlite3odbc.so
FileUsage=1
Threading=2
[Postgresql]
Description=General ODBC for PostgreSQL
Driver=/usr/local/lib/psqlodbca.so
Setup=/usr/lib/libodbcpsqlS.so
Setup64=/usr/lib64/libodbcpsqlS.so
```

Please check these paths and adjust them so that they match the place where the drivers and setup libraries are stored on your system. At first, the name of the entry is defined, in our example `SQLite` and `Postgresql`, then the configuration for the driver is given.
The `Threading=2` option enables multithreading support for the SQLite driver. If you don't need to use multiple threads/instances accessing the SQLite ODBC data source in parallel, you can gain a bit of performance by activating the single-threaded mode. In this case you can set `Threading=0`.

> You have to make sure for yourself that only single-threaded use is present. This setting just disables mutexes in the driver. So if you are not really sure what you are doing, it's safer to enable the multi-thread support!
>
> More information about this setting is available at https://www.sqlite.org/threadsafe.html.

The setting `FileUsage=1` indicates to unixODBC that the driver is file-based.

### 474.6.1 odbc.ini

After the driver settings, we have to define the actual ODBC data sources in `/etc/unixODBC/odbc.ini`. This content of the file should look like this:

```
[Selektra]
Description=SQLite Database for Elektra
Driver=SQLite
Database=/home/user/elektraOdbc.db
Timeout=3500
[Pelektra]
Description=Postgresql
Driver=Postgresql
Trace=No
Database=elektraDB
Servername=localhost
Username=elektraUser
Password=elektra
Port=5432
Protocol=6.4
ReadOnly=No
RowVersioning=No
ShowSystemTables=No
ShowOidColumn=No
FakeOidIndex=No
```

At first, the data source name is chosen, in our example `Selektra` and `Pelektra`, then the configuration values for the data source are defined. The value for the key `Driver` must match the corresponding name of the entry that is defined in `odbcinst.ini`.

## 474.7 6. Compiling Elektra with support for the ODBC Backend

The plugin for the ODBC backend is currently marked as **EXPERIMENTAL** and therefore not built by default.
You must use the correct cmake-parameters for the plugin to be built. If you want to build all plugins, you can just use

```
cmake -DPLUGINS="ALL" <path to elektra root dir>
```

If you want the default behavior and just additionally include the ODBC backend plugin, you can use the following command:

```
cmake -DPLUGINS="ALL;backend_odbc;-EXPERIMENTAL" <path to elektra root dir>
```

Alternatively, you can also use the ncurses-based `ccmake` tool to define the option. For more information about how to build Elektra, please see /doc/COMPILE.md.
Check the cmake output to be sure that the build system finds the ODBC libraries. Otherwise, the `backend_odbc` plugin gets automatically excluded.

## 474.8 7. Mounting the data sources into the global Key Database (KDB) of Elektra

Now, we can finally mount our ODBC data sources into the global Key Database (KDB) of Elektra. Currently, a separate command `kdb mountOdbc` exists for this purpose. Unfortunately, in contrast to the well-known `kdb mount` for the file-based backend, currently no additional plugins are supported.
However, for now we use the `kdb mountOdbc` command. If you just type the command without additional arguments, you get some information about how the command works and which arguments are expected.

> Only `system:/` and `user:/` namespaces are supported for ODBC mountpoints.

Finally, we can create the mountpoint for our SQLite database using the `kdb mountOdbc` command. Further details about the arguments of the command are available in its man page.

```
kdb mountOdbc Selektra "" "" elektraKeys keyName keyValue metaKeys keyName metaKeyName metaKeyValue ""
       user:/odbcSqlite
```

```
# The new mountpoint for the ODBC data source was successfully created!
```

Now we can just access the data like with any other mountpoint.

```
kdb ls user:/odbcSqlite/
# user:/odbcSqlite/sqliteapp1/key1
# user:/odbcSqlite/sqliteapp1/key2
# user:/odbcSqlite/sqliteapp2/key1
# user:/odbcSqlite/sqliteapp2/key2
# user:/odbcSqlite/sqliteapp2/key3
# user:/odbcSqlite/sqliteapp3/key1
kdb get /odbcSqlite/sqliteapp1/key1
# sqlite val 1.1
kdb set /odbcSqlite/sqliteapp1/key1 newVal
kdb get /odbcSqlite/sqliteapp1/key1
# sqlite newVal
kdb meta-show user:/odbcSqlite/sqliteapp1/key1
# metakey 1.1.1: metaval 1.1.1
# metakey 1.1.2: metaval 1.1.2
# metakey 1.1.3: metaval 1.1.3
# metakey 1.1.4: metaval 1.1.4
```

For unmounting, there is not an extra command for ODBC data sources. You can unmount the ODBC backend, exactly like mounted files:

```
kdb umount user:/odbcSqlite
```

# Chapter 475

# How-To: Write a Plugin

This file serves as a tutorial on how to write a plugin.

## 475.1 Types of Plugins

- Storage plugins are used by Elektra in order to store data in the Elektra Key Database in an intelligent way. They act as a liaison between configuration files and the Key Database. Storage plugins are largely responsible for the functionality of Elektra and they allow many of its advanced features to work. These plugins act as sources and destinations of configuration settings.
- Filter plugins are simpler than storage plugins. They receive configuration settings in the same way as storage plugins but they do not have the responsibility to serialize the configuration settings to configuration files. For example, `checker` plugins which validate configuration are filter plugins.
- Resolver plugins are more complicated and not covered by this tutorial.

This tutorial mostly uses storage plugins as example but also explains differences to filter plugins.

## 475.2 Basics

First, there are a few basic points to understand about Elektra plugins. This first section will explain the basic layout of a plugin and what various methods exists within one.

### 475.2.1 The Interface

All plugins use the same basic interface. This interface consists of five basic functions:

- `elektraPluginOpen`,
- `elektraPluginGet`,
- `elektraPluginSet`,
- `elektraPluginError`, and
- `elektraPluginClose`

. The developer replaces `Plugin` with the name of their plugin. So in the case of the line plugin, the names of these functions would be `elektraLineOpen()`, `elektraLineGet()`, `elektraLineSet()`, `elektraLineError()`, and `elektraLineClose()`. Additionally, there is one more function called `ELEKTRA_PLUGIN_EXPORT`, where once again `Plugin` should be replaced with the name of the plugin, this time in uppercase. So for the line plugin this function would be `ELEKTRA_PLUGIN_EXPORT(line)`. The developer may also define `elektraPluginCheckConf()` if configuration validation at mount-time is desired.
The KDB relies on the first five functions for interacting with configuration files stored in the key database. Calls to `kdbGet()` and `kdbClose()` will call the functions `elektraPluginGet()` and `elektraPluginClose()` respectively for the plugin that was used to mount the configuration data. `kdbSet()` calls `elektraPluginSet()` but also `elektraPluginError()` when an error occurs. `elektraPluginOpen()` is

called before the first call to `elektraPluginGet()` or `elektraPluginSet()`. These functions serve different purposes that allow the plugin to work:

- `elektraPluginOpen()` is designed to allow each plugin to do initialization if necessary.
- `elektraPluginGet()` is designed to turn information from a configuration file into a usable `KeySet`, this is technically the only function that is **required** in a plugin.
- `elektraPluginSet()` is designed to store the information from the keyset back into a configuration file.
- `elektraPluginError()` is designed to allow proper rollback of operations if needed and is called if any plugin fails during the set operation. This is not needed for storage plugins as the resolver already takes care to unlink the configuration files in such situations.
- `elektraPluginClose()` is used to free resources that might be required for the plugin.
- `ELEKTRA_PLUGIN_EXPORT` simply lets Elektra know that the plugin exists and what the name of the above functions are.

Most simply put: most plugins consist of five major functions, `elektraPluginOpen()`, `elektraPluginClose()`, `elektraPluginGet()`, `elektraPluginSet()`, and `ELEKTRA_EXPORT_PLUGIN`.
Because remembering all these functions can be cumbersome, we provide a skeleton plugin in order to easily create a new plugin. The skeleton plugin is called `template` and a new plugin can be created by calling the copy-template script . For example, the author of the line plugin used the command `scripts/dev/copy-template line` to create the initial version of the plugin. Afterwards two important things are left to be done:

- remove all functions (and their exports) from the plugin that are not needed. For example not every plugin actually makes use of the `elektraPluginOpen()` function.
- provide a basic contract as described above

After these two steps your plugin is ready to be compiled, installed and mounted for the first time. Have a look at How-To: kdb mount

#### 475.2.1.1 C++ Based Plugins

If you want to use C++ instead of C for plugin development you can use `copy-template` to create a plugin based on `cpptemplate`. For example, to create a new plugin called `pluginbaby` use the command:
```
scripts/dev/copy-template -p pluginbaby
```

## 475.3 Contract

In Elektra, multiple plugins form a backend. If every plugin would do whatever it likes to do, there would be chaos and backends would be unpredictable.
To avoid this situation, plugins export a so-called *contract*. In this contract the plugin states how nicely it will behave and what other plugins can depend on.

### 475.3.1 Writing a Contract

Because the contracts also contain information for humans, these parts are written in a `README.md` files of the plugins. To make the contracts machine-readable, the following CMake command exists:
```
generate_readme(pluginname)
```
It will generate a `readme_plugginname.c` (in the build-directory) out of the `README.md` of the plugin's source directory.
But prefer to use
```
add_plugin(pluginname)
```
where the generation of the readme (among many other things) are already done for you. More details about how to write the `CMakeLists.txt` will be discussed later in the tutorial.
The `README.md` will be used by:

- the build system (`-DPLUGINS=`), e.g. to exclude experimental plugins (`infos/status`)
- the mount tool, e.g. to correctly place and order plugins
- to know dependencies between plugin and what metadata they process

### 475.3.2 Content of <tt>README.md</tt>

The first lines must look like:

```
- infos = Information about YAJL plugin is in keys below
- infos/author = Markus Raab <elektra@libelektra.org>
- infos/licence = BSD
- infos/provides = storage/json
- infos/needs = directoryvalue
- infos/recommends = rebase comment type
- infos/placements = getstorage setstorage
- infos/status = maintained coverage unittest
- infos/description = JSON using YAJL
```

Every of these line represents a clause of the contract. All these clauses need to be present for every plugin.
The information of clauses are limited to a single line, starting with - (so that the file renders nicely in Markdown), followed by the clause itself separated by =. Only for the description an unlimited amount of lines can be used (until the end of the file).
For the meaning (semantics) of these clauses, please refer to contract specification. For details of how plugins are ordered look here The only difference for filter plugins to storage plugins is that their `infos/provides` and `infos/placements` are different, e.g., for checker plugins `presetstorage` usually is enough.
The already mentioned `generate_readme` will produce a list of Keys using the information in `README.md`. It would look like (for the third key):

```
keyNew ("system:/elektra/modules/yajl/infos/licence",
        KEY_VALUE, "BSD", KEY_END);
```

## 475.4 Including <tt>readme_pluginname.c</tt>

In your plugin, specifically in your `elektraPluginGet()` implementation, you have to return the contract whenever configuration below `system:/elektra/modules/plugin` is requested:

```
if (!strcmp (keyName(parentKey), "system:/elektra/modules/plugin"))
{
        KeySet *moduleConf = elektraPluginContract();
        ksAppend(returned, moduleConf);
        ksDel(moduleConf);
        return 1;
}
```

The `elektraPluginContract()` is a method implemented by the plugin developer containing the parts of the contract not specified in `README.md`. An example of this function (taken from the `yajl` plugin):

```
static inline KeySet *elektraYajlContract()
{
        return ksNew (30,
        keyNew ("system:/elektra/modules/yajl",
                KEY_VALUE, "yajl plugin waits for your orders", KEY_END),
        keyNew ("system:/elektra/modules/yajl/exports", KEY_END),
        keyNew ("system:/elektra/modules/yajl/exports/get",
                KEY_FUNC, elektraYajlGet,
                KEY_END),
        keyNew ("system:/elektra/modules/yajl/exports/set",
                KEY_FUNC, elektraYajlSet,
                KEY_END),
#include "readme_yourplugin.c"
        keyNew ("system:/elektra/modules/yajl/infos/version",
                KEY_VALUE, PLUGINVERSION, KEY_END),
        keyNew ("system:/elektra/modules/yajl/config", KEY_END),
        keyNew ("system:/elektra/modules/yajl/config/",
                KEY_VALUE, "system",
                KEY_END),
        keyNew ("system:/elektra/modules/yajl/config/below",
                KEY_VALUE, "user",
                KEY_END),
        KS_END);
}
```

It basically only contains the symbols to be exported (these symbols depend on the functions the plugin provides) and the plugin version information that is always defined by the macro `PLUGINVERSION`.
As already said, `readme_yourplugin.c` is generated in the binary directory, so make sure that your `CMakeLists.txt` contains (prefer to use `add_plugin` where this is already done correctly):

```
include_directories (${CMAKE_CURRENT_BINARY_DIR})
```

## 475.5 CMake

For every plugin you have to write a `CMakeLists.txt`. If your plugin has no dependencies, you can skip this section. The full documentation of `add_plugin` is available here.

In order to understand how to write the `CMakeLists.txt`, you need to know that the same file is included multiple times for different reasons.

1. The first time, only the name of plugins and directories are enquired. In this phase, only the `add_plugin` should be executed.
2. The second time (if the plugin is actually requested), the `CMakeLists.txt` is used to detect if all dependencies are actually available.

This means that in the first time, only the `add_plugin` should be executed and in the second time the detection code together with `add_plugin`.
So that you can distinguish the first and second phase, the variable `DEPENDENCY_PHASE` is set to `ON` iff you should search for all needed CMake packages. You should avoid to search for packages otherwise, because this would:

- clutter the output
- introduce more variables into the `CMakeCache` which are irrelevant for the user
- maybe even find libraries in wrong versions which are incompatible to what other plugins need

So usually you would have:

```
if (DEPENDENCY_PHASE)
        find_package (MyLib QUIET)
        if (MYLIB_FOUND)
                # add testdata, test cases...
        else ()
                remove_plugin (myplugin "mylib not found")
        endif ()
endif ()
```

So if you are in the second phase (`DEPENDENCY_PHASE`), you will search for all dependencies, in this case `MyLib`. If all dependencies are satisfied, you add everything needed for the plugin, except the plugin itself. This happens after `endif ()`:

```
add_plugin (myplugin
        SOURCES
                ...
        LINK_LIBRARIES
                ${LIBXML2_LIBRARIES}
        DEPENDENCIES
                ${LIBXML2_FOUND}
        )
```

Important is that you pass the information which packages are found as boolean. The plugin will actually be added iff all of the `DEPENDENCIES` are true.
Note that no code should be outside of `if (DEPENDENCY_PHASE)`. It would be executed twice otherwise. The only exception is `add_plugin` which *must* be called twice to successfully add a plugin.

> Please note that the parameters passed to `add_plugin` need to be constant between all invocations. Some `find_package` cache their variables, others do not, which might lead to toggling variables. To avoid problems, create a variable containing all `LINK_LIBRARIES` or `DEPENDENCIES` within `DEPENDENCY_PHASE`.

If your plugin makes use of compilation variants you should also read the information there.

## 475.6 Coding

This section will focus on an overview of the kind of code you would use to develop a plugin. It gives examples from real plugins and should serve as a rough guide on how to write a storage plugin that can read and write configuration data into an Elektra `KeySet`.

### 475.6.1 <tt>elektraPluginGet</tt>

`elektraPluginGet` is the function responsible for turning information from a file into a usable `KeySet`. This function usually differs pretty greatly between each plugin. This function should be of type `int`, it returns either `1` or on `0` on success.

- `1`: The function was successful (`ELEKTRA_PLUGIN_STATUS_SUCCESS`).

- 0: The function was successful and the given keyset/configuration was **not changed** (ELEKTRA_PLUGIN_STATUS_NO_UPDATE).

Any other return value indicates an error (ELEKTRA_PLUGIN_STATUS_ERROR). The function will take in a Key, usually called parentKey which contains a string containing the path to the file that is mounted. For instance, if you run the command kdb mount /etc/linetest system:/linetest line then keyString(parentKey) should be equal to /etc/linetest. At this point, you generally want to open the file so you can begin saving it into keys. Here is the trickier part to explain. Basically, at this point you will want to iterate through the file and create keys and store string values inside of them according to what your plugin is supposed to do. I will give a few examples of different plugins to better explain.
The line plugin was written to read files into a KeySet line by line using the newline character as a delimiter and naming the keys by their line number such as #1, #2, .. #_22 for a file with 22 lines. So once I open the file given by parentKey, every time as I read a line I create a new key, let's call it new_key using dupKey(parentKey). Then I set new_key's name to lineNN (where NN is the line number) using keyAddBaseName and store the string value of the line into the key using keySetString. Once the key is initialized, I append it to the KeySet that was passed into the elektraPluginGet function, let's call it returned for now, using ksAppendKey(returned, new_key). Now the KeySet will contain new_key with the name #N properly saved where it should be according to the kdb mount command (in this case, system:/linetest/#N), and a string value equal to the contents of that line in the file. The line plugin repeats these steps as long as it hasn't reached end of file, thus saving the whole file into a KeySet line by line.
The simpleini plugin works similarly, but it parses for ini files instead of just line-by-line. At their most simple level, ini files are in the format of name=value with each pair taking one line. So for this plugin, it makes a lot of sense to name each Key in the KeySet by the string to the left of the = sign and store the value into each key as a string. For instance, the name of the key would be name and keyGetString(name) would return value.
As you may have noticed, simpleini and line plugins work very similarly. However, they just parse the files differently. The simpleini plugin parses the file in a way that is more natural to ini file (setting the key's name to the left side of the equals sign and the value to the right side of the equals sign). The elektraPluginGet function is the heart of a storage plugin, it's what allows Elektra to store configurations in its database. This function isn't just run when a file is first mounted, but whenever a file gets updated, this function is run to update the Elektra Key Database to match.

### 475.6.2 <tt>elektraPluginSet</tt>

We also give a brief overview of the elektraPluginSet function. This function is basically the opposite of elektraPluginGet. Where elektraPluginGet reads information from a file into the Elektra Key Database, elektraPluginSet writes information from the database back into the mounted file.
First have a look at the signature of elektraLineSet:

```
int elektraLineSet(Plugin *handle ELEKTRA_UNUSED, KeySet *toWrite, Key *parentKey);
```

Let's start with the most important parameters, the KeySet and the parentKey. The KeySet supplied is the KeySet that is going to be persisted in the file. In our case it would contain the Keys representing the lines. The parentKey is the topmost Key of the KeySet and serves several purposes. First, it contains the filename of the destination file as its value. Second, errors and warnings can be emitted via the parentKey. We will discuss error handling in more detail later. The Plugin handle can be used to persist state information in a thread-safe way with elektraPluginSetData. As our plugin is not stateful and therefore does not use the handle, it is marked as unused in order to suppress compiler warnings.
Basically the implementation of elektraLineSet can be described with the following pseudocode:

```
// open the file
if (error)
{
        ELEKTRA_SET_RESOURCE_ERROR(parentKey, keyString(parentKey));
}
for (/* each key */)
{
        // write the key value together with a newline
}
// close the file
```

The full-blown code can be found at line plugin.
As you can see, all elektraLineSet does is open a file, take each Key from the KeySet (remember they are named #1, #2 ... #_22) in order, and write each key as its own line in the file. Since we don't care about the name of the Key in this case (other than for order), we just write the value of keyString for each Key as a new line in the file. That's it. Now, each time the mounted KeySet is modified, elektraPluginSet will be called and the mounted file will be updated.

#### 475.6.2.1 <tt>ELEKTRA_SET_<CONCRETE_TYPE>_ERROR</tt>

We haven't discussed ELEKTRA_SET_<CONCRETE_TYPE>_ERROR yet. Because Elektra is a library, printing errors to stderr wouldn't be a good idea. Instead, errors and warnings can be appended to a key in the form of metadata. This is what ELEKTRA_SET_<CONCRETE_TYPE>_ERROR does. The <CONCRETE_TYPE> in the text means the concrete error type such as RESOURCE, INSTALLATION, etc. There are also abstract error types which are not instantiable. You can read more about concrete and abstract error types in the error-categorization.md guideline. Note that you also have a varargs macro with ...ERRORF that allows you to insert a string and substitute parts with variables. You can see all available error types as well as their categorization guidelines here. Because the parentKey always exists even if a critical error occurs, we write the error to the parentKey. The error does not necessarily have to be in a configuration. If there are multiple errors in a configuration, only the first occurrence will be written to the metadata of the parentKey.
The second parameter can be used to provide additional information about the error. In our case we simply supply the filename of the file that caused the error. The kdb tools will interpret this error and print it in a pretty way. Notice that this can be used in any plugin function where the parentKey is available.

### 475.6.3 <tt>elektraPluginOpen</tt> and <tt>elektraPluginClose</tt>

The elektraPluginOpen and elektraPluginClose functions are not commonly used for storage plugins, but they can be useful and are worth reviewing. elektraPluginOpen function runs before elektraPlugin↩Get and is useful to do initialization if necessary for the plugin. On the other hand elektraPluginClose is run after other functions of the plugin and can be useful for freeing up resources.

### 475.6.4 <tt>elektraPluginCheckConf</tt>

The elektraPluginCheckConf function may be used for validation of the plugin configuration during mount-time. The signature of the function is:

```
int elektraLineCheckConf (Key * errorKey, KeySet * conf);
```

The configuration of the plugin is provided as conf. The function may report an error or warnings using the errorKey and the return value.
The following convention was established for the return value of elektraPluginCheckConf:

- 0: The configuration was OK and has not been changed
- 1: The configuration has been changed and now it is OK
- -1: The configuration was not OK and could not be fixed. An error has to be set to errorKey.

The following example demonstrates how to limit the length of the values within the plugin configuration to 3 characters.

```
int elektraLineCheckConf (Key * errorKey, KeySet * conf)
{
        Key * cur;
        ssize_t ksSize = ksGetSize (conf);
        for (elektraCursor it = 0; it < ksSize; ++it)
        {
                cur = ksAtCursor (conf, it);
                const char * value = keyString (cur);
                if (strlen (value) > 3)
                {
                        ELEKTRA_SET_VALIDATION_SYNTACTIC_ERRORF ( errorKey,
                                        "Value '%s' is more than 3 characters long",
                                        value);
                        return -1; // The configuration was not OK and could not be fixed
                }
        }
        return 0; // The configuration was OK and has not been changed
}
```

The elektraPluginCheckConf function is exported via the plugin's contract. The following example demonstrates how to export the checkconf function (see section Contract for further details):

```
keyNew ("system:/elektra/modules/" ELEKTRA_PLUGIN_NAME "/exports/checkconf", KEY_FUNC, elektraLineCheckConf,
       KEY_END);
```

Within the checkconf function all of the plugin configuration values should be validated. Errors should be reported via Elektra's error handling mechanism (see section ELEKTRA*SET*_ERROR for further details). If checkconf encounters a configuration value, that is not strictly invalid but can not be parsed by the plugin (e.g. a parameter which is not part of the plugin configuration), then a warning should be appended to errorKey, using ELEKTRA_ADD_<CONCRETE_TYPE>_WARNING. You also have a ...WARNINGF vararg macro that allows you to substitute parts of the message with variables.

### 475.6.5 <tt>ELEKTRA_PLUGIN_EXPORT</tt>

A function that is always needed in a plugin, is `ELEKTRA_PLUGIN_EXPORT`. This functions is responsible for letting Elektra know that the plugin exists and which methods it implements. The code from the line plugin is a good example and pretty self-explanatory:

```
Plugin *ELEKTRA_PLUGIN_EXPORT
{
        return elektraPluginExport("line",
        ELEKTRA_PLUGIN_GET, &elektraLineGet,
        ELEKTRA_PLUGIN_SET, &elektraLineSet,
        ELEKTRA_PLUGIN_END);
}
```

For further information see the API documentation.

### 475.6.6 <tt>elektraPluginGetGlobalKeySet</tt>

In order to enable communication between plugins which is more complex than what can be done with metadata, Elektra provides a global keyset which plugins can read from and modify.
The keyset is initialized and closed by a KDB handle and can be accessed by all plugins of a single handle except for plugins created manually (e.g. with `elektraPluginOpen`). It is not shared between different KDB handles.
It can be accessed by calling the `elektraPluginGetGlobalKeySet` function, which returns a handle to the global keyset.
Plugins using the global keyset are responsible for cleaning up the parts of the keyset they no longer need.
To make sure there is no collision between plugins, each plugin should use a unique prefix for its keys, e.g. `system:/elektra/<plugin>` for `<plugin>`.
To improve performance, the `cache` plugin also caches parts of the global keyset. If your plugin uses non-cacheable data, you don't have to do anything special. However, if you want your plugin's keys to be cached you should put them below `system:/elektra/cached` (to avoid collisions, use e.g. `system:/elektra/cached/<plugin>` for `<plugin>`). The `cache` plugin also caches keys below `system:/elektra/cache`, but those are reserved for use by the plugin itself.

## 475.7 Note on Direct Method Calls via External Integrations

Some applications want to call Elektra methods directly via native access. A `KeySet` is a data structure over which functions can iterate. If you want to start again from to first element, you have to explicitly call `rewind()` to set the internal pointer to the start. Any plugin expects the passed `KeySet` to be **rewinded**.

## 475.8 Memory Leaks

If you experience memory leaks you may use `valgrind` in order to locate them. If you have analyzed your code, and you know that your code does not contain memory leaks continue reading:
It is possible, that a library the plugin depends on contains some memory leaks which cannot be fixed by you. In such a case you may want to suppress them in order the CI does not fail. In order to suppress them, a few measurements have to be taken:

- Update the plugin contract within the plugins `README.md` by appending `memleak` to `info/status` if not already done
- If the `CMakeLists.txt` does not add the `MEMLEAK` label anywhere (just search for it in the file), append

  ```
  if (ADDTESTING_PHASE)
    include (LibAddTest)
    add_plugintest (replace_with_the_actual_plugin_name MEMLEAK)
  endif (ADDTESTING_PHASE)
  ```

  to the end of the file and remove the `INSTALL_TEST_DATA` label from the `add_plugin(...)` function.
- Add the memleak rules to the `tests/valgrind.suppression` file.

The rules can be obtained through the jenkins pipeline (click on "details" next to the "continuous-integration/jenkins/pr-merge" GitHub check) within the "Tests" tab at the top. There will be expandable items highlighted red. After expanding, they show a verbose output. Just look for the blocks in the following form:

```
{
   <insert_a_suppression_name_here>
   Memcheck:Leak
```

```
   match-leak-kinds: definite
   fun:malloc
   fun:malloc
   fun:resize_scopes
   fun:dl_open_worker_begin
   fun:_dl_catch_exception
   fun:dl_open_worker
   fun:_dl_catch_exception
   fun:_dl_open
   fun:dlopen_doit
   fun:_dl_catch_exception
   fun:_dl_catch_error
   fun:_dlerror_run
   fun:dlopen_implementation
   fun:dlopen@@GLIBC_2.34
   fun:elektraModulesLoad
}
```

and append them to the bottom of the `tests/valgrind.suppression` file and do not forget to give them a clear name.
After committing and pushing, the CI memleak errors should be suppressed.

## 475.9 Further Readings

Read more about:

- contracts
- of how to write a storage plugin

# Chapter 476

# Profiling

## 476.1 Execution Time

One of the primary resources in computing is execution time. To keep usage of this resource type low, it makes sense to profile code and check which code paths in a progamm take the longest time to execute. There exist various tools to handle this kind of profiling. For this tutorial we will use

1. `Callgrind` and the graphical frontend `KCacheGrind/QCacheGrind`, and
2. `XRay` and `FlameGraph` to visualize the data produced by `XRay`

### 476.1.1 Callgrind

#### 476.1.1.1 Choosing the Correct Build Type

Since we want to improve the readability of the Callgrind output we choose a build type that includes debug symbols. The two obvious choices for the build type are:

- `RelWithDebInfo` (optimized build with debug symbols), and
- `Debug` (non-optimized build with debug symbols)

. We use `Debug` here, which should provide the most detailed profiling information.

#### 476.1.1.2 Disabling <tt>dlclose</tt> Calls

For this tutorial we decided to profile the YAJL plugin. Since Elektra loads plugin code via `dlopen` and Callgrind `does not support the function dlclose properly` we remove the `dlclose` calls in the file `dl.c` temporarily. At the time of writing one option to do that is deleting

- a single line `dlclose` statement, and
- an `if`-statement that checks the return value of a `dlclose` call

. An unfortunate effect of this code update is that Elektra will now leak memory when it unloads a plugin. On the other hand, Callgrind will be able to add source code information about the YAJL plugin to the profiling output.

#### 476.1.1.3 Building Elektra

As we already described before we use the `Debug` build type for the profiling run. To make sure we test the actual performance of the YAJL plugin we disable debug code and the logger. The following commands show one option to translate Elektra using this configuration, if we use `Ninja` as build tool:

```
mkdir build
cd build
cmake -GNinja ..               \
      -DCMAKE_BUILD_TYPE=Debug \
      -DENABLE_LOGGER=OFF      \
      -DENABLE_DEBUG=OFF       \
      -DPLUGINS=ALL
ninja
cd .. # Change working directory back to the root of repository
```

#### 476.1.1.4 Profiling the Code

We use the tool `benchmark_plugingetset` to profile the execution time of YAJL. The file `keyframes_complex.json` serves as input file for the plugin. Since `benchmark_plugingetset` requires a data file called `test.$plugin.in`, we save a copy of `keyframes_complex.json` as `test.yajl.in` in the folder `benchmarks/data`:

```
mkdir -p benchmarks/data
cp src/plugins/yajl/yajl/keyframes_complex.json benchmarks/data/test.yajl.in
```

. After that we call `benchmark_plugingetset` directly to make sure that everything works as expected:

```
build/bin/benchmark_plugingetset benchmarks/data user yajl get
```

. If the command above fails with a segmentation fault, then please check

- that the build system included YAJL, and
- that your OS is able to locate the plugin (e.g. append the `lib` directory in the build folder to `LD_LIBRARY_PATH` on Linux)

. If `benchmark_plugingetset` executed successfully, then you can now use Callgrind to profile the command:

```
valgrind --tool=callgrind --callgrind-out-file=callgrind.out \
build/bin/benchmark_plugingetset benchmarks/data user yajl get
```

. The command above will create a file called `callgrind.out` in the root of the repository. You can now remove the input data and the folder `benchmarks/data`:

```
rm benchmarks/data/test.yajl.in
rmdir benchmarks/data
```

. If you use Docker to translate Elektra, then you might want to fix the paths in the file `callgrind.out` before you continue:

```
# The tool `sponge` is part of the `moreutils` package: https://joeyh.name/code/moreutils
sed -E 's~/home/jenkins/workspace/(\.\./)*~~g' callgrind.out | sponge callgrind.out
```

. Now we can analyze the file `callgrind.out` with a graphical tool such as QCacheGrind:

```
qcachegrind&
```

. If everything worked as expected QCacheGrind should open the file `callgrind.out` and display a window that look similar to the one below:

. You can now select different parts of the call graph on the left to check which parts of the code take a long time to execute.

### 476.1.2 XRay

`XRay` is an extension for LLVM that adds profiling code to binaries. Profiling can be dynamically enabled and disabled via the environment variable `XRAY_OPTIONS`.

#### 476.1.2.1 Choosing the Correct Build Type

Since `XRay` currently requires LLVM we need to set the compiler appropriately. We use Clang 8 in our example.

```
export CC=clang-8
export CXX=clang++-8
```

. We enable the static build (`BUILD_STATIC=ON`) and disable the dynamic build (`BUILD_SHARED=OFF`), since `XRay currently does not support dynamic libraries`. To enable Xray we use the compiler switch `-fxray-instrument`. To instrument every function we set the instruction threshold to `1` with `-fxray-instruction-threshold=1`.

```
export REPOSITORY_DIRECTORY="$PWD"
export BUILD_DIRECTORY="$REPOSITORY_DIRECTORY/build"
mkdir -p "$BUILD_DIRECTORY"
cd "$BUILD_DIRECTORY"
cmake -GNinja "$REPOSITORY_DIRECTORY"                                    \
      -DCMAKE_BUILD_TYPE=Release                                         \
      -DBUILD_SHARED=OFF                                                 \
      -DBUILD_STATIC=ON                                                  \
      -DCOMMON_FLAGS='-fxray-instrument -fxray-instruction-threshold=1' \
      -DPLUGINS=ALL
```

We will analyze the YAJL plugin below. Please make sure that the CMake command above includes the plugin:

```
...
-- Include plugin yajl
...
```

. Now we can translate the code with `Ninja` and change the current directory back to the root of the repository:

```
ninja
cd "$REPOSITORY_DIRECTORY"
```

. In the next step we use `benchmark_plugingetset` to execute YAJL for the input file `keyframes_complex.json`. To do that we

1. create the folder `data` in the directory `benchmarks`, and
2. save the file `keyframes_complex.json` as `test.yajl.in`

. The following commands show you how to do that:

```
mkdir -p benchmarks/data
cp src/plugins/yajl/yajl/keyframes_complex.json benchmarks/data/test.yajl.in
```

. Now we first check if running [`benchmark_plugingetset`][] works without instrumentation:

```
"$BUILD_DIRECTORY/bin/benchmark_plugingetset" benchmarks/data user yajl get
```

. If everything worked correctly, then the command above should finish successfully and not produce any output. To instrument the binary we set the environment variable `XRAY_OPTIONS` to the value `xray_mode=xray-basic verbosity=1`.

```
export XRAY_OPTIONS='xray_mode=xray-basic patch_premain=true verbosity=1'
"$BUILD_DIRECTORY/bin/benchmark_plugingetset" benchmarks/data user yajl get
```

. The command above will print the location of the XRay log file to `stdterr`:

```
...
...XRay: Log file in 'xray-log.benchmark_plugingetset.gpcX3t'
...
```

. Now we can use the log file to analyze the runtime of the execution paths of the binary. To do that we first save the name of the log file in the variable `LOGFILE`. This way we do not need to repeat the filename every time in the commands below.

```
LOGFILE=$("$BUILD_DIRECTORY/bin/benchmark_plugingetset" benchmarks/data user yajl get 2>&1 |
          sed -nE "s/.*Log file in '(.*)'.*/\1/p")
```

. To list the 10 functions with the longest runtime we use the command `llvm-xray account`:

```
llvm-xray account "$LOGFILE" -top=10 -sort=sum -sortorder=dsc -instr_map
       "$BUILD_DIRECTORY/bin/benchmark_plugingetset"
#> Functions with latencies: 35
#>    funcid      count [      min,       med,       90p,       99p,       max]       sum  function
#>         1          1 [ 0.004791,  0.004791,  0.004791,  0.004791,  0.004791]  0.004791  <invalid>:0:0:
       main
#>      1333          1 [ 0.002007,  0.002007,  0.002007,  0.002007,  0.002007]  0.002007  <invalid>:0:0:
       elektraYajlGet
#> ...
```

. We can also use the log file to create a `Flame Graph`. To do that we use the `llvm-xray stack` to create an input file for the tool `flamegraph.pl`

```
llvm-xray stack "$LOGFILE" -stack-format=flame -aggregation-type=time -all-stacks \
               -instr_map "$BUILD_DIRECTORY/bin/benchmark_plugingetset" > flamegraph.txt
```

. We then create the Flame Graph with the following command:

```
# Depending on how you installed Flame Graph the executable
# might also be called `flamegraph.pl` instead of `flamegraph`.
flamegraph flamegraph.txt > flamegraph.svg
```

. The image below shows one example how the picture could look like:

. Additional information on how to use the data produced by `XRay` is available `here`.

# Chapter 477

# How-To: Python kdb

## 477.1 Table of Contents



## 477.2 Introduction

When programming in Python it is possible to access the kdb database, changing values of existing keys, adding and deleting keys and a few other things.

## 477.3 Installation

Either `build` the package or install from a repository.

### 477.3.1 Alpine Linux

The `python bindings package` is only available in the testing repository (as of 2019-04-29).

```
docker run -it alpine:edge /bin/sh
echo "https://dl-cdn.alpinelinux.org/alpine/edge/testing" » /etc/apk/repositories
# Install elektra and the python bindings
apk update && apk add elektra elektra-python py3-elektra
```

Under regular alpine, you have to install python3 from the edge repository. If you do not want to add the edge repositories permanently like above, you can do

```
apk add --repository "https://dl-cdn.alpinelinux.org/alpine/edge/main" python3
apk add --repository "https://dl-cdn.alpinelinux.org/alpine/edge/testing" elektra elektra-python py3-elektra
```

### 477.3.2 Debian

The Elektra python-binding is currently built for:

- Debian 11 `bullseye`
- Debian 10 `buster`

```
docker run -it debian:bullseye
apt-get update
apt-get install ca-certificates
apt-get install vim gnupg
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys F26BBE02F3C315A19BF1F791A9A25CC1CC83E839
vim /etc/apt/sources.list
```

Append `deb https://debs.libelektra.org/bullseye bullseye main` to `/etc/apt/sources.list` and install it:

```
apt-get update
apt-get install python3-elektra
```

## 477.4 Import kdb

In order to being able to use `kdb`, you at first need to `import kdb`. You need access to a Python object of `KDB`. This is accomplished by calling `kdb.KDB()` and saving this to a variable because later on this object will be needed for various operations. The easiest way to do this would be:

```
import kdb
with kdb.KDB() as k:
    print("Hello world! I have a kdb-instance! :D")
    # do all kinds of operations explained below
```

## 477.5 Keyset

A keyset is basically a list of the keys that lie within the specified range of the database. When creating an empty keyset this range is obviously zero. It is possible to load the whole database into a keyset but in a lot of cases this is not needed and you can specify which keys exactly you need (which I mean with specified range). At first it is necessary to create a new keyset. When simply calling `kdb.KeySet()` the keyset is of size 0. There is no restriction to the keyset's size. It is possible to specify a certain (maximum) size for a keyset. To load keys into to keyset from the database you simply call the method `get` provided by the kdb-object.

```
import kdb
with kdb.KDB() as k:
    # create empty keyset
    ks = kdb.KeySet()
    print(len(ks))  # should be 0
    # load an existing keyset
    k.get(ks, 'user')
```

It is also possible to iterate as expected over a keyset and use `len`, `reversed` and copying. The elements of a keyset can be accessed by indexes and a keyset can be sliced. Another way of accessing a key is by the key name (`keyset_name['/path/to/keys/key_name']`). `If a key with the given name does not exist within the keyset, a` KeyError` exception is thrown.

An example that shows how to load an existing keyset and then access every key and value of the loaded keyset:

```
import kdb
with kdb.KDB() as k:
    ks = kdb.KeySet()
    # load an existing keyset
    k.get(ks, 'user')
    # if you for some reason want to loop through the keyset last key first
    # use: for key in reversed(ks):
    for key in ks:
        # print for every key in the keyset the key and the value
        print("key: {} value: {}".format(key, key.value))
```

Here an example of how you can easily check if a key exists:

```
import kdb
with kdb.KDB() as k:
    namespace = "user:/"
    path = '{}/test'.format(namespace)
    ks = kdb.KeySet()
    k.get(ks, namespace)
    try:
        print("The value of the key {} is {}!".format(ks[path], ks[path]
                                                      .value))
    except KeyError:
        print("The key {} does not exist!".format(path))
```

Ways of copying a keyset:

```
import copy, kdb
with kdb.KDB() as k:
    ks = kdb.KeySet()
    k.get(ks, 'spec')
    # creating a deep copy
    ks_deepcopy = copy.deepcopy(ks)
    # creating a shallow copy
    ks_shallowcopy = copy.copy(ks)
```

Slicing works just like for normal lists in Python. But be careful: Afterwards the result will be a list - not a keyset.

```
import kdb
with kdb.KDB() as k:
    ks = kdb.KeySet()
    k.get(ks, 'system')
    # creating a shallow copy by slicing
    ks_copy_by_slicing = ks[:]
    # in the following examples start & end need to be replaced by integers
    start, end = 1, 3
    # create keyset with keys from start to end-1
    a = ks[start:end]
    # create keyset with keys from start to the end of the original keyset
    start = len(ks) - 3
    b = ks[start:]
    # create keyset with keys from the beginning of the keyset to end
    c = ks[:end]
    for keyset in [a, b, c]:
        for key in keyset:
            try:
                print('{}: {}'.format(key, key.value))
            except UnicodeDecodeError:
                pass  # Ignore keys and values that store non-ASCII characters
        print('')
```

If you have changed anything in the keyset and want those changes to be saved to the database, you need to call `set` which is just like `get` provided by the kdb-object.

An example of everything up until now could look like this:

```
import kdb
with kdb.KDB() as k:
    ks = kdb.KeySet()
    k.get(ks, '/path/to/keys')
    # Any number of operations that manipulate the keyset as explained above
    # Setting and by doing so saving the new keyset
    k.set(ks, '/path/to/keys')
    # by changing the path here it is for instance possible to set the keyset
    # of one user identical to another users
```

If you have a key a very simple way to get its name and value:

```
import kdb
with kdb.KDB() as k:
    ks = kdb.KeySet()
    k.get(ks, 'user')
    for key in ks:
        print("key name:  {}".format(key.name))
        print("{0}{1}\n{0}{2}\n".format("key value: ", key.value,
                                        ks.lookup(key).string))
```

## 477.6 Keys

It is possible to create new keys:

```
import kdb
with kdb.KDB() as k:
    # the first argument is the path to the key,
    # the third argument is the key's value
    new_key = kdb.Key('/user/sw/pk/key_name', kdb.KEY_VALUE, 'key_value')
    print("{}: {}".format(new_key, new_key.value))
```

You can also duplicate a key:

```
import kdb
with kdb.KDB() as k:
    key1 = kdb.Key('/user/sw/pk/key_name', kdb.KEY_VALUE, 'key_value')
    key2 = kdb.Key(key1.dup())
    print("{}: {}".format(new_key, new_key.value))
    print("{}: {}".format(key2, key2.value))
```

An example for working with meta-keys

```
import kdb
with kdb.KDB() as k:
    key1 = kdb.Key("user:/key1", kdb.KEY_VALUE, "some_value")
    key1.setMeta("foo",     "bar")
    key1.setMeta("owner",   "manuel")
    key1.setMeta("comment/#0", "this is my example key")
    for meta in key1.getMeta():
        print("  key1.{0} = \"{1}\"".format(meta.name, meta.value))
```

Keys can be added to a keyset using `append`. If the key already exists, the value will be updated. Calling 'keyset↩
_name['/path/to/key'] = 'new_value`does not work for updating keys already in a keyset. Keys can be removed with`pop, remove`or`cut`

```
from kdb import KDB, KEY_VALUE, Key, KeySet
class KeySet(KeySet):
    def __repr__(self):
        """Return a textual representation of this keyset."""
        return '\n'.join(
            ['{}: {}'.format(key.name, key.value) for key in self])
def key(name, value):
```

```
    """Create a new key with the given name and value."""
    return Key(name, KEY_VALUE, value)
def describe(keyset, title, newline=True):
    return '{}\n{}\n{}{}'.format(title, '=' * len(title), keyset,
                                 '\n' if newline else '')
with KDB() as data:
    keyset = KeySet()
    data.get(keyset, 'user')
    print(describe(keyset, "Initial Keyset"))
    new_key = key('user:/sw/pk/key_name', 'key_value')
    # adding new_key to the existing key-set,
    # ks['/user/sw/pk/key_name'].value == 'key_value'
    keyset.append(new_key)
    print(describe(keyset, "Add New Key"))
    newer_key = key('user:/sw/pk/key_name', 'other_key_value')
    # adding newer_key to the existing key-set, by doing so replacing new_key,
    # ks['/user/sw/pk/key_name'].value == 'other_key_value'
    keyset.append(newer_key)
    print(describe(keyset, "Replace Key", newline=False))
```

## 477.7 Merging KeySets

The internal three-way merge algorithm is also included in the Python bindings.

```
import kdb, kdb.merge
baseKeys = kdb.KeySet(100,
                    kdb.Key("system:/test/key1", "k1"),
                    kdb.Key("system:/test/key2", "k2"),
                    kdb.KS_END,
                    )
ourKeys = kdb.KeySet(100,
                     kdb.Key("system:/test/key1", "k1"),
                     kdb.Key("system:/test/key3", "k3"),
                     kdb.KS_END,
                     )
theirKeys = kdb.KeySet(100,
                       kdb.Key("system:/test/key1", "k1"),
                       kdb.Key("system:/test/key4", "k4"),
                       kdb.KS_END,
                       )
base = kdb.merge.MergeKeys(baseKeys, kdb.Key("system:/test"))
theirs = kdb.merge.MergeKeys(theirKeys, kdb.Key("system:/test"))
ours = kdb.merge.MergeKeys(ourKeys, kdb.Key("system:/test"))
merger = kdb.merge.Merger()
mergeResult = merger.merge(base, ours, theirs, kdb.Key("system:/test"), kdb.merge.ConflictStrategy.THEIR)
# prints ['system:/test/key1', 'system:/test/key3', 'system:/test/key4']
print(mergeResult.mergedKeys)
```

# Chapter 478

# README

Elektra has many different aspects to explore. Not all parts are needed by everyone. In this document we classify which parts should be read by whom.

### 478.0.1 General Information

Read this first to get the basic concepts of Elektra.

- Namespaces
- Key names
- Cascading
- Arrays
- Mount Configuration Files

### 478.0.2 Developers

For these tutorials we assume you want to elektrify your application, that means, you want your application to participate in the global key database Elektra provides.

- Hello, Elektra in C
- Integration of your C Application
- Writing a specification for your configuration
- `Meta specification language`
- Plugins Introduction
- Storage Plugins
- Compilation Variants of plugins (advanced topic)
- High Level API
- Command Line Options
- Python Bindings
- Java Bindings
- Java Plugins
- Ruby Bindings
- High Level API Bindings
- Notifications
- Changetracking

### 478.0.3 System Administrators

For these tutorials we assume that you want to work with the configuration of applications already somehow integrated with Elektra.

- Import Configuration
- Export Configuration
- Intercept Environment
- Intercept File System
- Merge Configuration (new version)
- Merge Configuration (deprecated)
- Validate Configuration
- Encrypt Configuration
- Install Configuration Files
- Write a specification for dockerd
- Recording Changes
- Configure Xfconf Applications

### 478.0.4 Elektra Developers

These tutorials are for persons that want to contribute to Elektra:

- Contributing with CLion
- Contributing with Visual Studio (Windows)
- Run all Tests with Docker
- Run Reformatting with Docker
- Using Podman instead of Docker
- Language Bindings
- Code Generator
- Benchmarking
- Profiling
- Logging
- Changetracking

### 478.0.5 Installation Manuals

These tutorials provide additional information on how to install and set up specific tools.

- ODBC Backend
- Webui

# Chapter 479

# Recording Changes to the KDB

Elektra provides a powerful session recording feature. You can control it through the `kdb` command-line utility. Session recording will track all the changes done to the KDB while it is enabled. This includes changes done by elektrified applications themselves too, not only changes done through the `kdb` utility. The recorded changes can be exported, undone or used for auditing purposes.
There are seven commands to interact with the session recording feature:

- `kdb record-start`: starts recording. If there are existing recorded changes, they will NOT be removed. New changes will be appended.
- `kdb record-stop`: stops recording.
- `kdb record-reset`: removes all recorded changes.
- `kdb record-export`: export the changes.
- `kdb record-undo`: undo everything that has been recorded.
- `kdb record-state`: show information about the state of session recording. This includes information about what keys will be recorded and which changes have already been made to the KDB.
- `kdb record-rm`: Remove specific keys from the recording.

For a more detailed description and options for each command, please take a look at their respective man pages.

## 479.1 A Simple Recording Session

```
# Seed some data first
kdb set user:/test/name Franz
kdb set user:/test/color Red
# Enable session recording for all keys below user:/
kdb record-start user:/
# RET:0
kdb set user:/test/name Hans
kdb set user:/test/age 29
kdb rm user:/test/color
# View the current state of session recording
kdb record-state
#> Recording is active for user:/↩↩Added 1 key(s)↩Modified 1 key(s)↩Removed 1 key(s)↩↩Added key
       user:/test/age↩Modified key user:/test/name↩Removed key user:/test/color
# Stop session recording again
kdb record-stop
# RET:0
```

## 479.2 Exporting Recorded changes

One of the most powerful features of session recording is the ability to export the recorded changes. This allows you to apply the specific modifications on other computers, without having to export and overwrite the complete configuration.

```
kdb record-export [<source>] [<format>]
```

Where `source` and `format` are optional parameters. The `source` parameter lets you specify which keys you want to export. By default, this is all keys, or `/`. The `format` parameter lets you specify which format you want to

export the changes in. By default, we use the `ansible` format. Note that the format is just the name of the Elektra storage plugin you want to use.
You can also use the `-c` option to specify a list of configuration parameters for the storage plugin. See the README of the plugin for more details about which parameters are supported.

```
# Export recorded changes into an Ansible playbook
kdb record-export / ansible -c playbook/name="Recording Tutorial"
# RET:0
---
- name: Recording Tutorial
  hosts: all
  collections:
    - elektra_initiative.libelektra
  tasks:
    - name: Set Elektra Keys
      elektra:
        keys:
          - user:
              test:
                age:
                  - value: 29
                color:
                  - remove: true
                name:
                  - value: Hans
```

# Chapter 480

# Introduction

Running all the tests like the build server requires multiple dependencies. To overcome this problem, instead of trying to install all the necessary dependencies on your own, an appropriate Docker image can be used. This way you can easily and quickly run all the tests.

## 480.1 Who Is This Guide For?

For anyone who wants to run all the tests, like it is done by the build server.
This is a step-by-step guide. Just follow the steps and you are good to go!

## 480.2 Prerequisites

- Docker for Linux containers has to be pre-installed. Please refer to https://docs.docker.com/install/ if you haven't installed it yet. Your host OS can be either Linux, macOS or Windows. Alternatively, you can use the Podman container engine, see Using Podman instead of Docker.
- Basic knowledge of Docker (not mandatory)

## 480.3 Podman support

Alternatively, you can use podman, a different container engine which is compatible with Docker. See https://podman.io/ for more details and an installation guide. If you are using podman, and want to follow this tutorial, just replace the docker command with podman.

## 480.4 What to Begin With?

### 480.4.1 1. Docker Image

To build your own Docker image, run the following command from the source root directory:

```
docker build -t buildelektra-bullseye \
--build-arg JENKINS_USERID=$(id -u) \
--build-arg JENKINS_GROUPID=$(id -g) \
-f scripts/docker/debian/bullseye/Dockerfile \
scripts/docker/debian/bullseye/
# RET: 0
```

The build process depends on your Internet connection speed and the overall performance of your hardware. Most likely, it will take at least 5 minutes. Please be patient. Once you have built the image, you can reuse it multiple times.
The image tag `buildelektra-bullseye` we suggested can be replaced by a name of your own choosing.
Another alternative but not recommended(!) option would be to pick one of the publicly available Docker images of Elektra. If you do not know the difference, just pick this one --> "build-elektra-debian-stretch". Unfortunately, it will take some time to download it, since it is pretty big, but you can be sure you'll have all the needed dependencies. You can choose a light-weight Alpine image which won't take long to download, however it is not recommended. This image does not contain all necessary dependencies.

If you want to view all the available images, execute this command:
```
docker run --rm anoxis/registry-cli -r https://hub-public.libelektra.org
```
You will see something like this:
```
--------------------------------
Image: build-elektra-debian-stretch
  tag: 201906-ecf9161f41a8b472b3b0282a85a9f91d1f0f45357756e5451ae043fce8d0100e
  tag: 201902-b6d49f470e1171348248b2f87ef397d58d7a2dae14d201f4073564079ce0c070
  tag: 201903-b95dc56352aa684e16dfb8628bded4c69c712223f5d7ed99ebdd644852a32123
  tag: 201905-ecf9161f41a8b472b3b0282a85a9f91d1f0f45357756e5451ae043fce8d0100e
  tag: 201901-6b08855f13ba26e3ad1fa80e399b87df860cc24889f2d1854fa0050834567b26
  tag: 201904-ecf9161f41a8b472b3b0282a85a9f91d1f0f45357756e5451ae043fce8d0100e
  tag: 201904-1a6be7b9c3740a2338b14d08c757332cae5254ce58219b6cc2908c7bd6e4f460
  tag: 201903-1a6be7b9c3740a2338b14d08c757332cae5254ce58219b6cc2908c7bd6e4f460
  tag: 201903-6b08855f13ba26e3ad1fa80e399b87df860cc24889f2d1854fa0050834567b26
  tag: 201902-6b08855f13ba26e3ad1fa80e399b87df860cc24889f2d1854fa0050834567b26
```
Afterwards pull your desired image as you would do from any public registry:
```
docker pull hub-public.libelektra.org/<image_name>:<tag_name>
```
Example:
```
docker pull
       hub-public.libelektra.org/build-elektra-debian-bullseye:202212-96dfa4c7e15462369375db000361ff9bf71076c7bfe27464eef18d0
```

### 480.4.2 2. Run the Docker Container

You have to be in the root of the source directory of the Elektra Initiative, so that the container can properly map all the source files.
So from your root source folder run the following:
```
docker run -it --rm \
-v "$PWD:/home/jenkins/workspace" \
-w /home/jenkins/workspace \
buildelektra-bullseye
```

### 480.4.3 3. Build

After starting the container, you should be automatically inside it in the working directory `/home/jenkins/workspace`.
Create a folder where Elektra will be installed, create another folder for building the source and `cd` to it and like this:
```
mkdir elektra-install && mkdir elektra-build-docker && cd elektra-build-docker
```
Build it with
```
cmake /home/jenkins/workspace \
-DBINDINGS="ALL;-DEPRECATED" \
-DPLUGINS="ALL;-DEPRECATED" \
-DTOOLS="ALL" \
-DENABLE_DEBUG="ON" \
-DKDB_DB_HOME="/home/jenkins/workspace/elektra-build-docker/.config/kdb/home" \
-DKDB_DB_SYSTEM="/home/jenkins/workspace/elektra-build-docker/.config/kdb/system" \
-DKDB_DB_SPEC="/home/jenkins/workspace/elektra-build-docker/.config/kdb/spec" \
-DBUILD_DOCUMENTATION="OFF" \
-DCMAKE_RULE_MESSAGES="OFF" \
-DCMAKE_INSTALL_PREFIX="/home/jenkins/workspace/elektra-install" \
-DCOMMON_FLAGS="-Werror"
```
and then with
```
make -j 10
```
The number 10 can be changed as follows: number of supported simultaneous threads by your CPU + 2. But don't worry, this can only affect the speed of the building, it cannot really break it.
Additionally, you may want to install Elektra inside the container, because the installed tests rely on it. The build server also does this and it is possible that the installed tests have different results (e.g. if test data is missing) Just run this command:
```
make install
```
After Elektra has been installed we need to add it to the PATH variable, meaning you and the tests can interact with Elektra by typing/executing `kdb` in the command line.
```
export PATH="/home/jenkins/workspace/elektra-install/bin:$PATH"
export LD_LIBRARY_PATH="/home/jenkins/workspace/elektra-install/lib:$LD_LIBRARY_PATH"
export LUA_CPATH="/home/jenkins/workspace/elektra-install/lib/lua/5.2/?.so;"
```

### 480.4.4 4. Run Tests

Finally, run the tests. There are two sets of tests. Run the first one with this command:
```
make run_all
```
For the second set to run, remember to execute `make install` from the previous step. Run the second set with this command:
```
kdb run_all
```

# Chapter 481

# Introduction

Running the reformat-all script requires multiple dependencies. To overcome this problem, instead of trying to install all the necessary dependencies on your own, you can easily build a Docker image and run the script inside a Docker container based on this image.

## 481.1 Who Is This Guide For?

Do you want to run the reformat-all and the fix-spelling script easily and without any hassle? You've come to the right place.
This is a step-by-step guide. Just follow the steps and you are good to go!

## 481.2 Prerequisites

- Docker for Linux containers has to be preinstalled. Please refer to `https://docs.docker.com/install/` if you haven't installed it yet. Your host OS should better be Linux, because we are going to use your current Linux user ID for building the image. Alternatively, you can use the Podman container engine, see Using Podman instead of Docker.
- Basic knowledge of Docker (not mandatory)

## 481.3 What to Begin With?

### 481.3.1 1. Build Your Own Docker Image

Now you are going to build your own Docker image based on Debian.
From the source root directory run the following command:

```
docker build -t buildelektra-sid \
        --build-arg JENKINS_USERID=$(id -u) \
        --build-arg JENKINS_GROUPID=$(id -g) \
        -f scripts/docker/debian/sid/Dockerfile \
        scripts/docker/debian/sid/
```

The build process depends on your Internet connection speed and the overall performance of your hardware. Most likely, it will take at least 5 minutes. Please be patient. Once you have built the image, you can reuse it multiple times.
The image tag `buildelektra-sid` we suggested can be replaced by a name of your own choosing. Note that currently our `sid` image is the only one with all the necessary tools installed.

### 481.3.2 2. Run the Docker Container

After you built the image, you can execute a container like this:

```
docker run -it --rm \
        -v "$PWD:/home/jenkins/workspace" \
        -w /home/jenkins/workspace \
        buildelektra-sid
```

Again, if you changed the image tag `buildelektra-sid`, please change it above as well.

### 481.3.3 3. Running the Script

After starting the container, you should be automatically inside it in the working directory `/home/jenkins/workspace`.
Now run the script:

```
scripts/dev/reformat-all
```

All your files should be reformatted afterwards.
To also fix spelling errors in your files, run the script:

```
scripts/dev/fix-spelling
```

All your spelling errors should be fixed afterwards.

# Chapter 482

# How to Write a Specification in Elektra

## 482.1 Overview

### 482.1.1 Introduction

In this tutorial you will learn how to interactively use the `SpecElektra` specification language and `kdb` to write a configuration specification for an example application.

### 482.1.2 What you should already know

- `how to install Elektra`
- basic Elektra commands and concepts (`kdb get`, `kdb set`, `kdb ls`)
- how to open and use a terminal

### 482.1.3 What you'll learn

- how to create and mount a specification using `kdb`
- how to add keys with different types, defaults and examples to your specification and how to validate them
- the benefits of using `kdb` to generate a specification, instead of writing one by hand
- how to use the final specification during the installation process of applications

### 482.1.4 What you'll do

- use `kdb` to create and mount a specification for an example CRUD (Create, Read, Update, Delete) application
- define defaults, examples and checks for keys in the validation
- use the specification as a starting point for customizing the configuration of installed applications

## 482.2 Example App Overview

For this tutorial you will write a specification for a simple CRUD backend application. You need to configure a `port` and a `secure` property, that toggles SSL usage, for the REST server. An `ip` and a SQL `dialect` for the database server the app will connect to and finally a `date` where all the data will be saved to a backup.
So the application will need the following configuration options:

- a server port
- server secure
- a database ip
- a database dialect
- a backup date

## 482.3 Getting Started

Make sure you have Elektra installed on your local machine:

```
kdb --version
# KDB_VERSION: 0.9.9
# SO_VERSION: 5
```

Otherwise refer to the getting started guide to install it.

## 482.4 Mounting the Specification

### 482.4.1 Step 1: Mount a Specification File

First you need to mount a specification file, in this case spec.ni to the spec:/ namespace. You can define the path inside the spec:/ namespace as /tests/sw/org/app/#0/current, refer to the documentation to find out more about constructing the name.
You will be using the profile current, you can find out more about profiles in the documentation as well.
We will be writing values mostly to the user:/ namespace. If you want to learn more about namespaces in general, refer to the the documentation on namespaces
You also need to specify the plugin you will use for writing to the file in the correct format. In this case you can choose the ni plugin to write to the specification file.

```
sudo kdb mount `pwd`/spec.ni spec:/tests/sw/org/app/\#0/current ni
```

> **_Attention_**: Mounting the specification by supplying an absolute path (like in the previous example with `pwd`) is only recommended for defining the specification in the first place. It is not recommended when mounting the final specification for usage with the application, especially not in production environments!
>
> Please read the section Using the specification at the end of this document for further information.

Using the command below you can get the location of the concrete file that is used by Elektra.

```
kdb file spec:/tests/sw/org/app/\#0/current
# /current/working/directory/spec.ni
```

### 482.4.2 Step 2: Define a mountpoint

Next you can define, that this specification uses a specific mountpoint for a concrete application configuration. So you can say the concrete configuration should be written to app.ni.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current mountpoint app.ni
```

Your spec.ni file should now look something like this:

```
cat $(kdb file spec:/tests/sw/org/app/\#0/current)
# ;Nil
# ; Generated by the ni plugin using Elektra (see libelektra.org).
# =
# []
#  meta:/mountpoint = app.ni
```

### 482.4.3 Step 3: Do a specification mount

```
sudo kdb spec-mount /tests/sw/org/app/\#0/current ni
```

This specification mount makes sure that the paths where the concrete configuration should be (app.ni) are ready to fulfill our specification (spec.ni). Be aware that different files get mounted for different namespaces. You've a specification file (spec.ni) for the spec-namespace and three files (app.ni) on different locations for the dir-user- and system-namespaces.
You can see the files by providing the namespace as prefix to the kdb file command:

```
kdb file system:/tests/sw/org/app/#0/current
# /etc/kdb/app.ni
kdb file user:/tests/sw/org/app/#0/current
# /home/user/.config/app.ni
kdb file dir:/tests/sw/org/app/#0/current
# /current/working/directory/.dir/app.ni
```

> **_Note_**: The files only exist, when configuration values are stored there, i.e. they are created on the first kdb set and removed with the last kdb rm.

For more information about namespaces in Elektra please see here, a tutorial about the topic is available here.

## 482.5 Adding your first key to the specification

### 482.5.1 Step 1: Adding the server port

The first key you will add to your specification is the port of the server. You add it by using the following command:

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/port type unsigned_short
```

What you also specified in the command above is the type of the configuration value. Elektra uses the CORBA type system and will check if values conform to the type specified.

So after adding the initial key, your specification should look something like this:

```
cat $(kdb file spec:/tests/sw/org/app/\#0/current)
# ;Ni1
# ; Generated by the ni plugin using Elektra (see libelektra.org).
#  =
# server/port =
# []
#  meta:/mountpoint = app.ni
# [server/port]
#  meta:/type = unsigned_short
```

### 482.5.2 Step 2: Adding more metadata

So with your first key added, you of course want to specify more information for the port. There surely is more information to a port than just the type. What about a `default`, or what about an `example` for a usable port? Maybe a `description` what the port really is for? Let's add that next!

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/port default 8080
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/port example 8080
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/port description "port of the REST server that runs
      the application"
```

Beautiful! Your specification is starting to look like something useful. But wait! Shouldn't a port just use values between `1` and `65535`?

Of course Elektra also has a plugin for that. You can just use the `network` checker plugin.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/port check/port ”
```

Now we define the plugins that we want to load. In our example we need the `ni`-Plugin for reading and writing the configuration files, the `type`-plugin for validating data types and the `network`-plugin for validating port numbers.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current infos/plugins "ni type network"
```

Nice! You just have to do one more thing when using a new plugin. Elektra needs to remount the spec to use the new plugin. Use the command from before:

```
sudo kdb spec-mount /tests/sw/org/app/\#0/current
```

Your final specification after adding the port should now look something like this:

```
cat $(kdb file spec:/tests/sw/org/app/\#0/current)
# ;Ni1
# ; Generated by the ni plugin using Elektra (see libelektra.org).
#  =
# server/port =
# []
#  meta:/mountpoint = app.ni
#  meta:/infos/plugins = ni type network
# [server/port]
#  meta:/check/port =
#  meta:/type = unsigned_short
#  meta:/example = 8080
#  meta:/description = port of the REST server that runs the application
#  meta:/default = 8080
```

You can now try to read the value of the newly created configuration. Since you did not set the value to anything yet, you will get the default value back.

```
kdb get /tests/sw/org/app/\#0/current/server/port
#> 8080
```

Try to set the port to `65536` now.

```
kdb set user:/tests/sw/org/app/\#0/current/server/port 65536
# RET: 5
#  Sorry, 1 warning was issued ;(
#  1: Module network issued the warning C03200:
#       Validation Semantic: Port 65536 on key user:/tests/sw/org/app/#0/current/server/port was not within
      0 - 65535
# Sorry, module type issued the error C03200:
# Validation Semantic: The type ’unsigned_short’ failed to match for
      ’user:/tests/sw/org/app/#0/current/server/port’ with string ’65536’
```

Did it work? I hope not. The validation plugins you specified will now correctly validate the port you enter and give you an error.

In this example, the `network` plugin and the `type` plugin are emitting warnings or errors, because the valid ranges for `port` and `unsigned_short` are the same.

### 482.5.3 Step 3: Adding boolean keys

Next up you will configure the `secure` property of our server. This boolean key will toggle if your server encrypts the communication via SSL.
So we will add the key and some metadata for it:

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/secure type boolean
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/secure default 1
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/secure example 0
kdb meta-set spec:/tests/sw/org/app/\#0/current/server/secure description "true if the REST server uses SSL
      for communication"
```

By default the `type` plugin will normalize boolean values when setting them, before storing them. This only works for the concrete config, so when setting the values for the spec you have to use the unnormalized values. In the case it uses `1` for boolean `true` and `0` for boolean `false`.
Since the key `/sw/org/app/\#0/current/server/secure` has a default value of `1`, we are able to retrieve the default value from the key database:

```
kdb get /tests/sw/org/app/\#0/current/server/secure
#> 1
```

You can read more about this in the documentation for the `type plugin`.

## 482.6 Adding the database keys to the specification

### 482.6.1 Step 1: Adding the database ip

Next up you will add a key for the database `ip` address. Like with the key before, you will add a `type`, `default`, `example` and a `description` so that the configuration will be easily usable.
Don't forget the most important rule of configurations: **Always add sensible defaults!**
Now let's try something different. What if you change the file manually? Will Elektra pick up on the changes? And save you from writing **a lot** of `kdb` commands?
*of course*
So just open your file using good old `vim` and add the following lines to specify configuration for the `ip` address.

```
vim $(kdb file spec:/tests/sw/org/app/\#0/current)
# ;Ni1
# ; Generated by the ni plugin using Elektra (see libelektra.org).
database/ip =
#  =
# server/port =
# server/secure =
[database/ip]
 meta:/check/ipaddr =
 meta:/type = string
 meta:/example = 127.0.0.1
 meta:/description = ip address of the database server, that the application will connect to
 meta:/default = 127.0.0.1
# []
#  meta:/mountpoint = app.ni
#  meta:/infos/plugins = ni type network
# [server/port]
#  meta:/check/port =
#  meta:/type = unsigned_short
#  meta:/example = 8080
#  meta:/description = port of the REST server that runs the application
#  meta:/default = 8080
# [server/secure]
#  meta:/type = boolean
#  meta:/example = 0
#  meta:/description = true if the REST server uses SSL for communication
#  meta:/default = 1
```

Alternatively you can of course use `kdb` again to set the configuration values that way. Here are the commands to do that.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/ip type string
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/ip default 127.0.0.1
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/ip example 127.0.0.1
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/ip description "ip address of the database server,
      that the application will connect to"
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/ip check/ipaddr ”
```

### 482.6.2 Step 2: Adding the database dialect

Next up you will add a key for the SQL `dialect` the database will use. Since there are only a few databases your application will support, you can define the possible dialects via an `enum` type. This allows us to prohibit all other possible dialects that are not SQL.
First you define the size of the `enum` type, and then you can add the different `enum` values.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect type enum
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect check/enum "#4"
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect check/enum/\#0 postgresql
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect check/enum/\#1 mysql
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect check/enum/\#2 mssql
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect check/enum/\#3 mariadb
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect check/enum/\#4 sqlite
```

Afterwards you define all the other parameters, just as before.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect default sqlite
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect example mysql
kdb meta-set spec:/tests/sw/org/app/\#0/current/database/dialect description "SQL dialect of the database
       server, that the application will connect to"
```

After this meta-setting bonanza your specification file should look something like this:

```
cat $(kdb file spec:/tests/sw/org/app/\#0/current)
# ;Ni1
# ; Generated by the ni plugin using Elektra (see libelektra.org).
# database/ip =
#  =
# server/port =
# server/secure =
# database/dialect =
# [database/ip]
#  meta:/check/ipaddr =
#  meta:/type = string
#  meta:/example = 127.0.0.1
#  meta:/description = ip address of the database server, that the application will connect to
#  meta:/default = 127.0.0.1
# []
#  meta:/mountpoint = app.ni
#  meta:/infos/plugins = ni type network
# [server/port]
#  meta:/check/port =
#  meta:/type = unsigned_short
#  meta:/example = 8080
#  meta:/description = port of the REST server that runs the application
#  meta:/default = 8080
# [server/secure]
#  meta:/type = boolean
#  meta:/example = 0
#  meta:/description = true if the REST server uses SSL for communication
#  meta:/default = 1
# [database/dialect]
#  meta:/check/enum/#2 = mssql
#  meta:/check/enum/\#0 = postgresql
#  meta:/type = enum
#  meta:/check/enum/#1 = mysql
#  meta:/example = mysql
#  meta:/description = SQL dialect of the database server, that the application will connect to
#  meta:/check/enum/#4 = sqlite
#  meta:/check/enum/#3 = mariadb
#  meta:/default = sqlite
#  meta:/check/enum = #4
```

## 482.7 Adding the backup date

The last key you will add to our specification is a `date` key for the annual backup and restart (this should probably not be annually in a real application). Here you use the `check/date` plugin with the `ISO8601` format. You also specify a `check/date/format`. You can find all possible date formats on the `plugin page`. For this you can use the following commands:

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/backup/date type string
kdb meta-set spec:/tests/sw/org/app/\#0/current/backup/date check/date ISO8601
kdb meta-set spec:/tests/sw/org/app/\#0/current/backup/date check/date/format "calendardate complete
       extended"
```

Then just add examples, defaults and description as always.

```
kdb meta-set spec:/tests/sw/org/app/\#0/current/backup/date default 2021-11-01
kdb meta-set spec:/tests/sw/org/app/\#0/current/backup/date example 2021-01-12
kdb meta-set spec:/tests/sw/org/app/\#0/current/backup/date description "date of the annual server and
       database backup"
```

Now we add the validation plugin for dates and remount the specification:

```
kdb meta-set spec:/tests/sw/org/app/\#0/current infos/plugins "ni type network date"
sudo kdb spec-mount /tests/sw/org/app/\#0/current
```

If we try to add a value that is not in the specified format, an error should get emitted.

```
kdb set user:/tests/sw/org/app/\#0/current/backup/date "03.04.2022"
# RET: 5
# Sorry, module date issued the error C03100:
# Validation Syntactic: Date '03.04.2022' doesn't match iso specification calendardate complete extended
```

To double-check if things are correct, we try to get the value from the `user`-namespace and via cascading lookup.

```
kdb get user:/tests/sw/org/app/\#0/current/backup/date
# RET: 11
# STDERR: Did not find key 'user:/tests/sw/org/app/#0/current/backup/date'
```

```
kdb get /tests/sw/org/app/\#0/current/backup/date
#> 2021-11-01
```

As expected, no value was written to the `user`-namespace and a cascading lookup returns the date that was given as default value in the specification. If we now use the correct format, the new date should be stored in the `user`-namespace and retrieved with both `kdb get` lookups.

```
kdb set user:/tests/sw/org/app/\#0/current/backup/date "2022-04-03"
#> Create a new key user:/tests/sw/org/app/#0/current/backup/date with string "2022-04-03"
kdb get user:/tests/sw/org/app/\#0/current/backup/date
#> 2022-04-03
kdb get /tests/sw/org/app/\#0/current/backup/date
#> 2022-04-03
```

If we explicitly query the `system`-namespace, no key is found.

```
kdb get system:/tests/sw/org/app/\#0/current/backup/date
# RET: 11
# STDERR: Did not find key 'system:/tests/sw/org/app/#0/current/backup/date'
```

## 482.8 Final specification code

Your specification should be complete now! After adding all the keys that are necessary for our application, your specification should look something like this:

```
cat $(kdb file spec:/tests/sw/org/app/\#0/current)
# ;Ni1
# ; Generated by the ni plugin using Elektra (see libelektra.org).
# backup/date =
# database/ip =
#  =
# server/port =
# server/secure =
# database/dialect =
# [backup/date]
#  meta:/check/date/format = calendardate complete extended
#  meta:/type = string
#  meta:/example = 2021-01-12
#  meta:/description = date of the annual server and database backup
#  meta:/default = 2021-11-01
#  meta:/check/date = ISO8601
# [database/ip]
#  meta:/check/ipaddr =
#  meta:/type = string
#  meta:/example = 127.0.0.1
#  meta:/description = ip address of the database server, that the application will connect to
#  meta:/default = 127.0.0.1
# []
#  meta:/mountpoint = app.ni
#  meta:/infos/plugins = ni type network date
# [server/port]
#  meta:/check/port =
#  meta:/type = unsigned_short
#  meta:/example = 8080
#  meta:/description = port of the REST server that runs the application
#  meta:/default = 8080
# [server/secure]
#  meta:/type = boolean
#  meta:/example = 0
#  meta:/description = true if the REST server uses SSL for communication
#  meta:/default = 1
# [database/dialect]
#  meta:/check/enum/#2 = mssql
#  meta:/check/enum/\#0 = postgresql
#  meta:/type = enum
#  meta:/check/enum/#1 = mysql
#  meta:/example = mysql
#  meta:/description = SQL dialect of the database server, that the application will connect to
#  meta:/check/enum/#4 = sqlite
#  meta:/check/enum/#3 = mariadb
#  meta:/default = sqlite
#  meta:/check/enum = #4
```

## 482.9 Using the specification

Now, after you've finished your specification and want to use it for daily business, some aspects have to be considered. A specification usually is written to cover the most common use cases and to provide sensible defaults. In some cases, the administrator may want to change the specification, e.g. for extending it or introducing further restrictions.

In such cases, **directly changing** the specification that was designed for and delivered with the application, is **not** the **recommended** approach.

The recommended way is making a **copy** of the default specification while installing the application and then saving changes to that copy. If misconfiguration occurs, you can easily look at the default specification or reapply it to start over.
If you mount the specification with an absolute path (e.g. by using `` `pwd` `` in scripts), like it was done for defining the specification with `kdb`, the file gets changed directly. If `` `pwd` `` refers to the installation directory of the application, this would be totally fine, but if this is done during development and `` `pwd` `` refers to the source directory, the source specification would be modified via `kdb set spec:/...` calls.
First we have to unmount our original configuration file we just created:
```
sudo kdb umount spec:/tests/sw/org/app/\#0/current
```
The **recommended way** to apply the specification is:
```
# choose a unique filename for your application instead of spec.ni
sudo kdb mount spec.ni spec:/tests/sw/org/app/\#0/current ni
sudo kdb import spec:/tests/sw/org/app/\#0/current ni ./spec.ni
```
Because we used a relative path (not starting with `/`), Elektra will use a file within the `spec` directory (`/usr/share/elektra/specification` by default) for storing the specification.
The `kdb import` command just tells Elektra to load the file `./spec.ni` with the `ni` plugin and write it into `spec:/tests/sw/org/app/\#0/current`. Using a separate `kdb import` also means we could use a different storage plugin for mounting than what `./spec.ni` uses.
Another alternative is copying the file manually:
```
sudo kdb mount spec.ni spec:/tests/sw/org/app/\#0/current ni
sudo cp ./spec.ni $(kdb file spec:/tests/sw/org/app/\#0/current)
```
This works like the previous snippet, except that we directly modify the spec file without going through Elektra. For very big specifications this might be faster, but we get no validation that the file is actually readable and `./spec.ni` has to be readable by the storage plugin used with `kdb mount`.
Finally, in some cases you may want to control where the mounted spec file is stored (e.g. if you are updating a legacy application that has an existing configuration directory). In those cases using an absolute path is fine:
```
kdb mount /etc/spec.ni spec:/tests/sw/org/app/\#0/current ni
kdb import spec:/tests/sw/org/app/\#0/current ni ./my-spec.ini
```
or
```
sudo kdb mount /etc/spec.ni spec:/tests/sw/org/app/\#0/current ni
sudo cp ./spec.ni $(kdb file spec:/tests/sw/org/app/\#0/current)
```
Please note that also in these examples, the file referred to by the absolute path `/etc/spec.ni` is a **new file** where the content of the file `./spec.ni` in the working directory gets imported or copied to.

## 482.10 Cleanup

If you want to remove the files that were created in the course of the tutorial, the following steps are necessary.
```
sudo rm -v `kdb file spec:/tests/sw/org/app/#0/current`
rm -v `kdb file user:/tests/sw/org/app/#0/current`
sudo rm -v /usr/share/elektra/specification/spec.ni
sudo rm -v /etc/spec.ni
```
If you take a look at `kdb mount`, you'll see that there are currently two mountpoints open.
Mountpoints are meant to mount (external) files into the key database structure of Elektra. This mechanism is similar to `mount` on Linux: changes made to the key database will be written to the underlying mounted file. If you want to learn more on mounting and mountpoints in Eletra, refer to the documentation.
To round up this tutorial, we will `kdb umount` these two mountpoints:
```
rm -v ./spec.ni
sudo kdb umount /tests/sw/org/app/#0/current
sudo kdb umount spec:/tests/sw/org/app/#0/current
```
In case something went wrong and you want to reset the whole content of your kdb, please refer to the man page of kdb reset.

## 482.11 Summary

- You set up and mounted a specification using `kdb mount` and `kdb spec-mount`.
- You added keys to the specification using `kdb meta-set`.
- You added different types of keys with `type string`, `type boolean` or `type unsigned_short`.
- You added keys with `enum types`, to restrict specific configuration settings to a defined set of possible values.
- You added default parameters, examples and descriptions with `example`, `default`, `description`.

- You also added validation checks using different plugins, like `check/port` or `check/date`.
- You know how to use the specification for installed applications (esp. in production environments).

## 482.12 Learn more

- `Tutorial Overview`

# Chapter 483

# How-To: Write a (Well Behaved) Storage Plugin

The plugin tutorial already covers some of the most interesting parts on how to write a (storage) plugin. This text will tell you a little bit more about how a storage plugin should act. While it is usually relatively easy to create a plugin that stores basic key-value pairs, adding advanced features such as support for

- arrays, and
- metadata,

takes more work. Before you continue with this text, please make sure that you read all of the linked documents above.

## 483.1 Don't Add Additional Keys

One common problem of storage plugins is, that they store too many keys. For example, if the user adds the keys

- `user:/tests/storage/root` and
- `user:/tests/storage/root/level1/level2/level3`,

then your plugin should only store those two keys. **Do not** add the keys

- `user:/tests/storage/root/level1`, or
- `user:/tests/storage/root/level1/level2`

to the key set. One plugin that handles this situation properly is YAML CPP, as the following `Markdown Shell Recorder` test shows:

```
# Mount plugin
sudo kdb mount config.yaml user:/tests/storage yamlcpp
# Add key-value pairs
kdb set user:/tests/storage/root cockerel
kdb set user:/tests/storage/root/level1/level2/level3 hatching chick
# Make sure that YAML CPP did not store any additional keys
kdb ls user:/tests/storage/root
#> user:/tests/storage/root
#> user:/tests/storage/root/level1/level2/level3
# Undo modifications to the key database
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

. For more information on why we allow "holes" in the hierarchy, please take a look here.

## 483.2 Differentiate Between Empty Keys and Keys Containing an Empty String

Elektra supports both binary and textual values. The main difference between binary and textual data is that textual data always ends with a null byte. Therefore you are not allowed to store the code point `0` inside textual data. Binary data does not have this limitation.

The simplest textual data is the empty string ("" = 0) and has length 1, while the simplest binary data stores nothing at all and therefore has length 0. In the kdb utility you can disambiguate between these value by checking for the metakey `binary`. The following Markdown Shell Recorder test shows how a storage plugin should handle empty values.

```
# Mount plugin
sudo kdb mount config.yaml user:/tests/storage yamlcpp
kdb set user:/tests/storage/empty ”
#> Create a new key user:/tests/storage/empty with string ""
kdb get user:/tests/storage/empty
#>
kdb meta-ls user:/tests/storage/empty
#>
# Undo modifications to the key database
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

## 483.3 Convert Boolean Data

Elektra uses `0` and `1` to represent binary data. A storage plugin that uses other values (e.g. false and true) needs to convert these values to 0 and 1. The Markdown Shell Recorder test below shows that YAML CPP handles the conversion from and to YAML's boolean type properly. In the test we also use the `type` plugin to makes sure that YAML CPP interacts correctly with this essential plugin.

```
# Mount plugin
sudo kdb mount config.yaml user:/tests/storage yamlcpp type
kdb set user:/tests/storage/bool/value true
kdb meta-set user:/tests/storage/bool/value type boolean
kdb get user:/tests/storage/bool/value
#> 1
kdb set user:/tests/storage/bool/value 1
kdb get user:/tests/storage/bool/value
#> 1
kdb set user:/tests/storage/bool/value false
kdb get user:/tests/storage/bool/value
#> 0
kdb set user:/tests/storage/bool/value 'non boolean'
# RET: 5
kdb get user:/tests/storage/bool/value
#> 0
# Undo modifications to the key database
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

## 483.4 Support Values Inside Non-Leaf Keys

Sometimes the most “natural” mapping of key-value pairs to a file format might cause a storage plugin to not be able to store values in so-called directory (non-leaf) keys.
For example, in a key set that contains the keys:

```
user:/directory
user:/directory/leaf1
user:/directory/leaf2
user:/leaf3
```

, all keys at the bottom of the hierarchy:

```
        user
      /      \
  directory  leaf3
   /     \
leaf1   leaf2
```

, such as

- user:/directory/leaf1
- user:/directory/leaf2
- user:/leaf3

are called leaf keys, while user:/directory is a directory key. Plugins such as YAJL or YAML CPP will not be able to store data in the key with the name user:/directory directly. To work around this issue these plugin use the Directory Value plugin. In the ReadMe of the Directory Value plugin and YAML CPP you will find more information about this issue, and how to handle it.
The following Markdown Shell Recorder test shows **the proper behavior**:

```
# Mount plugin
sudo kdb mount config.yaml user:/tests/storage yamlcpp
```

```
# Add key-value pair (leaf key)
kdb set user:/tests/storage/root cockerel
# Since we add a key below 'user:/tests/storage/root', the key
# 'user:/tests/storage/root' turns from a leaf key to a directory key.
kdb set user:/tests/storage/root/level1/level2/level3 hatching chick
# Make sure that the directory key still stores the correct value
kdb get user:/tests/storage/root
#> cockerel
# Check the value of the leaf key
kdb get user:/tests/storage/root/level1/level2/level3
#> hatching chick
# Undo modifications to the key database
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

. To make sure that your storage plugin works correctly, please just replace `yamlcpp` with the name of your plugin and verify that the test above still works.

## 483.5 Support Array And Non-Array Data Properly

You already learned about the array syntax and the mandatory `array` metakey in the array tutorial. Now it is time to check, if your storage plugin supports array and non-array keys properly. Let us look at a concrete example. We use a key set that contains the following keys as example:

```
user:/tests/storage/array
user:/tests/storage/array/#0
user:/tests/storage/array/#1
user:/tests/storage/map
user:/tests/storage/map/#0
user:/tests/storage/map/#1
```

. If we assume that only `user:/tests/storage/array` stores the metakey `array`, then the keys

- `user:/tests/storage/array/#0`, and
- `user:/tests/storage/array/#1`

represent array elements, while

- `user:/tests/storage/map/#0`, and
- `user:/tests/storage/map/#1`

are normal key-value pairs. The following example shows that the storage plugin `YAML CPP` handles this situation properly:

```
# Mount plugin
sudo kdb mount config.yaml user:/tests/storage yamlcpp
# Create an array containing two elements
kdb meta-set user:/tests/storage/array array "
kdb set user:/tests/storage/array/#0 one
kdb set user:/tests/storage/array/#1 two
# The array parent key stores the basename of the last element
kdb meta-get user:/tests/storage/array array
#> #1
# If you do not add the metakey 'array', then keys
# containing array syntax '#0', '#1', ... will not be
# interpreted as arrays.
kdb set user:/tests/storage/map ""
kdb set user:/tests/storage/map/#0 ""
kdb set user:/tests/storage/map/#1 ""
kdb meta-get user:/tests/storage/map array
# RET: 12
# Undo modifications to the key database
kdb rm -r user:/tests/storage
sudo kdb umount user:/tests/storage
```

## 483.6 Storing Comments

Most markup languages provide the possibility of adding comments. Elektra can store those comments in its metadata as well. This can be achieved by setting the meta Keys `comment/#` for the respective configuration Key. Also the `hosts` plugin stores the comments of the file in the respective Elektra configuration Key:

```
# Mount empty hosts file
sudo kdb mount --with-recommends hosts user:/tests/hosts hosts
# Add a line to the hosts file containing a comment
echo '127.0.0.1    localhost # test comment' >  `kdb file user:/tests/hosts`
# Check if the line has been synced successfully
```

```
kdb get user:/tests/hosts/ipv4/localhost
#> 127.0.0.1
kdb meta-ls user:/tests/hosts/ipv4/localhost
#> comment/#0
#> comment/#0/space
#> comment/#0/start
#> order
kdb meta-get user:/tests/hosts/ipv4/localhost 'comment/#0'
#>  test comment
# Undo modifications to the key database
kdb rm -r user:/tests/hosts
sudo kdb umount user:/tests/hosts
```

## 483.7 Ordering of Elements

If your plugin also has the ability to store configuration options in a certain order, then this is also supported by Elektra. Keys can have the metakey `order`, which indicates in which order lines should be written back to the configuration file. Inversely, when reading from configuration files, plugins should add the `order` metakey to the respective KDB entries.

This behavior can be illustrated via the usage of the `hosts` plugin, which honors this convention:

```
# Mount empty hosts file
sudo kdb mount --with-recommends hosts user:/tests/hosts hosts
# Add lines to the hosts file
echo '127.0.0.1    localhost.1' >  `kdb file user:/tests/hosts`
echo '127.0.0.1    localhost.2' »  `kdb file user:/tests/hosts`
# Check if the lines have been synced successfully
kdb ls user:/tests/hosts/ipv4
#> user:/tests/hosts/ipv4/localhost.1
#> user:/tests/hosts/ipv4/localhost.2
# Checking the created Meta KeySet
kdb meta-ls user:/tests/hosts/ipv4/localhost.1
#> comment/#0
#> order
# Getting the content of the order
kdb meta-get user:/tests/hosts/ipv4/localhost.1 order
#> 1
kdb meta-get user:/tests/hosts/ipv4/localhost.2 order
#> 2
# adding some additional Keys out of order
kdb set user:/tests/hosts/ipv4/localhost.4 127.0.0.1
kdb set user:/tests/hosts/ipv4/localhost.3 127.0.0.1
# lines in hosts file have improper ordering
cat `kdb file user:/tests/hosts`
#> 127.0.0.1    localhost.3
#> 127.0.0.1    localhost.4
#> 127.0.0.1    localhost.1
#> 127.0.0.1    localhost.2
# setting the correct order
kdb meta-set user:/tests/hosts/ipv4/localhost.4 order 4
kdb meta-set user:/tests/hosts/ipv4/localhost.3 order 3
# lines in hosts file are also in correct order afterwards
cat `kdb file user:/tests/hosts`
#> 127.0.0.1    localhost.1
#> 127.0.0.1    localhost.2
#> 127.0.0.1    localhost.3
#> 127.0.0.1    localhost.4
# Undo modifications to the key database
kdb rm -r user:/tests/hosts
sudo kdb umount user:/tests/hosts
```

As you can see by setting the order metakey in the respective KDB entries, we can manipulate the order in which entries get written to the hosts file. Also when importing from the initial hosts file, the plugin stores the correct order in the meta KeySet.

# Chapter 484

# Introduction

Podman is an alternative container engine which does not require root privileges on the host OS.
Podman aims at being compatible with Docker and in most cases simply replacing `docker` by `podman` is enough.
However, there are some notable differences when running containers themselves as a user other than root.

## 484.1 Who Is This Guide For?

Anyone who uses Podman to run their containers.

## 484.2 Prerequisites

- A working Podman installation

## 484.3 Basics

To build the Debian bullseye container with Docker the command is

```
docker build \
  --tag "build-elektra-debian-bullseye" \
        --build-arg "JENKINS_USERID=$(id -u)" \
        --build-arg "JENKINS_GROUPID=$(id -g)" \
        --file "scripts/docker/debian/bullseye/Dockerfile" \
        scripts/docker/debian/bullseye/
```

Simply replacing `docker` with `podman` works in this case

```
podman build \
  --tag "build-elektra-debian-bullseye" \
        --build-arg "JENKINS_USERID=$(id -u)" \
        --build-arg "JENKINS_GROUPID=$(id -g)" \
        --file "scripts/docker/debian/bullseye/Dockerfile" \
        scripts/docker/debian/bullseye/
```

## 484.4 Running a container as a user other than root

One area where Podman notably differs from Docker is when interacting with the filesystem of the host.
This is relevant when mounting directories of the host filesystem as volumes.
With Docker the container engine itself runs as a priviledged process and therefore the permissions in the host OS are not relevant.
With Podman the situation is more involved, as the container engine does not run with root priviledges.
For instance, the containers specified for building Elektra are configured such that the user inside the container is a non-root user.
If one wishes to mount the Elektra source directory from the host filesystem as a volume for the container, extra steps are necessary.
Inside the source directory, you can change the permissions to any user id with `podman unshare chown $(id -u):$(id -u) -R .`. Keep in mind that this changes the *host* filesystem. You can read more about this here.
After having changed the permissions, one can run the container

```
podman run \
```

```
   --user $(id -u) \
   --interactive \
   --tty \
   --rm \
   --volume "$PWD:/home/jenkins/workspace:Z" \
   --workdir "/home/jenkins/workspace" \
   build-elektra-debian-bullseye
```

Do not forget the `:Z` label. You can read more about the labels in the Podman documentation.
Alternatively, if you prefer to not change the permissions of the host filesystem, you can run the container as a root user. Note however, that the environment will differ from that used in continuous integration and the usual caveats concerning running processes as root apply:

```
podman run \
   --user "root" \
   --interactive \
   --tty \
   --rm \
   --volume "$PWD:/home/jenkins/workspace:Z" \
   --workdir "/home/jenkins/workspace" \
   build-elektra-debian-bullseye
```

# Chapter 485

# Validation

## 485.1 Introduction

Configuration in <abbr title="Free/Libre and Open Source Software">FLOSS</abbr> unfortunately is often stored completely without validation. Notable exceptions are sudo (`visudo`), or user accounts (`adduser`) but in most cases you only get feedback of non-validating configuration when the application fails to start.
Elektra provides a generic way to validate any configuration before it is written to disc.

## 485.2 User Interfaces

Any of Elektra's user interfaces will work with the technique described in this tutorial, e.g.:

1. `kdb qt-gui`: graphical user interface
2. `kdb editor`: starts up your favorite text editor and allows you to edit configuration in any syntax. (generalization of `visudo`)
3. `kdb set`: manipulate or add individual configuration entries. (generalization of `adduser`)
4. Any other tool using Elektra to store configuration (e.g. if the application itself has capabilities to modify its configuration)

## 485.3 Metadata Together With Keys

The most direct way to validate keys is

```
kdb meta-set user:/tests/together/test check/validation "[1-9][0-9]*"
kdb meta-set user:/tests/together/test check/validation/match LINE
kdb meta-set user:/tests/together/test check/validation/message "Not a number"
kdb set user:/tests/together/test 123
#> Set string to "123"
# Undo modifications
kdb rm -r user:/tests/together
```

The approach is not limited to validation via regular expressions, but any values-validation plugin can be used, e.g. type. For a full list refer to the section "Value Validation" in the list of all plugins.
Note that it's also easy to write your own (value validation) plugin.
The drawbacks of this approach are:

- The administrator needs to take care that the validation plugins are available where needed.
- Some metadata (in this case `check/validation`) needs to be stored next to the key which won't work with most configuration files. This is the reason why we explicitly used `dump` as storage in `kdb mount`.
- After the key is removed, the validation information is gone, too.
- You cannot validate structure of which keys must be present or absent.

## 485.4 Get Started with <tt>spec</tt>

These issues are resolved straightforward by separating the configuration from its configuration specification (often called schemata in XML or JSON). The purpose of the spec namespace is to hold the configuration specification, i.e., the description of how to validate the keys of all other namespaces.

To make this work, we need a plugin that applies all metadata found in the `spec`-namespace to all other namespaces. This plugin is called `spec` and needs to be mounted globally (will be added by default and also with any `kdb global-mount` call).

Before we start, let us make a backup of the current data in the spec and user namespace:

```
kdb set system:/tests/specbackup $(mktemp)
kdb set system:/tests/userbackup $(mktemp)
kdb export spec:/ dump > $(kdb get system:/tests/specbackup)
kdb export user:/ dump > $(kdb get system:/tests/userbackup)
```

We write metadata to the namespace `spec` and the plugin `spec` applies it to every cascading key:

```
kdb meta-set spec:/tests/spec/test hello world
kdb set user:/tests/spec/test value
kdb meta-ls spec:/tests/spec/test | grep -v '^internal/ini'
#> hello
kdb meta-ls /tests/spec/test | grep -v '^internal/ini'
#> hello
kdb meta-get /tests/spec/test hello
#> world
```

But it also supports globbing (_ for any key, `?` for any char, `[]` for character classes):

```
kdb meta-set "spec:/tests/spec/_" new metaval
kdb set user:/tests/spec/test value
kdb meta-ls /tests/spec/test | grep -v '^internal/ini'
#> hello
#> new
# Remove keys and metadata from the commands above
kdb rm -r spec:/tests/spec
kdb rm -r user:/tests/spec || kdb rm -r system:/tests/spec
```

So let us combine this functionality with validation plugins. So we would specify:

```
kdb meta-set spec:/tests/spec/test check/validation "[1-9][0-9]*"
kdb meta-set spec:/tests/spec/test check/validation/match LINE
kdb meta-set spec:/tests/spec/test check/validation/message "Not a number"
```

If we now set a new key with

```
kdb set user:/tests/spec/test "not a number"
#> Create a new key user:/tests/spec/test with string "not a number"
```

this key has adopted all metadata from the spec namespace:

```
kdb meta-ls /tests/spec/test | grep -v '^internal/ini'
#> check/validation
#> check/validation/match
#> check/validation/message
```

Note that this key should not have passed the validation that we defined in the spec namespace. Nonetheless we were able to set this key, because the validation plugin was not active for this key. On that behalf we have to make sure that the validation plugin is loaded for this key with:

```
kdb mount tutorial.dump /tests/spec dump validation
```

This mounts the backend `tutorial.dump` to the mount point **/tests/spec** and activates the validation plugin for the keys below the mount point. The validation plugin now uses the metadata of the keys below **/tests/spec** to validate values before storing them in `tutorial.dump`.

If we try setting the key again, we will get an error:

```
kdb set user:/tests/spec/test "not a number"
# STDERR: .*Validation Syntactic.*Not a number.*
# ERROR:  C03100
# RET: 5
```

However, if we add a key that adheres to the validation rules, it will work:

```
kdb set user:/tests/spec/test 42
#> Create a new key user:/tests/spec/test with string "42"
```

Although this is better than defining metadata in the same place as the data itself, we can still do better. The reason for that is that one of the aims of Elektra is to remove the trouble of validation and finding the files that hold your configuration from the users. At the moment a user still has to know which files should hold the configuration and which plugins must be loaded when they mount configuration files.

This problem can be addressed by recognizing that the location of the configuration files and the plugins that must be loaded is part of the *schema* of our configuration and therefore should be stored in the spec namespace.

```
# Undo modifications
kdb rm -r spec:/tests/spec
kdb rm -r user:/tests/spec || kdb rm -r system:/tests/spec
kdb umount /tests/spec
```

### 485.4.1 Specfiles

We call the files, that contain a complete schema for configuration below a specific path in form of metadata, *Specfiles*.
A *Specfile* contains metadata, among others, that defines how the configuration settings should be validated.
Let us create an example *Specfile* in the dump format, which supports metadata. Although the specfile is stored in the dump format, we can still create it using the human-readable ni format by using `kdb import` (note that the `\\` are due to `Markdown Shell Recorder`, do not copy them to your shell):

```
sudo kdb mount tutorial.dump spec:/tests/tutorial dump
cat « HERE | kdb import spec:/tests/tutorial ni  \
[]                                               \
 mountpoint=tutorial.dump                       \
 infos/plugins=dump validation                  \
                                                  \
[/links/_]                                       \
check/validation=https?://.*\..*                \
check/validation/match=LINE                     \
check/validation/message=not a valid URL \
description=A link to some website              \
HERE
kdb meta-ls spec:/tests/tutorial
#> infos/plugins
#> mountpoint
```

We now have all the metadata that we need to mount and validate the data below `/tutorial` in one file.
For a description which metadata is available, have a look in `METADATA.ini`.
Now we apply this *Specfile* to the key database to all keys below `tests/tutorial`.

```
kdb spec-mount /tests/tutorial
```

This command automatically mounts `/tests/tutorial` to the backend `tutorial.dump`. Furthermore it adds all plugins necessary for all metadata within the specification. So in this example the validation plugin will be loaded automatically for us. `spec-mount` basically does a normal mount except that it automatically selects plugins. As a result there is no `spec-umount` command since the normal `umount` is sufficient.
Please be aware that if you require many plugins for the same mount point, you can run into `this` error.

```
kdb set user:/tests/tutorial/links/url "invalid url"
# STDERR: .*Validation Syntactic.*not a valid URL.*
# ERROR:  C03100
# RET:    5
```

Note that the backend `tutorial.dump` is mounted for all namespaces:

```
kdb file user:/tests/tutorial
# STDOUT-REGEX: /.*/tutorial\.dump
kdb file system:/tests/tutorial
# STDOUT-REGEX: /.*/tutorial\.dump
kdb file dir:/tests/tutorial
# STDOUT-REGEX: /.*/tutorial\.dump
```

If you want to go without validation, you can work around that by setting the keys with the `-f` (`--force`) option:

```
kdb set -f system:/tests/tutorial/links/elektra "invalid url"
#> Create a new key system:/tests/tutorial/links/elektra with string "invalid url"
```

## 485.5 Rejecting Configuration Keys

Up to now we only discussed how to reject keys that have unwanted values. Sometimes, however, applications require the presence or absence of keys. There are many ways to do so directly supported by the spec plugin. Another way is to trigger errors with the error plugin:

```
kdb meta-set spec:/tests/tutorial/spec/should_not_be_here trigger/error C03200
kdb spec-mount /tests/tutorial
kdb set user:/tests/tutorial/spec/should_not_be_here abc
# RET:    5
# ERROR:C03200
kdb get /tests/tutorial/spec/should_not_be_here
# RET: 11
# STDERR: Did not find key '/tests/tutorial/spec/should_not_be_here'
```

If we want to reject every optional key (and only want to allow required keys) we can use the plugin `required` as further discussed below.
Before we look further let us undo the modifications to the key database.

```
kdb rm -r spec:/tests/tutorial
kdb rm -r system:/tests/tutorial
kdb rm -rf user:/tests/tutorial
kdb umount spec:/tests/tutorial
kdb umount /tests/tutorial
kdb rm -rf spec:/
kdb rm -rf user:/
kdb import spec:/ dump < $(kdb get system:/tests/specbackup)
kdb import user:/ dump < $(kdb get system:/tests/userbackup)
rm $(kdb get system:/tests/specbackup)
rm $(kdb get system:/tests/userbackup)
```

```
kdb rm system:/tests/specbackup
kdb rm system:/tests/userbackup
```

## 485.6 Validate Existing Keys

To check if an existing set of keys can be read and written with the current validation rules `kdb validate` should be used. Validate will read the values of all string keys under the point defined as argument in the command line, sets the key value to something different, then back to the original and finally writes that original value back to the key database. All loaded validation plugins are now used to validate the values of keys with the necessary meta-keys (see above).
Only string keys are validated! Binary keys are skipped!

```
# mount test config file and set a value
sudo kdb mount range.ecf /tests/range range dump
# set value
kdb set user:/tests/range/value 5
# add range check to all keys under /tests/range/
kdb meta-set spec:/tests/range/_ check/range "1-10"
# check if validate passes
kdb validate /tests/range
# set new key to invalid value (with kdb set -f)
kdb set -f user:/tests/range/value2 11
# validation fails now
kdb validate /tests/range
# RET:11
# clean up
kdb rm -r /tests/range/
sudo kdb umount /tests/range
```

# Chapter 486

# Using Xfconf with Elektra

Elektra provides the ability to work hand in hand with Xfconf. Depending on what you want, you can use Elektra in different ways.

## 486.1 Altering Existing Xfconf Settings

Elektra provides an Xfconf storage plugin that allows you to change all the configuration settings made by Xfconf. To access the properties which are owned by Xfconf, the appropriate channel must be mounted. Some known channel names are

- **Thunar**: thunar
- **Xfwm**: xfwm4
- **Xfce Panel**: xfce4-panel

Assuming that the properties of Thunar anf Xfwm are to be changed, the following command can be used to mount the channels. If you are operating in a headless console session (e.g. docker), make sure that dbus is running. If it is not, `export $(dbus-launch)` can be used for that.

```
touch none
kdb mount -R noresolver none /sw/xfce4/thunar xfconf channel=thunar
kdb mount -R noresolver none /sw/xfce4/xfwm4 xfconf channel=xfwm4
```

**Warning**: The following operations will cause permanent changes to your system, please handle with care.
Thunar should now be ready to be configured by Elektra. The following commands can be used to configure some selected options of Xfce.

```
# Use the details view instead of the grid view in Thunar
kdb set system:/sw/xfce4/thunar/last-view ThunarDetailsView
#> Create a new key system:/sw/xfce4/thunar/last-view with string "ThunarDetailsView"
# Do not perform a recursive search on network directories
kdb set system:/sw/xfce4/thunar/misc-recursive-search THUNAR_RECURSIVE_SEARCH_LOCAL
#> Create a new key system:/sw/xfce4/thunar/misc-recursive-search with string
       "THUNAR_RECURSIVE_SEARCH_LOCAL"
# Move the window buttons to the left
kdb set system:/sw/xfce4/xfwm4/general/button_layout "CM|O"
#> Create a new key system:/sw/xfce4/xfwm4/general/button_layout with string "CM|O"
```

The result can then be verified using both Elektra and `xfconf-query`.

```
kdb get /sw/xfce4/thunar/last-view
#> ThunarDetailsView
xfconf-query -c thunar -p /last-view
#> ThunarDetailsView
kdb get /sw/xfce4/thunar/misc-recursive-search
#> THUNAR_RECURSIVE_SEARCH_LOCAL
xfconf-query -c thunar -p /misc-recursive-search
#> THUNAR_RECURSIVE_SEARCH_LOCAL
kdb get /sw/xfce4/xfwm4/general/button_layout
#> CM|O
xfconf-query -c thunar -p /xfwm4/general/button_layout
#> CM|O
```

Using a text editor, you can also view the changes in the file `${XDG_CONFIG_HOME:-$HOME/.config}/xfce4/xfconf/xf`
However, this file may not be up-to-date since the Xfconf daemon has a caching mechanism.
When you are finished configuring the Xfce component, you should unmount the corresponding channel with the following command.

```
kdb umount /sw/xfce4/thunar/
kdb umount /sw/xfce4/xfwm4/
```

## 486.2 Replacing Xfconf with Elektra

It is also possible to replace Xfconf entirely with Elektra. The Xfconf binding in Elektra implements the Xfconf API in such a way that it can be used as a drop-in replacement.

### 486.2.1 Setup

To start, the Xfconf library must be replaced with the Xfconf binding. Replacing the system libraries may require root privileges.

```
# Use Elektra instead of Xfconf system-wide
kdb xfconf-system-lib-replace
# Use Elektra instead of Xfconf only for the current user
kdb xfconf-user-lib-replace && source ~/.xprofile
```

Note that a system upgrade may reset the system library replacement.

All applications that normally use Xfconf will now use Elektra instead. This can be verified using both `xfconf-query` and `kdb`.

```
# Set an Xfconf property with Elektra
kdb set system:/sw/org/xfce/xfconf/test/hello world
#> Create a new key system:/sw/org/xfce/xfconf/test/hello with string "world"
# Verify it with Xfconf
xfconf-query -c test -p /hello
#> world
```

### 486.2.2 Run Xfce Using Elektra

The Xfce desktop provides some default settings which are loaded on first login. However, if you are using Elektra instead of Xfconf, you will need to load these defaults manually. This can be done entirely using `kdb`.

```
kdb xfconf-populate
```

Once this is done, an Xfce session can be started either from the login manager or manually. From this point on, the Xfce desktop will read and write to the Elektra database. However, Xfce also uses some properties that are not stored in Xfconf. These properties will not work properly. An example of such a property is the Gtk theme.

### 486.2.3 Reverting the Libraries

If you want to stop using Elektra instead of Xfconf, you have to choose the right option depending on how you did the setup.

If you did the setup using the system libraries, it is sufficient to run it:

```
kdb xfconf-system-lib-restore
```

If you have chosen the user only way, you will need to edit your `~/.xprofile` file and remove the statements where the `LD_*` variables contain anything related to Xfconf.

## 486.3 Data-Types Used in Xfconf

Xfconf uses the glib2 type system. This system is not fully compatible with Elektra. However, all values are stored in Elektra in their string representation when used as a drop-in replacement. In addition, the name of the glib2 type is stored in Elektra as a metakey. This makes Xfconf's type system completely independent of Elektra, at the cost of memory and computational resources.

The following table shows a comparison between the different type systems.

| Xfconf | Elektra | Xfce Property Example (Channel, Property) | Note |
|---|---|---|---|
| `string` | `char*` | `xfce4-panel`, `/panels/panel-1/position` | |
| `uchar` | `kdb_octet_t` | | Not implemented in the Xfconf binding. |
| `char` | `kdb_char_t` | | Not implemented in the Xfconf binding. |
| `uint16` | `kdb_unsigned_short_t` | | Not implemented in the Xfconf binding. |
| `int16` | `kdb_short_t` | | Not implemented in the Xfconf binding. |

| Xfconf | Elektra | Xfce Property Example (Channel, Property) | Note |
|---|---|---|---|
| uint | kdb_unsigned_long_t | xfce4-panel, /panels/panel-1/length | |
| int | kdb_long_t | xfce4-session, /sessions/Count | |
| uint64 | kdb_unsigned_long_↩ long_t | | |
| int64 | kdb_long_long_t | | |
| float | kdb_float_t | | |
| double | kdb_double_t | | |
| bool | kdb_boolean_t | xfce4-panel, /panels/panel-1/position-locked | |
| array | | xfce4-panel, /panels | No direct type in Elektra. The key name structure determines whether it is an array or not. |
| empty | | xfce4-panel, /panels/panel-1 | Does not exist in Elektra. Paths in Elektra which are no leaf nodes will be equally handled. |

# Chapter 487

# elektra-bindings Use Cases

The use cases of this directory are intended to describe scenarios where using bindings of libelektra is required. An understanding how libelektra is structured is highly recommended for all use cases.

- Developing a new application
- Integrate into an existing application
- Usage as a drop-in replacement

# Chapter 488

# Use Case: Use libelektra as drop-in Replacement

## 488.1 Summary

- **Scope:** API
- **Level:** Kite
- **Actors:** Administrator
- **Brief:** Administrator wants to use libelektra in an application which uses another configuration management than libelektra
- **Status:** Draft

## 488.2 Scenarios

- **Precondition:** A binding for the configuration management which the application uses must exist.
- **Main success scenario:** The application can be fully configured through libelektra by using libelektra as a drop-in replacement without changing any source code.
- **Alternative scenario:** Due to limitations of certain bindings, not everything but the majority can be configured.
- **Error scenario:** The application might be put into a faulty state when the keys get modified by another application or manually.
- **Postcondition:** The application is able to read and write through libelektra.

# Chapter 489

# Use Case: Programming Language Bindings in existing Application

## 489.1 Summary

- **Scope:** API
- **Level:** Underwater
- **Actors:** API User
- **Brief:** Developer wants to use libelektra in an existing application written in another language than C
- **Status:** Draft

## 489.2 Scenarios

- **Precondition:** A binding for the programming language must exist and the maintainers of the application decided to use libelektra and the developer must have write access to the application.
- **Main success scenario:** The application can be fully configured through libelektra
- **Alternative scenario:** Due to limitations of certain bindings, not everything but the majority can be configured
- **Error scenario:** The application might be put into a faulty state when the keys get modified by another application or manually
- **Postcondition:** The application is able to read and write with libelektra

# Chapter 490

# Use Case: Programming Language Bindings

## 490.1 Summary

- **Scope:** API
- **Level:** Sea
- **Actors:** API User
- **Brief:** Developer wants to use libelektra in a new application written in another language than C
- **Status:** Draft

## 490.2 Scenarios

- **Precondition:** A binding for the programming language must exist and the maintainers of the new application decided to use libelektra.
- **Main success scenario:** The application can be fully configured through libelektra
- **Alternative scenario:** Due to limitations of certain bindings, not everything but the majority can be configured
- **Error scenario:** The application might be put into a faulty state when the keys get modified by another application or manually
- **Postcondition:** The new application is able to read and write with libelektra

# Chapter 491

# Use cases for <tt>libelektra-core</tt>

This folder contains the use cases for `libelektra-core`.
`libelektra-core` primarily implements an ordered, hierarchical associative array data structure, called `KeySet`, which:

1. uses arbitrary byte sequences, grouped into namespaces, as keys
2. associates each key with
   (a) values: an arbitrary byte sequence
   (b) metadata: another ordered, hierarchical associative array, which only associates keys with values
3. orders keys first by namespace then lexicographically with respect to hierarchy
4. supports the operations: insert, remove, lookup, hierarchy lookup (a form of prefix lookup) and access by index (which enables iteration)

Additionally, `libelektra-core` provides a data structure, called `Key`, that represents a single key-value pair in the associative array, but can also be used standalone.
To support the hierarchical and ordered nature of a `KeySet` there are two fundamental comparison operations that can be performed on two `Key`s:

1. Order Comparison: Establishes the linear order of `Key`s and thereby defines the iteration order of a `KeySet`.
2. Hierarchy Comparison: Establishes the hierarchy of `Key`s, by defining based on their names whether one `Key` is a descendant of another.

The individual use cases provide details on these data structures and the operations it supports.

# Chapter 492

# Use Case: <tt>Key</tt> Basename

## 492.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to and manipulation of `Key`'s basename
- **Status:** Implemented

## 492.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests current basename of `Key`
  - Core returns current basename of `Key`
- **Alternative scenario:**
  - Caller requests to change basename of `Key` to new basename
  - Core stores new basename in `Key`
- **Alternative scenario:**
  - Caller requests to append new name part to `Key`
  - Core stores appended name in `Key`
- **Error scenario:**
  - `Key` has name marked as read-only (e.g., because `Key` is part of ordered collection like `KeySet`)
  - Caller requests to change basename of `Key` to new basename
  - Core returns error and basename of `Key` remains unchanged
- **Error scenario:**
  - `Key` has name marked as read-only (e.g., because `Key` is part of ordered collection like `KeySet`)
  - Caller requests to append to name of `Key`
  - Core returns error and basename of `Key` remains unchanged
- **Postcondition:** -
  - `Key` has a valid name
- **Non-functional Constraints:** -

# Chapter 493

# Use Case: <tt>Key</tt> clear

## 493.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Clearing contents of a `Key`
- **Status:** Implemented

## 493.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests the fields of a `Key` should get cleard
  - Name, value and metadata of the `Key` are empty after clearing
- **Alternative scenario:**
- **Error scenario:**
  - `Key` has name, value or metadata marked as read-only (e.g., because `Key` is part of ordered collection like `KeySet`)
  - Caller requests to clear `Key`
  - Core returns error and name, value and metadata of `Key` remain unchanged
- **Postcondition:**
- **Non-functional Constraints:**

# Chapter 494

# Use Case: <tt>Key</tt> copy

## 494.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Copying one or more fields from one `Key` to another
- **Status:** Implemented

## 494.2 Scenarios

- **Precondition:**
  - `Key` K1 has been created
  - `Key` K2 has been created
- **Main success scenario:**
  - Caller requests to copy one or multiple fields from `Key` k1 to `Key` k2
  - Fields that can be copied are name, value and metadata
  - Fields, that caller wanted to copy, of `Key` k2 now have the value of the respective fields of `Key` k1
- **Alternative scenario:**
- **Error scenario:**
  - `Key` k2 has field that should be copied marked as read-only (e.g., because `Key` is part of ordered collection like `KeySet`)
  - Caller requests to copy read-only field in `Key` k2 from `Key` k1
  - Copying to k2 fails and k2 retains the old field values
- **Postcondition:**
  - Comparing the respective fields from k1 and k2 returns that those fields are equal
- **Non-functional Constraints:**

# Chapter 495

# Use Case: Create <tt>Key</tt>

## 495.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller creates new `Key` instance

## 495.2 Scenarios

- **Precondition:**
  - The name used to create a `Key` must be valid
- **Main success scenario:**
  - Caller requests to create a new `Key` with a given name
  - Core instantiates a `Key` with the given name, no value and no metadata and returns it
- **Alternative scenario:**
  - Caller requests to create a new `Key`, with a given name and a value and/or metadata
  - Core instantiates a `Key`, with the given name, value, and metadata and returns it
- **Error scenario:** -
- **Postcondition:**
  - `Key` exists and is usable by Caller
  - `Key` has name defined by Caller
  - `Key` has value defined by Caller, if given, and no value otherwise
  - `Key` has metadata defined by Caller, if given and no metadata otherwise
- **Non-functional Constraints:** -

# Chapter 496

# Use Case: <tt>Key</tt> lock

## 496.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to and manipulation of `Key`'s lock flags
- **Status:** Implemented

## 496.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests lock status of one or multiple flags of a `Key`
  - Possible lock flags are: name, value, metadata
  - Core returns for which lock flags the lock bit is set in the `Key`
- **Alternative scenario:**
  - Caller requests to set one or multiple lock flags for `Key`
  - Possible lock flags are: Name, Value, Meta
  - Core sets bits for lock flags in the `Key`
- **Error scenario:**
- **Postcondition:**
  - Locked fields cannot be changed via other functions from the Elektra Core API
- **Non-functional Constraints:**

# Chapter 497

# Use Case: <tt>Key</tt> Metadata

## 497.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to `Key`'s metadata

## 497.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests metadata of `Key`
  - Core returns metadata of `Key` as modifiable `KeySet`
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 498

# Use Case: <tt>Key</tt> Name

## 498.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to and manipulation of `Key`'s name

## 498.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests current name of `Key`
  - Core returns current name of `Key`
- **Alternative scenario:**
  - Caller requests to change name of `Key` to new name
  - Core stores new name in `Key`
- **Error scenario:**
  - `Key` has name marked as read-only (e.g., because `Key` is part of ordered collection like `KeySet`)
  - Caller requests to change name of `Key` to new name
  - Core returns error and name of `Key` remains unchanged
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 499

# Use Case: <tt>Key</tt> Namespace

## 499.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to and manipulation of `Key`'s namespace

## 499.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests current namespace of `Key`
  - Core returns current namespace of `Key`
- **Alternative scenario:**
  - Caller requests to change namespace of `Key` to new namespace
  - Core stores new namespace in `Key`
- **Error scenario:**
  - `Key` has name marked as read-only (e.g., because `Key` is part of ordered collection like `KeySet`)
  - Caller requests to change namespace of `Key` to new namespace
  - Core returns error and namespace of `Key` remains unchanged
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 500

# Use Case: <tt>Key</tt> reference counting

## 500.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to and manipulation of `Key`'s reference counter
- **Status:** Implemented

## 500.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests value of reference counter of a `Key`
  - Core returns current value of reference counter of a `Key`
- **Alternative scenario:**
  - Caller requests to increment value of reference counter of a `Key`
  - Core returns current value of reference counter of a `Key` after incrementing
- **Alternative scenario:**
  - Caller requests to decrement value of reference counter of a `Key`
  - Core returns current value of reference counter of a `Key` after decrementing
- **Error scenario:**
  - Caller requests to increment value of reference counter of a `Key` whose reference counter is INT_MAX
  - Reference counter will not get incremented and stays at INT_MAX
- **Error scenario:**
  - Caller requests to decrement value of reference counter of a `Key` whose reference counter is 0
  - Reference counter will not get decremented and stays at 0
- **Postcondition:**
  - Reference Counter has a value from 0 to INT_MAX
- **Non-functional Constraints:**

# Chapter 501

# Use Case: <tt>Key</tt> Value

## 501.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Access to and manipulation of `Key`'s value

## 501.2 Scenarios

- **Precondition:**
  - `Key` has been created
- **Main success scenario:**
  - Caller requests current value of `Key`
  - Core returns current value of `Key`
- **Alternative scenario:**
  - Caller requests to change value of `Key` to new value
  - Core stores new value in `Key`
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 502

# Use Case: <tt>Key</tt> Hierarchy

## 502.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Hierarchy comparison between `Key`'s

## 502.2 Scenarios

- **Precondition:**
  - `Key` K1 has been created
  - `Key` K2 has been created
- **Main success scenario:**
  - Caller reads names N1 and N2 of `Key` K1 and `Key` K2
  - Caller requests hierarchy comparison between N1 and N2
  - Core returns one of these results
    - N1 and N2 do not form a hierarchy
    - N1 is the same as N2
    - N2 is a direct child of N1 in the hierarchy
    - N2 is a descendant of N1 in the hierarchy
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:**
  - The hierarchy comparison MUST form a *partial order* over `Key`s.
  - Each of the possible results MAY be a single value, but it MAY also be a whole class of value (e.g. negative integer).

# Chapter 503

# Use Case: <tt>Key</tt> Ordering

## 503.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Order comparison between `Key`s

## 503.2 Scenarios

- **Precondition:**
  - `Key` K1 has been created
  - `Key` K2 has been created
- **Main success scenario:**
  - Caller reads names N1 and N2 of `Key` K1 and `Key` K2
  - Caller requests order comparison between N1 and N2
  - Core returns
    - N1 is sorted before N2
    - N1 and N2 are sorted to the same place
    - N1 is sorted after N2
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:**
  - The order comparison with MUST form a *total order* over `Key`s.
  - Each of the possible results MAY be a single value, but it MAY also be a whole class of value (e.g. negative integer).

# Chapter 504

# Use Case: Create <tt>KeySet</tt>

## 504.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller creates new `KeySet` instance

## 504.2 Scenarios

- **Precondition:** -
- **Main success scenario:**
  - Caller requests to create a new `KeySet`
  - Core instantiates an empty `KeySet` and returns it
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:**
  - `KeySet` exists and is usable by Caller
- **Non-functional Constraints:**
  - `KeySet` MUST be resizable after creation
  - `KeySet` SHOULD be efficient for small (~10) to larger (~100.000) sizes

# Chapter 505

# Use Case: Index access to <tt>KeySet</tt>

## 505.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller accesses contents of `KeySet` by index

## 505.2 Scenarios

- **Precondition:**
  - `KeySet` has been created.
  - `Key` has been inserted into `KeySet`.
- **Main success scenario:**
  - Caller requests `Key` at valid index (`0 <= i < size`) from `KeySet`
  - Core returns `Key *` for `Key` at index
- **Alternative scenario:** -
- **Error scenario:**
  - Caller requests `Key` at invalid index (`i < 0 || i >= size`) from `KeySet`
  - Core returns `NULL`
- **Postcondition:**
  - The returned `Key *` MUST be valid until the `Key` is removed from the `KeySet`.
- **Non-functional Constraints:** -

# Chapter 506

# Use Case: Insert <tt>Key</tt> into <tt>KeySet</tt>

## 506.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller inserts new `Key` into existing `KeySet`

## 506.2 Scenarios

- **Precondition:**
  - `KeySet` has been created
- **Main success scenario:**
  - Caller requests to insert a new `Key` with non-cascading name into `KeySet`
  - If necessary Core resizes `KeySet`
  - If necessary Core removes existing `Key` with same name
  - Core adds `Key` to `KeySet`
- **Alternative scenario:**
  - Caller requests to insert all `Key`s contained in one `KeySet` A into another `KeySet` B
  - If necessary Core resizes B
  - If necessary Core removes all existing `Key`s contained in A and B from B
  - Core copies all `Key`s from A to B
- **Error scenario:** -
- **Postcondition:**
  - The new `Key`(s) provided by Caller MUST be part of `KeySet` exactly once. It MUST be possible to find exactly that/those `Key`(s) via a lookup.
- **Non-functional Constraints:** -

# Chapter 507

# Use Case: Cascading Lookup in <tt>KeySet</tt>

## 507.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller looks up `Key` with cascading name in existing `KeySet`

## 507.2 Scenarios

- **Precondition:**
  - `KeySet` has been created
  - (Only Alternative) Specification for desired `Key` exists and contains links
- **Main success scenario:**
  - Caller asks Core to look up `Key` by cascading name in `KeySet`
  - Core checks for specification matching desired `Key`
  - Core does not find specification
  - Core tries equivalent name in all appropriate namespaces in correct order
  - If a matching `Key` is found, Core returns a `Key *` to it. The name of the `Key` will be read-only, otherwise it is modifiable.
  - Otherwise, Core returns `NULL`
- **Alternative scenario:**
  - Caller asks Core to look up `Key` by cascading name in `KeySet`
  - Core checks for specification matching desired `Key`
  - Core finds specification
  - Core tries links from specification and equivalent names in appropriate namespaces in correct order
  - If a matching `Key` is found, Core returns a `Key *` to it. The name of the `Key` will be read-only, otherwise it is modifiable.
  - Otherwise, Core returns `NULL`
- **Error scenario:** -
- **Postcondition:**
  - The returned index value MUST be valid and correct until new `Key`(s) are inserted into or removed from the `KeySet`.
- **Non-functional Constraints:** -

# Chapter 508

# Use Case: Direct lookup in <tt>KeySet</tt>

## 508.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller looks up `Key` with non-cascading name in existing `KeySet`

## 508.2 Scenarios

- **Precondition:**
  - `KeySet` has been created
- **Main success scenario:**
  - Caller asks Core to look up `Key` by non-cascading name in `KeySet`
  - Core searches for `Key` with same name in `KeySet`
  - If a matching `Key` is found, Core returns a `Key *` to it. The name of the `Key` will be read-only, otherwise it is modifiable.
  - Otherwise, Core returns `NULL`
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:**
  - The returned index value MUST be valid and correct until new `Key`(s) are inserted into or removed from the `KeySet`.
- **Non-functional Constraints:** -

# Chapter 509

# Use Case: Cut <tt>Key</tt> hierarchy from <tt>KeySet</tt>

## 509.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller cuts (=looks up and removes) a `Key` hierarchy from an existing `KeySet`

## 509.2 Scenarios

- **Precondition:**
  - `KeySet` has been created
- **Main success scenario:**
  - Caller asks Core to cut a `Key` hierarchy descendant from a given `Key` R from a `KeySet` KS.
  - Core searches for the first `Key` in KS that is part of the desired hierarchy, i.e., the first `Key` in KS that is a descendant of R according to `Key` hierarchy comparison.
  - If not found, Core returns an empty KS
  - If found, Core searches for the last `Key` that is not part of the desired hierarchy, i.e., Core tries all `Key`s in KS in keyname order until it finds a `Key` that is not a descendant of R.
  - Core removes all `Key`s of the desired hierarchy from KS and puts them into a new `KeySet` KS1
  - Core returns KS1.
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 510

# Use Case: Remove <tt>Key</tt> from <tt>KeySet</tt>

## 510.1 Summary

- **Scope:** `libelektra-core`
- **Level:** Developer Goal
- **Actors:** Core, Caller
- **Brief:** Caller removes `Key` from existing `KeySet`

## 510.2 Scenarios

- **Precondition:**
  - `KeySet` has been created
- **Main success scenario:**
  - Caller requests to remove `Key` by name from `KeySet`
  - Core searches for `Key` with same name in `KeySet`
  - If a matching `Key` is found, Core removes it from `KeySet` returns a `Key *` to it. The name of the `Key` will be read-only, otherwise it is modifiable.
  - Otherwise, Core returns `NULL`
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:**
  - The removed `Key`(s) MUST NOT be part of `KeySet` anymore. A lookup for that/those `Key`s MUST NOT return any results.
- **Non-functional Constraints:** -

# Chapter 511

# elektra-web Use Cases

The path through the use cases is as follows:

- Setup instance
- View config

Now we are in the main part of the application, and we can do the following use cases:

- Adding keys
- Modifying keys
- Drag & Drop keys
- Finding keys

Any of these use cases (except for finding because no change is made), are extended by the Key validation use case. After doing any of these use cases, we can do the Undo/Redo use case to revert them.

# Chapter 512

# Use Case: Adding keys

## 512.1 Summary

- **Title:** Adding keys
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User adds a key (to a subtree).

## 512.2 Scenarios

- **Precondition:** View configuration of an instance.
- **Main success scenario:** User creates a new key (in a subtree).
- **Alternative scenario:** User enters data that violates a validation rule, an error message is shown.
- **Error scenario:** Technical problems while persisting to the key database. The user is informed about the problem.
- **Postcondition:** The new key is persisted to the database.
- **Non-functional Constraints:** -

# Chapter 513

# Use Case: Drag & Drop keys

## 513.1 Summary

- **Title:** Drag & Drop keys
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User drags a key (with a subtree) to a different subtree.

## 513.2 Scenarios

- **Precondition:** View configuration of an instance.
- **Main success scenario:** User moves a key (and its subtree) to another subtree by dragging it to the desired position.
- **Alternative scenario:** User enters data that violates a validation rule, an error message is shown.
- **Error scenario:** Technical problems while persisting to the key database. The user is informed about the problem.
- **Postcondition:** The updated key structure is persisted to the database.
- **Non-functional Constraints:** -

# Chapter 514

# Use Case: Finding keys

## 514.1 Summary

- **Title:** Finding keys
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User searches for a key by filtering the key database.

## 514.2 Scenarios

- **Precondition:** View configuration of an instance.
- **Main success scenario:**
  - User searches for a key by entering information (its path or value) in a field.
  - The whole key database is filtered according to the contents of the search field.
  - At first, only keys that match the path are shown. Then, the application will continue searching for values that contain the search input in the background, and show matches once they are found.
  - When the search input field is cleared, the whole key database will be shown again.
- **Alternative scenario:** No results found, the user is informed about this.
- **Error scenario:** -
- **Postcondition:** The filtered keys can be modified.
- **Non-functional Constraints:** -

# Chapter 515

# Use Case: Key validation

## 515.1 Summary

- **Title:** Key validation
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User gets feedback about a key's contents.

## 515.2 Scenarios

- **Extends:** Modifying keys, Adding keys, Drag & Drop keys.
- **Main success scenario:** If the entered data passes the validation, it is sent to the backend and saved to the key database.
- **Alternative scenario:** User enters data that fails the field validation, the input field is marked as invalid (e.g. with a red color and some information text) and the data is NOT sent to the backend.
- **Alternative scenario:** User enters data that violates a validation rule, an error message is shown and the data is NOT saved to the key database.
- **Error scenario:** Technical problems while persisting to the key database. The user is informed about the problem.
- **Postcondition:** The updated key is persisted to the database.
- **Non-functional Constraints:** -

# Chapter 516

# Use Case: Modifying keys

## 516.1 Summary

- **Title:** Modifying keys
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User modifies a key's contents.

## 516.2 Scenarios

- **Precondition:** View configuration of an instance.
- **Main success scenario:** User edits a key's contents in a field that is specifically formatted according to its metadata (e.g. number/date fields vs string fields with regex validation). Arrays are specifically formatted as lists of fields under a key. A description of the key is also shown.
- **Alternative scenario:** User enters data that fails the field validation, the input field is marked as invalid (e.g. with a red color and some information text) and the data is *not* sent to the backend.
- **Error scenario:** Technical problems while persisting to the key database. The user is informed about the problem.
- **Postcondition:** The updated key is persisted to the database.
- **Non-functional Constraints:** -

# Chapter 517

# Use Case: Setup Instance

## 517.1 Summary

- **Title:** Setup instance
- **Scope:** Setup
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User sets up a daemon on the instance and connects it to elektra-web.

## 517.2 Scenarios

- **Main success scenario:** User successfully sets up elektrad on the instance and connects it to elektra-web.
- **Alternative scenario:** Authentication issues when connecting the services, e.g. wrong API key.
- **Alternative scenario:** The instance is already connected to elektra-web. A message informs the user about this.
- **Error scenario:** Technical problems - the user is informed about the problem.
- **Postcondition:** User can now access the instance via the client.
- **Non-functional Constraints:**
  - Security mechanism

# Chapter 518

# Use Case: Undo/Redo

## 518.1 Summary

- **Title:** Undo/Redo
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User undos/redos changes to a configuration.

## 518.2 Scenarios

- **Precondition:** Modifying keys, Adding keys, Drag & Drop keys.
- **Main success scenario:** User successfully undos/redos a configuration change.
- **Error scenario:** Technical problems while configuring the instance. The user is informed about the problem.
- **Postcondition:** The updated configuration is persisted to the instance.
- **Non-functional Constraints:**
  - Every editor operation (preconditions) should be undoable

# Chapter 519

# Use Case: View Configuration of an Instance

## 519.1 Summary

- **Title:** View configuration of an instance
- **Scope:** Configuration
- **Level:** User Goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User views the configuration of an instance via the elektra-web client.

## 519.2 Scenarios

- **Precondition:** Setup instance.
- **Main success scenario:** User opens the configuration of an instance. The user is presented with an interactive tree view.
- **Alternative scenario:** Instance not online. The user is informed about the problem.
- **Error scenario:** Technical problems while configuring the instance. The user is informed about the problem.
- **Postcondition:** The interactive tree view for configuration is visible.
- **Non-functional Constraints:** -

# Chapter 520

# Use Case: Libelektra Configuration Management for Developer

## 520.1 Summary

- **Scope:** Configuration Management
- **Level:** Cloud
- **Actors:** Developer, System Administrator
- **Brief:** Developer uses Libelektra to manage and manipulate configurations in a unified way, ensuring consistency and ease of maintenance.
- **Status:** Draft

## 520.2 Scenarios

- **Precondition:** Developer has installed Libelektra and integrated it into their project.
- **Main success scenario:** Developer successfully uses Libelektra to manage configurations, leading to improved maintainability and consistency.
- **Alternative scenario:** Developer encounters issues while integrating Libelektra, requiring them to seek assistance or refer to the documentation.
- **Error scenario:** Libelektra integration causes conflicts or unintended behavior in the application, requiring the developer to debug and resolve the issue.
- **Postcondition:** Application configurations are managed by Libelektra, leading to a more maintainable and consistent system.
- **Non-functional Constraints:**
  - Performance: Libelektra should not introduce significant performance overhead.
  - Compatibility: Libelektra should be compatible with various configuration file formats and platforms.
  - Usability: Libelektra should be easy to integrate into projects and provide a clear, intuitive API for developers.

# Chapter 521

# Use Case: Libelektra Configuration for End Users

## 521.1 Summary

- **Scope:** End User Configuration
- **Level:** Kite
- **Actors:** End User, System Administrator
- **Brief:** End user customizes application settings using the configuration interface provided by Libelektra, enabling them to tailor the application to their preferences.
- **Status:** Draft

## 521.2 Scenarios

- **Precondition:** Application has been developed with Libelektra integrated for configuration management and provides a user-friendly interface for end users to modify settings.
- **Main success scenario:** End user successfully modifies application settings using the provided interface, leading to an improved user experience tailored to their preferences.
- **Alternative scenario:** End user encounters issues while trying to modify settings, requiring them to seek assistance from support or refer to help documentation.
- **Error scenario:** End user's modifications lead to unintended behavior or conflicts in the application, requiring either the end user or system administrator to resolve the issue.
- **Postcondition:** Application settings have been modified by the end user, resulting in a customized user experience.
- **Non-functional Constraints:**
  - Usability: The configuration interface should be user-friendly and easy to navigate for end users.
  - Flexibility: Libelektra should support a wide range of configuration options and settings to cater to various end user preferences.
  - Robustness: The configuration system should be robust and capable of handling potential conflicts or errors caused by end user modifications.

# Chapter 522

# Use cases for KDB

This folder contains the use cases for the Key Database (KDB). These primarily are the basis for `libelektra-kdb`, but may also influence other things like, e.g., the high-level API `libelektra-highlevel`. The KDB is primarily a system for storing application configuration data. The individual use cases provide details for this high-level use case.

# Chapter 523

# Use Case: Get configuration

## 523.1 Summary

- **Scope:** `libelektra-kdb`
- **Level:** Developer Goal
- **Actors:** Application, Elektra
- **Brief:** Application loads configuration from KDB

## 523.2 Scenarios

- **Precondition:** -
- **Main success scenario:**
  - Application requests current configuration from KDB
  - Elektra loads from backends
  - Elektra returns configuration
- **Alternative scenario:**
  - Application requests current configuration from KDB
  - Elektra detects configuration hasn't changed and is already present in memory (of current instance)
  - Elektra returns previously loaded configuration
- **Error scenario:**
  - Application requests current configuration from KDB
  - Elektra fails to load requested configuration
  - Elektra returns error with problem description
- **Postcondition:**
  - The returned configuration MUST exactly match the requested part of the KDB
- **Non-functional Constraints:**
  - Applications SHOULD be able to err on the side of "better load config more often than needed" rather than "only load when necessary".
    - Loading configuration MUST be reasonably fast, especially when nothing has changed (caching!).
    - Reloading configuration MUST NOT destroy previously loaded configuration.

# Chapter 524

# Use Case: Modifying application configuration

## 524.1 Summary

- **Scope:** `libelektra-kdb`
- **Level:** Developer Goal
- **Actors:** Application, Elektra
- **Brief:** Application modifies configuration

## 524.2 Scenarios

- **Precondition:** -
- **Main success scenario:**
  - Application loads configuration
  - Application modifies configuration in memory
  - Application stores configuration
- **Alternative scenario:** -
- **Error scenario:** -

  **Note**: The error scenarios from the references use cases still apply. However, the in-memory modification part itself has no error scenario.

  - **Postcondition:**
  - The modified configuration MUST be stored into the KDB exactly. Loading the configuration later (assuming no outside modifications) MUST return the modified configuration.
  - In case of error, the KDB MUST NOT be modified. Loading the configuration later (assuming no outside modifications) MUST return the unmodified configuration.
- **Non-functional Constraints:**
  - Modifying configuration MUST only touch the backends that have changed data.

# Chapter 525

# Use Case: Set configuration

## 525.1 Summary

- **Scope:** `libelektra-kdb`
- **Level:** Developer Goal
- **Actors:** Application, Elektra
- **Brief:** Application stores configuration into KDB

## 525.2 Scenarios

- **Precondition:** -
- **Main success scenario:**
  - Application asks to store new configuration into KDB
  - Elektra stores configuration in backends
- **Alternative scenario:** -
- **Error scenario:**
  - Application asks to store new configuration into KDB
  - Elektra fails to store configuration
  - Elektra returns error with problem description
- **Postcondition:**
  - The provided configuration MUST be stored into the KDB exactly. Loading the configuration later (assuming no outside modifications) MUST return the provided configuration.
  - In case of error, the KDB MUST NOT be modified. Loading the configuration later (assuming no outside modifications) MUST return the same configuration as before the attempt to store new configuration.
- **Non-functional Constraints:**
  - Storing configuration MUST only touch the backends that have new data.

# Chapter 526

# Use Case: Keeping Configuration Up-to-date

## 526.1 Summary

- **Scope:** `libelektra-kdb`
- **Level:** Developer Goal
- **Actors:** Application, Elektra
- **Brief:** Application wants up-to-date configuration data

## 526.2 Scenarios

- **Precondition:** -
- **Main success scenario:** Notifications
  - Application requests asynchronous callbacks via the notification system
  - Application loads configuration
  - Every time the configuration changes, Elektra sends a notification to Application
  - When receiving a notification, Application may need to load the configuration again
- **Alternative scenario:** Polling
  - Application loads configuration repeatedly in regular intervals
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:**
  - Notifications SHOULD use standard communication systems
  - there SHOULD NOT be a rate limit of notifications

# Chapter 527

# Use Case: Validating Configuration with Specification

## 527.1 Summary

- **Scope:** `libelektra-kdb`, `spec`
- **Level:** Developer Goal, User Goal
- **Actors:** Elektra, Application, User
- **Brief:** After loading configuration, Application can assume the configuration is valid

## 527.2 Scenarios

- **Precondition:**
  - A specification, defining how a valid configuration for Application looks like, exists.
- **Main success scenario:**
  - Application loads configuration During this process Elektra checks the configuration against the specification.
  - Loading configuration results in no errors or warnings.
  - Application knows configuration is valid.
- **Alternative scenario:**
  - User requests to change configuration value via Application (e.g. `kdb`) stores configuration. During this process Elektra checks the changed configuration against the specification.
  - Elektra blocks the modification, if it violates the specification.
- **Error scenario:**
  - Application loads configuration During this process Elektra checks the configuration against the specification.
  - Loading configuration results in an error or warning.
  - If a configuration is returned at all, Application knows the configuration may not be fully valid and can react accordingly.
- **Postcondition:**
  - Configuration stored in the KDB and modified only via Elektra MUST be valid according to the specification.
  - Configuration loaded by an application *without* error or warning MUST be valid according to the specification.

  - Configuration loaded by an application *with* error or warning MUST be valid according to the specification, except where an error or warning indicates otherwise.

- **Non-functional Constraints:**
  - Because validation happens during Application loads configuration, it MUST also be reasonably performant. Results MAY be cached to achieve this performance.
  - Fixing an invalid configuration via Elektra MUST be possible. This SHOULD NOT require more changes than necessary, i.e., fixing invalid configuration MUST NOT be equivalent to "delete everything and start again".

# Chapter 528

# record-elektra Use Cases

The contained use cases are both from a developer side and user side.
The main use case is interactive configuration.

## 528.1 User-oriented Use Cases

- Concerning libelektra:
  - Allow the recording of changes in the KDB
  - Export certain keys as Ansible playbook
- Concerning ansible-libelektra:
  - Allow for keys to be deleted using ansible-libelektra
  - Allow specifying how conflicts are to be handled in ansible-libelektra
  - Assert certain values for keys for the execution to continue
  - Start session recording after Ansible run
  - Allow different configurations for different hosts

## 528.2 Developer-oriented Use Cases

- Allow plugins to be notified when data changes
- Provide an API to know which keys caused a conflict on merge

# Chapter 529

# Use Case: Assert that certain keys contain specific values

## 529.1 Summary

- **Title:** Assert that certain keys contain specific values
- **Scope:** ansible-libelektra
- **Level:** User goal
- **Actors:** System administrator
- **Brief:** There should be a functionality in ansible-libelektra to assert certain keys have certain values. For example, some app B requires that a certain app A is already set up and running before configuring can be done. It may also be used to check that a certain plugin (e.g. `check`) is correctly installed within Elektra.

## 529.2 Scenarios

- **Precondition:** The user asserts a certain value for a key in the task
- **Main success scenario:** The key on the target machine meets the assertion -> continue the execution
- **Alternative scenario:** The key on the target machine does not meet the assertion -> fail the execution
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 530

# Use Case: Select merge strategy in ansible-libelektra

## 530.1 Summary

- **Title:** Select merge-strategy in ansible-libelektra
- **Scope:** ansible-libelektra
- **Level:** User goal
- **Actors:** User (usually sysadmin)
- **Brief:** It should be possible to specify the merge strategy used for conflicts. The merging should be performed by ansible-libelektra. The following strategies should be implemented:
  - OURS: use the new values
  - THEIRS: keep the old values
  - ABORT: on conflict don't merge and report an error

## 530.2 Scenarios

- **Precondition:** -
- **Main success scenario:** If the user has specified a strategy, this strategy is applied on merge conflicts.
- **Alternative scenario:** If no strategy is specified ABORT is used.
- **Error scenario:** If there are still conflicts that could not be resolved, the offending keys should be reported.
- **Postcondition:** -
- **Non-functional Constraints:** -

# Chapter 531

# Use Case: Remove keys in ansible-libelektra

## 531.1 Summary

- **Title:** Remove keys in ansible-libelektra
- **Scope:** ansible-libelektra
- **Level:** User Goal
- **Actors:** User (usually sysadmin)
- **Brief:** It should be possible to remove keys from the KDB using ansible-libelektra

## 531.2 Scenarios

- **Precondition:** User specifies which key to delete in the playbook
- **Main success scenario:** The key exists on the target machine and will be deleted
- **Alternative scenario:** The key does not exists, nothing will be deleted
- **Error scenario:** -
- **Postcondition:** The specified key does not exist on the target machine
- **Non-functional Constraints:** -

# Chapter 532

# Use Case: Start session recording after Ansible run

## 532.1 Summary

- **Title:** Start session recording after Ansible run
- **Scope:** Recording
- **Level:** User Goal
- **Actors:** User (usually sysadmin)
- **Brief:** After a new elektra configuration has been rolled out via Ansible, automatically enable or reset session recording for the appropriate namespaces

## 532.2 Scenarios

- **Precondition:** -
- **Main success scenario:** User deploys configuration via Ansible
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:** All locally changed keys get tracked.
- **Non-functional Constraints:** -

# Chapter 533

# Use Case: Exporting configuration as Ansible playbook

## 533.1 Summary

- **Title:** Exporting configuration as Ansible playbook
- **Scope:** Configuration
- **Level:** User goal
- **Actors:** User (usually a sysadmin)
- **Brief:** User exports certain configuration as Ansible playbook

## 533.2 Scenarios

- **Precondition:** -
- **Main success scenario:**
  - User specifies which configuration to include in playbook
  - It must be possible to select only configuration that has changed (was recorded in session recording)
  - Ansible playbook is generated
- **Alternative scenario:** When no configuration exists, an empty playbook is generated
- **Error scenario:** -
- **Postcondition:**
  - Ansible playbook exists
- ## **Non-functional Constraints:**

# Chapter 534

# Use Case: Notify plugins when data in the KDB changes

## 534.1 Summary

- **Title:** Notify plugins when data in the KDB changes
- **Scope:** Configuration/Logging
- **Level:** Developer goal
- **Actors:** Plugin developer
- **Brief:** Some plugins are only executed when something changed in the KDB and as an input receive the keys that changed in their old and new state.

## 534.2 Scenarios

- **Precondition:**
  - Something in the KDB changed
  - A plugin listening to such a hook is enabled
- **Main success scenario:** After a change in the KDB the plugin(s) listening on the hooks are notified
- **Alternative scenario:** -
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:**
  - Determining which plugin(s) listen on this hook should not noticably slow down Elektra

# Chapter 535

# Use Case: Get information which keys are conflicting in 3-way-merge

## 535.1 Summary

- **Title:** Get information which keys are conflicting in 3-way-merge
- **Scope:** Library/Development
- **Level:** Subfunction
- **Actors:** Developers
- **Brief:** It should be possible to get information about which keys are causing conflicts in a 3-way-merge.

## 535.2 Scenarios

- **Precondition:** -
- **Main success scenario:** If there is a conflict, the developer should be able to call an API that returns reasonable information wich keys are causing a conflict.
- **Alternative scenario:** If no conflict occurred, the API should indicate so.
- **Error scenario:** -
- **Postcondition:** -
- **Non-functional Constraints:**
  - Getting information about the conflict should not alter the result

# Chapter 536

# Use Case: Configure different hosts with the same playbook

## 536.1 Summary

- **Title:** Configure different hosts with the same playbook
- **Scope:** Configuration
- **Level:** User goal
- **Actors:** User (usually sysadmin)
- **Brief:** It should be possible to configure different hosts with the same playbook. Furthermore, when doing session recording and exporting the merge playbook, the changed keys should be updated in the appropriate spots in the playbook.

## 536.2 Scenarios

- **Precondition:** -
- **Main success scenario:** The user runs a playbook
- **Alternative scenario:**
  - The user changes keys that are specific to this host/group of hosts
  - The user exports a merged playbook
  - The user runs the exported playbook
- **Error scenario:** -
- **Postcondition:** All hosts have their appropriate configuration applied
- **Non-functional Constraints:** -

# Chapter 537

# Use Case: Interactively creating system config

## 537.1 Summary

- **Title:** Interactively creating system config
- **Scope:** Configuration
- **Level:** Summary
- **Actors:** System administrator
- **Brief:** In order to interactively create a workable system configuration, sometimes multiple iterations are required. It should be possible to modify the configuration on a single host, and then apply the changes onto all other hosts. To integrate seamlessly into an Ansible workflow, it should also be possible to merge the changes into the original playbook.

## 537.2 Scenarios

- **Precondition:** -
- **Main success scenario:**
  - User connects to the host and changes settings in Elektra
  - User exports a playbook that contains either
    - only the changed keys
    - the original playbook with the current changes applied
- **Alternative scenario:** If the host was not configured using ansible-elektra, it is only possible to export a playbook with the changed keys
- **Error scenario:** -
- **Postcondition:** A working playbook is exported
- **Non-functional Constraints:** -

# Chapter 538

# Use Case: Recording changes to the key database

## 538.1 Summary

- **Title:** Recording changes to the key database
- **Scope:** Configuration/Logging
- **Level:** User goal
- **Actors:** User (usually a sysadmin)
- **Brief:** Any changes that are made in the key database are recorded.

## 538.2 Scenarios

- **Precondition:**
  - Recording is enabled
  - An active recording session exists
- **Main success scenario:**
  - User makes a change to the key database, e.g. add key, delete key, modify key, modify meta
  - The old & new values for every changed key and metakey are recorded.
- **Alternative scenario:** -
- **Error scenario:** The `kdb` command will print an error message that the changes could not be recorded.
- **Postcondition:** All changed keys and their original values are recorded.
- **Non-functional Constraints:**
  - The recording mechanismn should not noticably slow down Elektra.

# Chapter 539

# Distinction of Use Cases

## 539.1 Scoring / Rating [Scope: Search]

Currently it is not planned to support scoring or rating of snippets. That means it is not possible to retrieve the most viewed, best rated or most often downloaded snippets.

## 539.2 Batch Manipulation

It is not planned to support batch manipulation, i.e. editing of several snippets at once or similar actions. If a feature like this will be implemented at some point, then as frontend only feature, resulting in multiple API requests.

## 539.3 Snippet Conversion

A feature to convert snippets in between formats will be implemented, although it will not support intermediate transformation via script languages like lua/python.

## 539.4 Invalidation of Sessions / Logout

It is not possible to invalidate sessions. This means that although the logout on the frontend is possible, the underlying session would still be active. A stateless server may not store any state information, therefore the session token has to be stored client side. The server has to accept all tokens it gave out for a certain period of time. This is also a known MITM vulnerability that is impossible to fix.

# Chapter 540

# Use Case: Authenticate

## 540.1 Summary

- **Title:** Authenticate
- **Scope:** Authentication
- **Level:** User Goal
- **Actors:** Anonymous user
- **Brief:** Any not logged-in user can authenticate to the service.

## 540.2 Scenarios

- **Precondition:** Website opened
- **Main success scenario:** User successfully authenticates against the server. The websites changes its state to authenticated, which enables more features (e.g. creation of snippets).
- **Alternative scenario:** The authentication was not successful (e.g. wrong credentials). The user is prompted to check their credentials and retry.
- **Error scenario:** Technical problems prevent the authentication from succeeding (e.g. connection issues). The user is informed about the issue.
- **Postcondition:** Website is in authenticated state, meaning that all subsequent requests will use the authentication token to identify to the server.
- **Non-functional Constraints:**
  - Security mechanism
  - Is precondition to use other functions

# Chapter 541

# Use Case: Convert Configuration Snippet

## 541.1 Summary

- **Title:** Convert configuration snippet
- **Scope:** Configuration conversion
- **Level:** User Goal
- **Actors:** Anonymous user (+ all others)
- **Brief:** User can input a configuration snippet in any supported format (e.g. ini, xml) and convert it into another supported format (e.g. JSON).

## 541.2 Scenarios

- **Precondition:** Service supports the in- and output configuration format
- **Main success scenario:** All inputs validate and the configuration snippet could be converted into the target format. The user is then shown the result of the conversion - the input snippet in another format.
- **Alternative scenario:** Wrong inputs lead to an error message and ask the user to correct them, then they can try again. It is also possible that the supplied configuration snippet is of an unsupported format. In either case the user retrieves a response containing error information.
- **Error scenario:** Technical problems prevent the conversion from completing. The user is informed about the issue.
- **Postcondition:** No changes have been made to the database.
- **Non-functional Constraints:**
  - Show-case functionality of Elektra

# Chapter 542

# Use Case: Create Configuration Snippet

## 542.1 Summary

- **Title:** Create configuration snippet
- **Scope:** Entry Management
- **Level:** User Goal
- **Actors:** Authenticated user
- **Brief:** User can create a new configuration snippet (database entry) based on a set of specified inputs, one of them being the snippet itself.

## 542.2 Scenarios

- **Precondition:** Authenticated, sufficient permissions
- **Main success scenario:** All inputs validate and the configuration snippet could be parsed and stored in the Elektra database successfully. The user is then redirected to the detailed view of the created snippet.
- **Alternative scenario:** Wrong inputs lead to an error message and ask the user to correct them, then they can try again. It is also possible that the supplied configuration snippet is of an unsupported format. In either case the user retrieves a response containing error information.
- **Error scenario:** Technical problems prevent the storage process from completing. The user is informed about the issue.
- **Postcondition:** Configuration snippet is stored in the database successfully and can be found through search and direct link.
- **Non-functional Constraints:**
  - Essential functionality of the service

# Chapter 543

# Use Case: Delete Configuration Snippet

## 543.1 Summary

- **Title:** Delete configuration snippet
- **Scope:** Entry Management
- **Level:** User Goal
- **Actors:** Authenticated user (snippet owner), Moderator
- **Brief:** Users can delete their own database entries. Moderators can also delete configuration snippets of other users.

## 543.2 Scenarios

- **Precondition:** Authenticated, sufficient permissions, snippet details page opened
- **Main success scenario:** Configuration snippet has been deleted successfully. A success response is returned and the user, if necessary, redirected to the entry overview.
- **Alternative scenario:** The specified snippet cannot be found (wrong URI) or the user has insufficient permissions to delete it. Either way the user is informed about the error by a response and redirected to the overview if necessary.
- **Error scenario:** Technical problems prevent the deletion process from completing. The user is informed about the issue.
- **Postcondition:** Configuration snippet has been deleted from the database and can neither be found through search, nor be viewed in detail anymore.
- **Non-functional Constraints:**
  - Moderative feature
  - Essential functionality of the service

# Chapter 544

# Use Case: View Details for Specific Configuration Snippet

## 544.1 Summary

- **Title:** View details for specific configuration snippet
- **Scope:** Entry Details
- **Level:** User Goal
- **Actors:** Anonymous user (+ all others)
- **Brief:** Any user can view all details about an entry (configuration snippet).

## 544.2 Scenarios

- **Preconditions:** Snippet has been found through search or URI is known
- **Main success scenario:** User sees detailed information of the requested entry, including tags and other meta information like a description, as well as the configuration snippet converted into all supported formats.
- **Alternative scenario:** The requested entry (URI) cannot be found, therefore the server responds with an error which is displayed to the user. If necessary, the user is redirected to the snippet overview.
- **Error scenario:** Technical problems prevent server from responding with detailed information. The user is informed about the issue.
- **Postcondition:** The database remains unchanged.
- **Non-functional Constraints:**
  - Essential functionality of the program

# Chapter 545

# Use Case: Download Configuration Snippet in Specific Format

## 545.1 Summary

- **Title:** Download configuration snippet in specific format
- **Scope:** Entry Details
- **Level:** User Goal
- **Actors:** Anonymous user (+ all others)
- **Brief:** Any user can download the configuration snippets in various formats (XML, JSON, ini, ...).

## 545.2 Scenarios

- **Precondition:** Entry has been found through search or detailed view of an entry is opened
- **Main success scenario:** User successfully downloads the raw configuration snippet in the selected format. The snippet can then be stored directly as configuration file.
- **Alternative scenario:** The specified entry (URI) cannot be found. The user is informed about the error by an error message.
- **Error scenario:** Technical problems prevent the operation from completing. The user is informed about the error by an error message containing further information.
- **Postcondition:** The database remains unchanged.
- **Non-functional Constraints:**
  - Essential functionality of the program

# Chapter 546

# Use Case: Create Configuration Snippet from Existing Snippet

## 546.1 Summary

- **Title:** Create configuration snippet from existing snippet
- **Scope:** Entry Management
- **Level:** User Goal
- **Actors:** Authenticated user
- **Brief:** User can create a new configuration snippet (database entry) based on an existing snippet. The existing snippet will be filled in the mask for the creation of a new snippet.

## 546.2 Scenarios

- **Precondition:** Authenticated, sufficient permissions
- **Main success scenario:** All inputs validate and the configuration snippet could be parsed and stored in the Elektra database successfully. The user is then redirected to the detailed view of the created snippet.
- **Alternative scenario:** Wrong inputs lead to an error message and ask the user to correct them, then they can try again. It is also possible that the supplied configuration snippet is of an unsupported format. In either case the user retrieves a response containing error information.
- **Error scenario:** Technical problems prevent the storage process from completing. The user is informed about the issue.
- **Postcondition:** Configuration snippet is stored in the database successfully and can be found through search and direct link.
- **Non-functional Constraints:**
  - Can be used to create snippet histories

# Chapter 547

# Use Case: Edit Configuration Snippet

## 547.1 Summary

- **Title:** Edit configuration snippet
- **Scope:** Entry Management
- **Level:** User Goal
- **Actors:** Authenticated user (snippet owner), Moderator
- **Brief:** Users can edit their own database entries. Moderators can edit all configuration snippets in the database, also the ones of other users.

## 547.2 Scenarios

- **Precondition:** Authenticated, sufficient permissions
- **Main success scenario:** Changes to the configuration snippet have been stored to the database successfully. The user is informed about the success.
- **Alternative scenario:** The submitted inputs could not be validated or the entry for which changes are requested cannot be found. The user is informed about the error.
- **Error scenario:** Technical problems prevent the storage process from completing. The user is informed about the issue.
- **Postcondition:** Changes to the configuration snippet are stored in the database successfully and can be immediately seen in new requests of the snippet.
- **Non-functional Constraints:**
  - Moderative feature

# Chapter 548

# Use Case: Edit User

## 548.1 Summary

- **Title:** Edit user
- **Scope:** User Management
- **Level:** User Goal
- **Actors:** Authenticated user, Administrator
- **Brief:** Users can edit their own account information (e.g. change password). Administrators can change all user accounts in the database. Administrators can also change permissions of users.

## 548.2 Scenarios

- **Precondition:** Authenticated, sufficient permissions
- **Main success scenario:** Changes to the user account have been stored to the database successfully. The operating user is informed about the success.
- **Alternative scenario:** The submitted inputs could not be validated or the user entry for which changes are requested cannot be found. The user is informed about the error.
- **Error scenario:** Technical problems prevent the storage process from completing. The operating user is informed about the issue.
- **Postcondition:** Changes to the user account are stored in the database successfully and are from now on valid for further requests (e.g. authentication attempts).
- **Non-functional Constraints:**
  - Administrative feature

# Chapter 549

# Use Case: Register User Account

## 549.1 Summary

- **Title:** Register user account
- **Scope:** Authentication
- **Level:** User Goal
- **Actors:** Anonymous user
- **Brief:** Any not logged-in user can register a new user account.

## 549.2 Scenarios

- **Precondition:**
- **Main success scenario:** The user successfully created a new account. The user is informed about the success.
- **Alternative scenario:** The inputs could not be validated (do not match the required constraints). The user is informed about their mistake.
- **Error scenario:** Technical problems prevent the registration from succeeding. The user is informed about the issue.
- **Postcondition:** The user account is created successfully in the database and the provided credentials can now be used to authenticate against the service.
- **Non-functional Constraints:**
  - Security mechanism
  - Is precondition to use other functions

# Chapter 550

# Use Case: Reset Password

## 550.1 Summary

- **Title:** Reset password
- **Scope:** Authentication
- **Level:** User Goal
- **Actors:** Anonymous user
- **Brief:** Any not logged-in user can reset their password by knowing their username and email address.

## 550.2 Scenarios

- **Precondition:** Website opened
- **Main success scenario:** User successfully resets their password. An email containing the new password will be sent.
- **Alternative scenario:** The entered credentials could not be validated (do not match), which will result in an error being displayed.
- **Error scenario:** Technical problems prevent the password reset from completing. The user is informed about the issue.
- **Postcondition:** User has a newly generated password that they can change after a successful authentication.
- **Non-functional Constraints:**
  - Security mechanism
  - Recovery mechanism

# Chapter 551

# Use Case: Search for Configuration Snippets

## 551.1 Summary

- **Title:** Search for configuration snippets
- **Scope:** Search
- **Level:** User Goal
- **Actors:** Anonymous user (+ all others)
- **Brief:** Any user can use the search to lookup configuration snippets.

## 551.2 Scenarios

- **Precondition:**
- **Main success scenario:** User is presented a list of possible candidates that match the entered search parameters.
- **Alternative scenario:** The search did not find any entry matching the given search parameters, which will result in an empty list being displayed.
- **Error scenario:** Technical problems prevent the search from succeeding. The user is informed about the issue.
- **Postcondition:** The search results offer additional functionality like downloading of configuration snippets or detailed view.
- **Non-functional Constraints:**
  - Essential functionality of the program

# Chapter 552

# Use Case: Search for Users

## 552.1 Summary

- **Title:** Search for users
- **Scope:** Search
- **Level:** User Goal
- **Actors:** Administrators
- **Brief:** Administrators can view and search all registered users.

## 552.2 Scenarios

- **Precondition:** Authenticated, sufficient permissions
- **Main success scenario:** Administrator is displayed a list of possible candidates that match the entered search parameters.
- **Alternative scenario:** The search did not find any user matching the given search parameters, which will result in an empty list being displayed.
- **Error scenario:** Technical problems prevent the search from succeeding. The administrator is informed about the issue.
- **Postcondition:** The search results offer additional functionality like editing and deletion of users.
- **Non-functional Constraints:**
  - Administrative feature

# Chapter 553

# Use Case: Create array specification for dockerd configuration file (daemon.json)

## 553.1 Summary

- **Scope:** `spec`
- **Level:** Developer Goal
- **Actors:** Dev-Ops Engineer
- **Brief:** This use case introduces an array specification for a part of the dockerd configuration file (daemon.↩ json).

## 553.2 Scenarios

- **Precondition:** The Dev-Ops Engineer has a working setup for `docker` for and the daemon `dockerd`.
- **Main success scenario:**
  - The Dev-Ops Engineer wants to write a specification for the `storage options` in the `dockerd` configuration file.
  - The array specification configuration key is `storage/opts/#`.
  - The configuration uses `array/min` and `description` as metakeys.
  - The array is defined by the wildcard character `#`.
  - The keys are all stored for the `spec` namespace.
- **Alternative scenario:**
  - Define the storage type with an underline specification.
  - The array specification configuration key is `storage/opts/_`.
  - The configuration uses `array/min` and `description` as metakeys.
- **Error scenario:**
  - Wrong metakeys are used (yielded as error to the user).
- **Postcondition:** The keys are all stored for the `spec` namespace.
- **Non-functional Constraints:** None.

## 553.3 Example

The `storage options` configuration for the `dockerd` could look like:

```
[storage/opts/#]
meta:/array/min = 0
meta:/description = Storage driver options
```

For the full specification of the `dockerd` configuration file see `dockerd-spec`.

# Chapter 554

# Use Case: Create enum specification for dockerd configuration file (daemon.json)

## 554.1 Summary

- **Scope:** `spec`
- **Level:** Developer Goal
- **Actors:** Dev-Ops Engineer
- **Brief:** This use case introduces a enum specification for a part of the dockerd configuration file (daemon.↩ json).

## 554.2 Scenarios

- **Precondition:** The Dev-Ops Engineer has a working setup for `docker` and the daemon `dockerd`.
- **Main success scenario:**
  - The Dev-Ops Engineer wants to write a specification for the `log level` in the `dockerd` configuration.
  - The configuration key is `log/level`.
  - The configuration key uses `description`, `default` and `enum` as metakeys.
  - The specification strictly defines all enum values `debug`, `info`, `warn`, `error` and `fatal`.
  - The keys are all stored for the `spec` namespace.
- **Alternative scenario:** None.
- **Error scenario:**
  - Wrong metakeys are used (yielded as error to the user).
- **Postcondition:** The keys are all stored for the `spec` namespace..
- **Non-functional Constraints:** None.

## 554.3 Example

The `log level` configuration for the `dockerd` could look like:

```
[log/level]
meta:/description = Set the logging level
meta:/enum/check = #4
meta:/enum/check/#0 = debug
meta:/enum/check/#1 = info
meta:/enum/check/#2 = warn
meta:/enum/check/#3 = error
```

```
meta:/enum/check/#4 = fatal
meta:/default = info
```

In case the key `log/level` does not exist, `spec` plugin creates a `default` key with value `info` in the default namespace.

For the full specification of the `dockerd` configuration file see `dockerd-spec`.

# Chapter 555

# Use Case: Create simple specification for dockerd configuration file (daemon.json)

## 555.1 Summary

- **Scope:** `spec`
- **Level:** Developer Goal
- **Actors:** Dev-Ops Engineer
- **Brief:** This use case introduces a simple specification for a part of the dockerd configuration file (daemon.↩ json).

## 555.2 Scenarios

- **Precondition:** The Dev-Ops Engineer has a working setup for `docker` and the daemon `dockerd`.
- **Main success scenario:**
  - The Dev-Ops Engineer wants to write a specification for the `default runtime` in the `dockerd` configuration.
  - The configuration key is `default/runtime`.
  - The configuration key uses `type`, `description` and `default` as metakeys.
  - The keys are all stored for the `spec` namespace.
- **Alternative scenario:** None.
- **Error scenario:**
  - Wrong metakeys are used (yielded as error to the user).
- **Postcondition:** The keys are all stored for the `spec` namespace..
- **Non-functional Constraints:** None.

## 555.3 Example

The `default runtime` configuration for the `dockerd` could look like:

```
[default/runtime]
meta:/type = string
meta:/description = Default OCI runtime for containers
meta:/default = runc
```

In case the key `default/runtime` does not exist, `spec` plugin creates a `default` key with value `runc` in the default namespace.
For the full specification of the `dockerd` configuration file see `dockerd-spec`.

# Chapter 556

# Use Case: Create underline specification for dockerd configuration file (daemon.json)

## 556.1 Summary

- **Scope:** `spec`
- **Level:** Developer Goal
- **Actors:** Dev-Ops Engineer
- **Brief:** This use case introduces an underline specification for a part of the dockerd configuration file (daemon.json).

## 556.2 Scenarios

- **Precondition:** The Dev-Ops Engineer has a working setup for `docker` and the daemon `dockerd`.
- **Main success scenario:**
  - The Dev-Ops Engineer wants to write a specification for the `runtimes` in the `dockerd` configuration file.
  - The underline specification configuration keys are `runtimes/_`, `runtimes/_/path`, `runtimes/_/runtimeArgs/#`.
  - The specification uses `type`, `description` and `example` as metakeys.
  - The underline configuration is defined by the wildcard character _.
  - The keys are all stored for the `spec` namespace.
- **Alternative scenario:**
  - Define the storage type with an array specification.
  - The array specification configuration keys are `runtimes/#`, `runtimes/#/path`, `runtimes/#/runtimeArgs/#`.
  - The configuration key `runtimes/#` uses `array/min`, `description` and `example` as metakeys.
  - The configuration key `runtimes/#/path` uses `type`, `description` and `example`.
  - The configuration key `runtimes/#/runtimeArgs/#` still uses `array/min`, `description` and `example`.
- **Error scenario:**
  - Wrong metakeys are used (yielded as error to the user).
- **Postcondition:** The keys are all stored for the `spec` namespace..
- **Non-functional Constraints:** None.

## 556.3 Example

The `runtimes` configuration for the `dockerd` could look like:

```
[runtimes/_]
meta:/type = string
meta:/description = Additional OCI compatible runtime
meta:/example = custom
[runtimes/_/path]
meta:/type = string
meta:/description = The path to the OCI compatible runtime
meta:/example = /usr/local/bin/my-runc-replacement
[runtimes/_/runtimeArgs/#]
meta:/array/min = 0
meta:/description = The runtime arguments for the OCI compatible runtime
meta:/example = --debug
```

For the full specification of the `dockerd` configuration file see `dockerd-spec`.

# Chapter 557

# Use Case: <Title>

## 557.1 Summary

- **Scope:** <e.g. Authentication>
- **Level:** <Cloud | Kite | Sea | Underwater | Clam>
- **Actors:** <e.g. API User, Administrator>
- **Brief:** <Short explanation, e.g. User authenticates against the service to gain more privileges>
- **Status:** <Draft | Decided | Implemented>

## 557.2 Scenarios

- **Precondition:** <What has to be satisfied before the use case can work?, e.g. Not authenticated>
- **Main success scenario:** <What is the main scenario, i.e. success scenario?, e.g. Is authenticated>
- **Alternative scenario:** <What is the alternative scenario, i.e. simple error handling?, e.g. Wrong credentials>
- **Error scenario:** <What is the other alternative scenario, i.e. fatal error handling?, e.g. Authentication block because of too many failed attempts>
- **Postcondition:** <What has to be valid after the scenario was run?, e.g. Access to more functions>
- **Non-functional Constraints:**
  - <Some non-functional constraint, e.g. Security mechanism>
  - <More non-functional constraints>

# Chapter 558

# Version

The version of Elektra is handled with the kdb.h macros `KDB_VERSION` which is a string and `KDB_VERSION_MAJOR`, `KDB_VERSION_MINOR` and `KDB_VERSION_PATCH` which are numbers for the publicly announced versions.
The version can also be retrieved at run-time from KDB:

```
system:/elektra/version/constants/KDB_VERSION
system:/elektra/version/constants/KDB_VERSION_MAJOR
system:/elektra/version/constants/KDB_VERSION_MINOR
system:/elektra/version/constants/KDB_VERSION_PATCH
```

All libraries and plugins from Elektra must be installed in exactly the same major and minor version, so no library or plugin-specific version information exists.

## 558.1 Scope

The version applies to following parts of Elektra:

- the API for programs using Elektra. Its interface is declared in src/include/kdb.h. Both applications and plugins use this API.
- the high-level API as declared in `src/include/elektra.h`.
- the API to plugins as declared in `src/plugins/doc/doc.h`.
- the API to `libease`
- the API to `libmeta`
- the API to `libmerge`
- the API to `module loading`
- the API to `libnotification`
- the API to `libopts`
- plugins that have `infos/status` of `compatible`.
- the CLI tool `kdb` with its command-line arguments, KDB access and its return values.

## 558.2 Behavior

The behavior of Elektra is defined by (in that order):

1. ABI test cases
2. Non-ABI test cases (including Shell Recorder)
3. The `API documentation`
4. The man pages

5. Tutorials (excluding Shell Recorder)
6. Examples

Any inconsistency within these artifacts within each other or with the implementation constitutes a bug.

## 558.3 Compatibility

This section describes under which circumstances API and ABI incompatibilities may occur. As Elektra developer your mission is to avoid any unwanted incompatibilities. The tool icheck which checks the interfaces mentioned in the Scope may help you.
Elektra uses a stricter version of Semantic Versioning 2.0.0, with the extensions explained here.
In `1.0.*` the API and ABI must be always forward-compatible and backwards-compatible. That means that Elektra's libraries and plugins may be upgraded and downgraded without any effect on applications, only bug or documentation fixes are allowed.
In minor or patch version updates the API and ABI must be always forward-compatible, but not backwards-compatible. That means that a program written and compiled against 1.0.0 compiles and links against 1.1.0. But because it is not necessarily backwards-compatible a program written for 1.1.0 may not link or compile against Elektra 1.0.0 (but it compiles if you use the compatible subset, maybe by using #ifdefs). Also here applications must continue to work as originally intended.
Changes in Elektra's datastructure will increase (at least) the minor version level. This is required, as the cache plugin exposes the datastructure and must have the same version as the core and other libraries.
When you add a new function you break ABI and API backward- compatibility (but not forward). You are only allowed to do so in a `1.*` change.
In the signature you are only allowed to add const to any parameter. You are *not* allowed to use subtypes to the objects, in C means you are not allowed to call any functions of an object which appear new. C does *not* type check that, it's your responsibility.
What C also does not check are the pre and postconditions. That means you are not allowed to demand more client code (e.g. first accept a NULL pointer and in the next version you crash on it) and you are not allowed to return values that the previous version did not return. It is a complex topic, so better don't underestimate it, but generally said the methods should behave on the same data the same way.
As we use symbol versioning, the SO_VERSION of Elektra always remains the same, even if the major version changes. That means, if we rename or remove a function in the `2.0` release, applications that linked against Elektra `1.0` can still link against Elektra `2.0` but they do not necessarily compile with Elektra `2.0` anymore. Once the code is adapted to use `2.0`, it cannot link against `1.0` anymore. Note that this feature is only available on platforms that support symbol versioning. For other platforms, all applications need to be recompiled for every major version.
In major upgrades, changes of behavior is possible even if it might break some use cases of some applications. Furthermore, symbols that already have been deprecated before, might be removed. For example, in `1.0` there is a deprecated method `ksNext` (not available in `kdb.h`). Thus in `2.0` it is subject to be removed.
References: https://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html https://packages.debian.org/de/sid/icheck

## 558.4 Bindings

Bindings are in general tied to Elektra's version number except for the patch level. So bindings can only add or change functionality when Elektra's core does. This is a quite severe restriction but makes the version much easier to understand for users. It serves the goal that Elektra does not prefer any programming languages, instead people expect from Elektra version `x.y.*` identical functionality.
The patch level can be chosen by bindings as wanted. Using the binding `x.y.z` does not mean that Elektra `x.y.z` will be used on the target system. Please read the changelog of the binding maintainers to know about which bug fixes are included.
The patch level might also be used to fix bugs within bindings. This means that applications can only introspect the patch level of Elektra by getting `system:/elektra/version/constants/KDB_VERSION_PATCH` but not by static patch levels the binding might provide. This should be no problem, as the patch level is supposed to not change the behavior.

# Chapter 559

# VISION

The vision of Elektra is to have systems in which every configuration setting is easily accessible and specifiable. We will use Samba and its configuration value `global/workgroup` as running example.

## 559.1 Problem

Currently, the administrator would first need to locate the configuration file, it could be `/etc/samba/smb.conf`, learn the `syntax` and then implement some configuration management code to add or replace the value. Furthermore, application developers need to implement the parser, design tools like `SWAT`, and document in detail how to configure their specific software. This is a huge effort on both sides.

## 559.2 Configuration Management

We envision, that instead a key-value interface allows us to change any configuration value as desired.
Either by either invoking command-line tools:

```
kdb set system:/sw/samba/#0/current/global/workgroup MYGROUP
```

Also by importing an INI file with the information:

```
kdb import system:/sw/samba/#0/current ini « HERE
[global]
workgroup=MYGROUP
HERE
```

Or using some compiled language like C (shown code uses the high-level API, with neither code generation nor error handling):

```
#include <elektra.h>
int main ()
{
        ElektraError* error;
        Elektra * elektra = elektraOpen ("system:/sw/samba/#0/current", 0, &error);
        elektraSetString (elektra, "global/workgroup", "MYGROUP", &error);
        elektraClose (elektra);
}
```

Or using some interpreted language like Python (shown code uses the Python binding of the low-level API):

```
import kdb
k = kdb.KDB()
ks = kdb.KeySet()
s = "system:/sw/samba/#0/current"
k.get (ks, s)
ks.append(kdb.Key(s+"/global/workgroup", kdb.KEY_VALUE, "MYGROUP"))
k.set (ks, s)
```

Or if you already use configuration management tools, the vision is that a single statement within the configuration management tool suffices to change a configuration value.
Key-value access in `puppet-libelektra`:

```
kdbkey {'system:/sw/samba/#0/current/global/workgroup':
        ensure => 'present',
        value => 'MYGROUP'
}
```

Key-value access in Chef:

```
kdbset 'system:/sw/samba/#0/current/global/workgroup' do
        value 'MYGROUP'
        action :create
end
```

Key-value access in Ansible:

```
- name: setup samba workgroup
```

```
  connection: local
  hosts: localhost
  collections:
    - elektra_initiative.libelektra
  tasks:
    - name: set workgroup
      elektra:
        mountpoint: system:/sw/samba/#0/current/global
        keys:
          workgroup:
            value: MYGROUP
```

In all these examples, we have set `system:/sw/samba/#0/current/global/workgroup` to `MYGROUP`.

## 559.3 Application Integration

Different to other solutions, in Elektra applications itself can be integrated, too. In Samba we would simply replace the configuration file parser with code like ( `low-level C code`, no error-handling and no cleanup):

```
#include <kdb.h>
#include <stdio.h>
int main (void)
{
        KeySet * myConfig = ksNew (0, KS_END);
        Key * key = keyNew ("/sw/samba/#0/current", KEY_END);
        KDB * handle = kdbOpen (NULL, key);
        kdbGet (handle, myConfig, key);
        Key * result = ksLookupByName (myConfig, "/sw/samba/#0/current/global/workgroup", 0);
        printf ("My workgroup is %s", keyString (result));
}
```

If any of the codes above were executed, the application will print `My workgroup is MYGROUP`.
Applications that were modified to directly use Elektra are called to be `elektrified`.
But the vision also includes legacy applications which do not directly use Elektra. Then distributions can mount configuration files, so that the configuration is visible within Elektra.
Elektra uses the same configuration to configure itself, so every way shown above can be used. To make mounting more simple, we introduced an extra tool:

```
kdb mount /etc/samba/smb.conf system:/sw/samba/#0/current ini
```

Mounting can also be done via configuration management tools.
Mounting in puppet-libelektra:

```
kdbmount {'system:/sw/samba/#0/current':
        ensure => 'present',
        file => '/etc/samba/smb.conf',
        plugins => 'ini'
}
kdbkey {'system:/sw/samba/global/log level':
        ensure => 'absent'
}
```

Mounting in Ansible:

```
- name: setup samba workgroup
  connection: local
  hosts: localhost
  tasks:
    - name: set workgroup
    elektra:
      mountpoint: system:/sw/samba
      file: /etc/samba/smb.conf
      plugins: ini
```

## 559.4 Specifications

Ideally, applications specify their settings. This way, we know which keys exist on a system and which values are expected for these keys. Then administrators do not need to guess.
Next to the documentation for humans, specifications also provide information for software. For example, the Web UI automatically gives input fields according to the type of the configuration setting. For example, a boolean configuration setting gets a checkbox.
Also the specifications are integrated within Elektra in the same way. Again, we can use any of the above ways to specify configuration.
Either by either invoking command-line tools:

```
kdb set-meta /sw/samba/#0/current/global/workgroup type string
kdb set-meta /sw/samba/#0/current/global/workgroup description "This controls what workgroup your server
       will appear to be in when queried by clients. Note that this parameter also controls the Domain name
       used with the security = domain setting."
```

Key-value specifications in `puppet-libelektra`:

```
kdbkey {'system:/sw/samba/#0/current/global/workgroup':
        ensure => 'present',
        check => {
                'type' => 'string',
                'default' => 'WORKGROUP',
                'description' => 'This controls what workgroup
                        your server will appear to be in when
                        queried by clients. Note that this
                        parameter also controls the Domain
                        name used with the security = domain
                        setting.'
}
```

Note, that the specification (in both examples above) actually lands up in `spec:/sw/samba/#0/current/global/workgrou`
The unique path to the configuration setting is `/sw/samba/#0/current/global/workgroup`, but the specification gets written to the namespace `spec`, while the system-configuration gets written to the namespace `system`.

## 559.5 Conclusion

Here we have shown how Elektra can be used to configure systems more easily. Both application developers and administrators can save time.
The main technique to achieve the vision is unique key names: Every configuration setting can be addressed by its unique key name.
With this unique key name, we get an identifier, which can be used at all places throughout the system.

- Continue reading big picture

# Chapter 560

# Who uses Elektra?

Elektra has three different main targets.

## 560.1 Desktop

Elektra allows applications to read and write from a global configuration tree. We miss a specification (schema) so that these configuration values can be shared (integrated).
Known users:

- `Oyranos`
- `LCDproc` (in progress)
- `KDE` (in progress)
- `OpenHAB` (in progress)

## 560.2 Embedded

Elektra is on the frontier for embedded systems because of its tiny core and the many possibilities with its plugins.
Known users:

- OpenWRT (distribution)
- Broadcom (blue-ray devices)
- Kapsch (cameras)
- Toshiba (TVs)

## 560.3 Server

Elektra is ideal suited for a local configuration storage by mounting existing configuration files into the global tree. Nodes using Elektra can be connected by already existing configuration management tools.
Known users:

- Allianz
- TU Wien
- Other Universities

## 560.4 Further Readings

- Continue reading: Why should I use Elektra?

# Chapter 561

# Why Should I use Elektra?

The three main points relevant for most people are:

1. Even though Elektra provides a global key database, that allows *read- and write access* of every single parameter *for any application* in an integrated fashion, configuration files stay human read- and writable.
2. Flexible adoption on how the configuration settings are accessed via plugins: you can run arbitrary code in multiple languages or notify others when configuration files are changed. Plugins allow us to support hundreds of different configuration file formats.
3. Elektra allows us to specify configuration values:
    - use the value of other configuration values (symbolic links)
    - calculate the values based on other configuration values
    - validate configuration files
    - generate code based on specifications
    - and much more

## 561.1 Why not Other Solutions?

Some might ask: isn't this solution overkill? Why not tackle these three issues separately? The answer is:

1. If we would only implement a configuration file library for applications, we would hinder the work of administrators and would not provide **external access** to configuration settings or specifications.
2. If we only implement an administrator tool that can parse and generate configuration files, but is not used by the applications themselves, we create a gap that leads to inconsistent understanding of the configuration file **syntax**.
3. If we only specify the meaning of configuration settings within applications but not in a form accessible for administrators, we would create a gap that leads to inconsistent understanding of the configuration settings' **semantics**.

For common understanding of syntax and semantics of configuration files a full-stack solution like Elektra is required. We acknowledge, however, that such a change cannot be done overnight, thus we integrate as many configuration file formats as possible. This way, people can continue using files in `/etc`, regardless of whether or not Elektra is used.
To give one example, in OpenLDAP 2.4.39 the value of `listener-threads` will be reduced to the next number that is a power of 2. To correctly manipulate the setting we need not only know the *syntax* of how to write listener-threads correctly in the configuration file, but also the knowledge how the value is transformed internally. Elektra solves all three issues, and then users can easily **externally** manipulate `listener-threads`, without caring about the concrete **syntax** of the file and getting feedback of the **semantics** (you might get validation errors and you can receive the value exactly as the application will get it).

## 561.2 Unique Features

Features that rarely can be found elsewhere (at least in this combination):

- Bootstrap code and proper abstraction is included:
  - You do not need to worry about the file names of configuration files in the application.
  - Cascading between `/etc`, `$HOME`, `cwd` and so on.
  - You can change which Elektra path is connected to which configuration file with mounting.
  - Portable across OS (Linux, BSD, w64, macOS,.. ) and desktop systems (GNOME, KDE,...).
- No daemon, so no single point of failure but still having guarantees of consistent, validated files with good performance.
- Provides 3-way merging for configuration upgrades.

## 561.3 Further Reasons

- Links and automatic calculation of values: unlike with other solutions you do not need to duplicate configuration values for different applications but you can comfortably link between them which makes many inconsistencies impossible.
- Allows us to easily create GUIs and web-UIs for the whole configuration settings on the system.
- Allows you to import/export all parts of the configuration settings.
- Syntax independence: you can consistently use your favorite syntax.
- Configuration Management (such as Puppet) can be used on top of it without having to fiddle with specifics of every configuration file.
- CLI-tool available
- `kdb editor` allows you to edit any path of Elektra with your favorite syntax (independent of the actual syntax of the configuration files that store values of this path).
- Allows us to also integrate command-line arguments and environment into a consistent place for configuration.
- Reduces huge amount of code: Nearly every application has very similar code:
  - finding the correct configuration file (for different OS)
  - parsing configuration files
  - validating configuration files
  - replace configuration files on changes
- All advantages libraries have:
  - Performance: Improvements in the library benefits all applications.
  - The library only needs to be loaded once in the memory.
  - On fixes not all binaries of all applications need to be replaced.
- All advantages maintained code with a community has:
  - If something does not work, open an issue.
  - If you have a question, open an issue.
  - Regular releases.

## 561.4 Further Readings

- Continue reading the vision.
- Look into the glossary.
- Another viewpoint why to use Elektra is described here
- Compile and Install Elektra

# Chapter 562

# Elektra

*Elektra serves as a universal and secure framework to access configuration settings in a global, hierarchical key database.*
Elektra provides a mature, consistent and easily comprehensible API. Its modularity effectively avoids code duplication across applications and tools concerning their configuration tasks. Elektra abstracts from cross-platform-related issues and enables applications to be aware of other applications' configurations, leveraging easy application integration.

## 562.1 Often Used Links

- If you are new, start reading Get Started
- If you enjoy working with docker take a look at our docker images.
- `Build server`
- `Website`
- `API documentation`

## 562.2 Overview

Elektra provides benefits for:

1. *Application Developers* by making it easier to access configuration settings in a modular, reliable, and extensible way.
2. *System Administrators* by making it possible to access configuration settings in the same way applications access them.
3. *Everyone* by making application integration possible and less misconfiguration a reality.

Elektra consists of three parts:

1. *LibElektra* is a modular configuration access toolkit to construct and integrate applications into a global, hierarchical key database. The building blocks are:
    - language bindings (inclusive high-level interfaces)
    - GenElektra, the code generator for type-safe bindings
    - plugins for configuration access behavior and validation
2. *SpecElektra* is a configuration specification language that is easy to use and self-contained in the same key database (i.e. written in any of the configuration file formats Elektra supports).
3. Tools on top of LibElektra for system administrators, such as CLI tools, web UIs, and GUIs.

To highlight a few concrete things about Elektra, configuration settings can come from any data source, but usually comes from configuration files that are _mounted_ into Elektra similar to mounting a file system. Elektra is a plugin-based framework, for example, plugins implement various configuration formats like INI, JSON, XML, etc. There is a lot more to discover like executing scripts (`python`, `lua` or `shell`) when a configuration value changes, or, enhanced validation plugins that will not allow corrupted configuration settings to reach your application.
As an application developer you get instant access to various configuration formats and the ability to fallback to default configuration settings without having to deal with this on your own. As an system administrator you can choose your favorite configuration format and *mount* this configuration for the application. *Mounting* enables easy application integration as any application using Elektra can access any *mounted* configuration. You can even *mount* `/etc` files such as `hosts` or `fstab`, so that there is no need to configure the same values twice in different files.
In case you are worried about linking to such a powerful library. The core is a small library implemented in C, works cross-platform, and does not need any external dependencies. There are bindings for other languages in case C is too low-level for you.

## 562.3 Contact

Do not hesitate to ask any question on `GitHub issue tracker` or directly to one of the authors.

## 562.4 Quickstart

### 562.4.1 Installation

The preferred way to install Elektra is by using packages provided for your distribution, see INSTALL for available packages and alternative ways for installation.

> Note: It is preferable to use a recent version: They contain many bug fixes and usability improvements.

### 562.4.2 Usage

Now that we have Elektra installed, we can start:

- using the kdb command,
- using qt-gui for people preferring graphical user interfaces, and
- using web-ui for people preferring web user interfaces.

### 562.4.3 Documentation

To get an idea of Elektra, you can take a look at the `presentation`.
In the GitHub repository the full documentation is available, including:

- tutorials,
- FAQ,
- glossary, and
- concepts and man pages

You can read the documentation for the kdb tool, either

- `on the Website`
- `in the API documentation`
- by using `man kdb`
- by using `kdb --help` or `kdb help <command>`
- `on GitHub`

> Note: All these ways to read the documentation provide the same content, all generated from the GitHub repository.

## 562.5 Facts and Features

- Elektra uses simple key-value pairs.
- Elektra uses the BSD licence.
- Elektra implements an API to fully access a global key database.
- Elektra can be thought of a virtual file system for configuration.
- Elektra supports mounting of existing configuration files into a global key database.
- Elektra has dozens of Plugins that make it possible to have a tiny core, but still support many features, including:
  - Elektra can import and export configuration files in any supported format.
  - Elektra is able to log and notify other software on any configuration changes, for example, using Dbus and Journald.
  - Elektra can improve robustness by rejecting invalid configuration via type checking, regex and more.
  - Elektra provides different mechanisms to locate configuration files.
  - Elektra supports different ways to escape and encode content of configuration files.
- Elektra is multi-process safe and can be used in multi-threaded programs.
- Elektra (except for some plugins) is portable and completely written in ANSI C99.
- Elektra (except for some plugins) has no external dependency.
- Elektra is suitable for embedded systems and early boot stage programs.
- Elektra provides many powerful Bindings to avoid low-level access code.
- Elektra provides powerful Code Generation Techniques for high-level configuration access.

## 562.6 News

Go to the website, see the news, and its RSS feed.

## 562.7 Download

Elektra uses a Git repository at GitHub.
You can clone the latest version of Elektra by running:

```
git clone https://github.com/ElektraInitiative/libelektra.git
```

Releases can be downloaded from here.

## 562.8 Build Server

The build server builds Elektra for every pull request and on every commit in various ways and also produces LCOV code coverage report.

## 562.9 Contributing

Take a look at how to start contributing.

## 562.10 Goals

- Make developer's life easier by proving a well-tested mature library instead of rolling your own configuration system for every application. This reduces rank growth of configuration systems (including but not limited to configuration file parsers) in our ecosystem and fosters well-maintained plugins instead.
- Postpone configuration decisions (such as which configuration files to use) from developers to system administrators and package maintainers to provide an overall more consistent and user-friendly system. (Default behavior of applications still is in control of developers, you can even roll your own plugins to provide exactly the same behavior as your application has now.)
- Make configuration storage more safe: avoid that applications receive wrong or unexpected values that could lead to undefined behavior.

And in terms of quality, we want:

1. Simplicity (make configuration tasks, like access of configuration settings, simple),
2. Robustness (no undefined behavior of applications), and
3. Extensibility (gain control over configuration access)

Continue reading about the goals of Elektra

# Chapter 563

# Scripts Index

There should be no scripts top-level but only in sub directories.

### 563.0.1 Scripts For Users

These files are installed on the target system.

- kdb: for scripts to be used with `kdb <script>`.
- ffconfig: to configure firefox.
- completion: for shell completion files.

The completion files need INSTALL_SYSTEM_FILES to be installed.

### 563.0.2 Scripts For Elektra Developers

- dev: for development scripts.
- cmake: everything shared for the CMake build system.
- randoop: For `automatic unit test generation for Java`.

### 563.0.3 Scripts For Build Server

- jenkins
- docker
- vagrant

# Chapter 564

# Deprecated List

**Member elektraInvokeInitialize (const char ∗elektraPluginName)**

Do not use.

**Member kdb::tools::Modules::load (std::string const &pluginName)**

do not use

**Member kdb::tools::Modules::load (std::string const &pluginName, kdb::KeySet const &config)**

do not use

**Member KEY_NULL**

do not use

# Chapter 565

# Module Index

## 565.1 Modules

Here is a list of all modules:

# Chapter 566

# Namespace Index

## 566.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

# Chapter 567

# Hierarchical Index

## 567.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 568

# Class Index

## 568.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 569

# File Index

## 569.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 570

# Module Documentation

## 570.1 High-level API

### Functions

- kdb_boolean_t checkSpec (Key ∗const parentKey, KeySet ∗contract, ElektraError ∗∗error)

  *Verify that specification is properly mounted and is equal to specification at compile time.*
- Elektra ∗ elektraOpen (const char ∗application, KeySet ∗defaults, KeySet ∗contract, ElektraError ∗∗error)

  *Initializes a new Elektra instance.*
- void elektraFatalError (Elektra ∗elektra, ElektraError ∗fatalError)

  *Promote an ElektraError to fatal and call the fatal error handler.*
- Key ∗ elektraHelpKey (Elektra ∗elektra)

  *This function is only intended for use with code-generation.*
- void elektraFatalErrorHandler (Elektra ∗elektra, ElektraErrorHandler fatalErrorHandler)

  *Sets the fatal error handler that will be called, whenever a fatal error occurs.*
- void elektraClose (Elektra ∗elektra)

  *Releases all resources used by the given elektra instance.*
- kdb_long_long_t elektraArraySize (Elektra ∗elektra, const char ∗name)

  *Gets the size of an array.*
- Key ∗ elektraFindArrayElementKey (Elektra ∗elektra, const char ∗name, kdb_long_long_t index, KDBType type)

  *Helper function for code generation.*
- const char ∗ elektraFindReferenceArrayElement (Elektra ∗elektra, const char ∗name, kdb_long_long_t index)

  *Resolves the reference stored in a key.*
- KDBType elektraGetArrayElementType (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Reads the type metadata of a given array element.*
- const char ∗ elektraGetRawStringArrayElement (Elektra ∗elektra, const char ∗name, kdb_long_long_t index)

  *Get the raw string value of an array element key.*
- void elektraSetRawStringArrayElement (Elektra ∗elektra, const char ∗name, kdb_long_long_t index, const char ∗value, KDBType type, ElektraError ∗∗error)

  *Set the raw string value of an array element key.*
- const char ∗ elektraGetStringArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets a string value array element.*
- kdb_boolean_t elektraGetBooleanArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_↩ t index)

  *Gets a boolean value array element.*
- kdb_char_t elektraGetCharArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets a char value array element.*
- kdb_octet_t elektraGetOctetArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets an octet value array element.*

- kdb_short_t elektraGetShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a short value array element.*
- kdb_unsigned_short_t elektraGetUnsignedShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a unsigned short value array element.*
- kdb_long_t elektraGetLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a long value array element.*
- kdb_unsigned_long_t elektraGetUnsignedLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a unsigned long value array element.*
- kdb_long_long_t elektraGetLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a long long value array element.*
- kdb_unsigned_long_long_t elektraGetUnsignedLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a unsigned long long value array element.*
- kdb_float_t elektraGetFloatArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a float value array element.*
- kdb_double_t elektraGetDoubleArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a double value array element.*
- void elektraSetStringArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, const char *value, ElektraError **error)

  *Sets a string value array element.*
- void elektraSetBooleanArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_boolean_t value, ElektraError **error)

  *Sets a boolean value array element.*
- void elektraSetCharArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_char_t value, ElektraError **error)

  *Sets a char value array element.*
- void elektraSetOctetArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_octet_t value, ElektraError **error)

  *Sets an octet value array element.*
- void elektraSetShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_short_t value, ElektraError **error)

  *Sets a short value array element.*
- void elektraSetUnsignedShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_unsigned_short_t value, ElektraError **error)

  *Sets a unsigned short value array element.*
- void elektraSetLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_long_t value, ElektraError **error)

  *Sets a long value array element.*
- void elektraSetUnsignedLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_unsigned_long_t value, ElektraError **error)

  *Sets a unsigned long value array element.*
- void elektraSetLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_long_long_t value, ElektraError **error)

  *Sets a long long value array element.*
- void elektraSetUnsignedLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_unsigned_long_long_t value, ElektraError **error)

  *Sets a unsigned long long value array element.*
- void elektraSetFloatArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_float_t value, ElektraError **error)

*Sets a float value array element.*

- void elektraSetDoubleArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_↩ double_t value, ElektraError ∗∗error)

  *Sets a double value array element.*
- ElektraError ∗ elektraErrorPureWarning (void)

  *Creates a dummy ElektraError struct to store warnings in.*
- const char ∗ elektraErrorCode (const ElektraError ∗error)
- const char ∗ elektraErrorDescription (const ElektraError ∗error)
- void elektraErrorReset (ElektraError ∗∗error)

  *Frees the memory used by the error and sets the referenced error variable to NULL.*
- Key ∗ elektraFindKey (Elektra ∗elektra, const char ∗name, KDBType type)

  *Helper function for code generation.*
- const char ∗ elektraFindReference (Elektra ∗elektra, const char ∗name)

  *Resolves the reference stored in a key.*
- KDBType elektraGetType (Elektra ∗elektra, const char ∗keyname)

  *Reads the type metadata of a given key.*
- const char ∗ elektraGetRawString (Elektra ∗elektra, const char ∗name)

  *Get the raw string value of a key.*
- void elektraSetRawString (Elektra ∗elektra, const char ∗name, const char ∗value, KDBType type, ElektraError ∗∗error)

  *Set the raw string value of a key.*
- const char ∗ elektraGetString (Elektra ∗elektra, const char ∗keyname)

  *Gets a string value.*
- kdb_boolean_t elektraGetBoolean (Elektra ∗elektra, const char ∗keyname)

  *Gets a boolean value.*
- kdb_char_t elektraGetChar (Elektra ∗elektra, const char ∗keyname)

  *Gets a char value.*
- kdb_octet_t elektraGetOctet (Elektra ∗elektra, const char ∗keyname)

  *Gets an octet value.*
- kdb_short_t elektraGetShort (Elektra ∗elektra, const char ∗keyname)

  *Gets a short value.*
- kdb_unsigned_short_t elektraGetUnsignedShort (Elektra ∗elektra, const char ∗keyname)

  *Gets a unsigned short value.*
- kdb_long_t elektraGetLong (Elektra ∗elektra, const char ∗keyname)

  *Gets a long value.*
- kdb_unsigned_long_t elektraGetUnsignedLong (Elektra ∗elektra, const char ∗keyname)

  *Gets a unsigned long value.*
- kdb_long_long_t elektraGetLongLong (Elektra ∗elektra, const char ∗keyname)

  *Gets a long long value.*
- kdb_unsigned_long_long_t elektraGetUnsignedLongLong (Elektra ∗elektra, const char ∗keyname)

  *Gets a long long value.*
- kdb_float_t elektraGetFloat (Elektra ∗elektra, const char ∗keyname)

  *Gets a float value.*
- kdb_double_t elektraGetDouble (Elektra ∗elektra, const char ∗keyname)

  *Gets a double value.*
- void elektraSetString (Elektra ∗elektra, const char ∗keyname, const char ∗value, ElektraError ∗∗error)

  *Sets a string value.*
- void elektraSetBoolean (Elektra ∗elektra, const char ∗keyname, kdb_boolean_t value, ElektraError ∗∗error)

  *Sets a boolean value.*
- void elektraSetChar (Elektra ∗elektra, const char ∗keyname, kdb_char_t value, ElektraError ∗∗error)

  *Sets a char value.*

- void elektraSetOctet (Elektra *elektra, const char *keyname, kdb_octet_t value, ElektraError **error)

  *Sets an octet value.*
- void elektraSetShort (Elektra *elektra, const char *keyname, kdb_short_t value, ElektraError **error)

  *Sets a short value.*
- void elektraSetUnsignedShort (Elektra *elektra, const char *keyname, kdb_unsigned_short_t value, ElektraError **error)

  *Sets a unsigned short value.*
- void elektraSetLong (Elektra *elektra, const char *keyname, kdb_long_t value, ElektraError **error)

  *Sets a long value.*
- void elektraSetUnsignedLong (Elektra *elektra, const char *keyname, kdb_unsigned_long_t value, ElektraError **error)

  *Sets a unsigned long value.*
- void elektraSetLongLong (Elektra *elektra, const char *keyname, kdb_long_long_t value, ElektraError **error)

  *Sets a long long value.*
- void elektraSetUnsignedLongLong (Elektra *elektra, const char *keyname, kdb_unsigned_long_long_t value, ElektraError **error)

  *Sets a unsigned long long value.*
- void elektraSetFloat (Elektra *elektra, const char *keyname, kdb_float_t value, ElektraError **error)

  *Sets a float value.*
- void elektraSetDouble (Elektra *elektra, const char *keyname, kdb_double_t value, ElektraError **error)

  *Sets a double value.*

## 570.1.1 Detailed Description

## 570.1.2 Function Documentation

### 570.1.2.1 checkSpec()

```
kdb_boolean_t checkSpec (
            Key *const parentKey,
            KeySet * contract,
            ElektraError ** error )
```

Verify that specification is properly mounted and is equal to specification at compile time.
Note: These checks are only executed, if the contract requires them.

**Parameters**

| | |
|---|---|
| *parentKey* | The parentKey of the application |
| *contract* | The contract passed to HL API by the application. |
| *error* | Pointer used to report errors |

**Return values**

| | |
|---|---|
| *True* | if the checks were successful (or not required). |
| *False* | if the checks were required but unsuccessful. |

### 570.1.2.2 elektraArraySize()

```
kdb_long_long_t elektraArraySize (
```

```
            Elektra * elektra,
            const char * name )
```

Gets the size of an array.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the array. |

**Returns**

the size of the array, 0 is returned if the array is empty or doesn't exist

#### 570.1.2.3 elektraClose()

```
void elektraClose (
            Elektra * elektra )
```

Releases all resources used by the given elektra instance.
The elektra instance must not be used anymore after calling this.

**Parameters**

| | |
|---|---|
| *elektra* | An Elektra instance. |

#### 570.1.2.4 elektraErrorCode()

```
const char* elektraErrorCode (
            const ElektraError * error )
```

**Returns**

the error code of the given error

#### 570.1.2.5 elektraErrorDescription()

```
const char* elektraErrorDescription (
            const ElektraError * error )
```

**Returns**

the description for the given error

#### 570.1.2.6 elektraErrorPureWarning()

```
ElektraError* elektraErrorPureWarning (
            void  )
```

Creates a dummy ElektraError struct to store warnings in.
If elektraErrorCode() is called on the resulting struct, it will return NULL.

**Returns**

A newly allocated ElektraError (free with elektraFree()).

#### 570.1.2.7 elektraFatalError()

```
void elektraFatalError (
            Elektra * elektra,
            ElektraError * fatalError )
```

Promote an ElektraError to fatal and call the fatal error handler.

**Parameters**

| | |
|---|---|
| *elektra* | Elektra instance whose fatal error handler shall be used. |
| *fatalError* | The error that will be raised. |

#### 570.1.2.8 elektraFatalErrorHandler()

```
void elektraFatalErrorHandler (
            Elektra * elektra,
            ElektraErrorHandler fatalErrorHandler )
```

Sets the fatal error handler that will be called, whenever a fatal error occurs.
Errors occurring in a function, which does not take a pointer to ElektraError, are always considered fatal.
If this function returns, i.e. it does not call exit() or interrupt the thread of execution in some other way, the behaviour of the function from which the error originated is generally undefined.

**Parameters**

| | |
|---|---|
| *elektra* | An Elektra instance. |
| *fatalErrorHandler* | The error handler that will be used henceforth. |

#### 570.1.2.9 elektraFindArrayElementKey()

```
Key * elektraFindArrayElementKey (
            Elektra * elektra,
            const char * name,
            kdb_long_long_t index,
            KDBType type )
```

Helper function for code generation.
Finds an array element Key from its relative name and index. Also checks type metadata, if `type` is not NULL.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The relative name of the array. |
| *index* | The index of the array element. |
| *type* | The expected type metadata value. |

**Returns**

the Key referenced by `name` or NULL, if a fatal error occurs and the fatal error handler returns to this function The returned pointer remains valid until the KeySet inside `elektra` is modified. Calls to elektraSet*() functions may cause such modifications. In any case, it becomes invalid when elektraClose() is called on `elektra`.

#### 570.1.2.10 elektraFindKey()

```
Key * elektraFindKey (
            Elektra * elektra,
            const char * name,
            KDBType type )
```

Helper function for code generation.
Finds a Key from its relative name. Also checks type metadata, if `type` is not NULL.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The relative name of the key. |
| *type* | The expected type metadata value. |

**Returns**

the Key referenced by `name` or NULL, if a fatal error occurs and the fatal error handler returns to this function The returned pointer remains valid until the KeySet inside `elektra` is modified. Calls to elektraSet*() functions may cause such modifications. In any case, it becomes invalid when elektraClose() is called on `elektra`.

#### 570.1.2.11 elektraFindReference()

```
const char * elektraFindReference (
            Elektra * elektra,
            const char * name )
```

Resolves the reference stored in a key.

1. Get the raw string value.
2. Resolve that reference.
3. Return resulting keyname relative to the parent key of the given Elektra instance.

IMPORTANT: this method DOES NOT check the type metadata of the key, it is only intended to be used by the code-generation API.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the key. |

**Returns**

the resolved version of the reference stored in the specified key (relative to the parent key of `elektra`) or NULL, if the key was not found, or the reference resolves two a key not below the parent key. The empty string is returned, if the value was the empty string (no resolution is attempted). The returned pointer becomes invalid when this function is called again (even with the same arguments). It is also invalidated when

elektraFindReferenceArrayElement() or elektraClose() are called on `elektra`.

#### 570.1.2.12 elektraFindReferenceArrayElement()

```
const char * elektraFindReferenceArrayElement (
            Elektra * elektra,
            const char * name,
            kdb_long_long_t index )
```

Resolves the reference stored in a key.

1. Get the raw string value.
2. Resolve that reference.
3. Return resulting keyname relative to the parent key of the given Elektra instance.

IMPORTANT: this method DOES NOT check the type metadata of the key, it is only intended to be used by the code-generation API.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the array. |
| *index* | The index of the array element. |

**Returns**

the resolved version of the reference stored in the specified key (relative to the parent key of `elektra`) or NULL, if the key was not found, or the reference resolves two a key not below the parent key. The empty string is returned, if the value was the empty string (no resolution is attempted). The returned pointer becomes invalid when this function is called again (even with the same arguments). It is also invalidated when elektraFindReference() or elektraClose() are called on `elektra`.

#### 570.1.2.13 elektraGetArrayElementType()

```
KDBType elektraGetArrayElementType (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Reads the type metadata of a given array element.

**Parameters**

| | |
|---|---|
| *elektra* | An Elektra instance. |
| *name* | The name of the array. |
| *index* | The index of the array element whose type information shall be read. |

**Returns**

the KDBType of the key

#### 570.1.2.14 elektraGetBoolean()

```
kdb_boolean_t elektraGetBoolean (
```

```
            Elektra * elektra,
            const char * keyname )
```

Gets a boolean value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the boolean stored at the given key

#### 570.1.2.15 elektraGetBooleanArrayElement()

```
kdb_boolean_t elektraGetBooleanArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a boolean value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the boolean stored at the given array element

#### 570.1.2.16 elektraGetChar()

```
kdb_char_t elektraGetChar (
            Elektra * elektra,
            const char * keyname )
```

Gets a char value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the char stored at the given key

#### 570.1.2.17 elektraGetCharArrayElement()

```
kdb_char_t elektraGetCharArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a char value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the char stored at the given array element

#### 570.1.2.18 elektraGetDouble()

```
kdb_double_t elektraGetDouble (
            Elektra * elektra,
            const char * keyname )
```

Gets a double value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the double stored at the given key

#### 570.1.2.19 elektraGetDoubleArrayElement()

```
kdb_double_t elektraGetDoubleArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a double value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the double stored at the given array element

#### 570.1.2.20 elektraGetFloat()

```
kdb_float_t elektraGetFloat (
            Elektra * elektra,
            const char * keyname )
```

Gets a float value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the float stored at the given key

#### 570.1.2.21 elektraGetFloatArrayElement()

```
kdb_float_t elektraGetFloatArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a float value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the float stored at the given array element

#### 570.1.2.22 elektraGetLong()

```
kdb_long_t elektraGetLong (
            Elektra * elektra,
            const char * keyname )
```

Gets a long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the long stored at the given key

#### 570.1.2.23 elektraGetLongArrayElement()

```
kdb_long_t elektraGetLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the long stored at the given array element

#### 570.1.2.24 elektraGetLongLong()

```
kdb_long_long_t elektraGetLongLong (
            Elektra * elektra,
            const char * keyname )
```

Gets a long long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the long long stored at the given key

#### 570.1.2.25 elektraGetLongLongArrayElement()

```
kdb_long_long_t elektraGetLongLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a long long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the long long stored at the given array element

#### 570.1.2.26 elektraGetOctet()

```
kdb_octet_t elektraGetOctet (
            Elektra * elektra,
            const char * keyname )
```

Gets an octet value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the octet stored at the given key

#### 570.1.2.27 elektraGetOctetArrayElement()

```
kdb_octet_t elektraGetOctetArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets an octet value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the octet stored at the given array element

#### 570.1.2.28 elektraGetRawString()

```
const char * elektraGetRawString (
            Elektra * elektra,
            const char * name )
```

Get the raw string value of a key.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the key. |

**Returns**

the raw value of the specified key or NULL, if the key was not found The returned pointer remains valid until the internal state of `elektra` is modified. Calls to elektraSet∗() functions may cause such modifications. In any case, it becomes invalid when elektraClose() is called on `elektra`.

#### 570.1.2.29 elektraGetRawStringArrayElement()

```
const char * elektraGetRawStringArrayElement (
            Elektra * elektra,
            const char * name,
            kdb_long_long_t index )
```

Get the raw string value of an array element key.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the array. |
| *index* | The index of the array element. |

**Returns**

the raw value of the specified key, or NULL if the key was not found The returned pointer remains valid until the internal state of `elektra` is modified. Calls to elektraSet∗() functions may cause such modifications. In any case, it becomes invalid when elektraClose() is called on `elektra`.

#### 570.1.2.30 elektraGetShort()

```
kdb_short_t elektraGetShort (
            Elektra * elektra,
            const char * keyname )
```

Gets a short value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the short stored at the given key

#### 570.1.2.31 elektraGetShortArrayElement()

```
kdb_short_t elektraGetShortArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a short value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the short stored at the given array element

#### 570.1.2.32 elektraGetString()

```
const char * elektraGetString (
            Elektra * elektra,
            const char * keyname )
```

Gets a string value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the string stored at the given key The returned pointer remains valid until the internal state of `elektra` is modified. Calls to elektraSet∗() functions may cause such modifications. In any case, it becomes invalid when elektraClose() is called on `elektra`.

#### 570.1.2.33 elektraGetStringArrayElement()

```
const char * elektraGetStringArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a string value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the string stored at the given array element The returned pointer remains valid until the internal state of `elektra` is modified. Calls to elektraSet∗() functions may cause such modifications. In any case, it becomes invalid when elektraClose() is called on `elektra`.

#### 570.1.2.34 elektraGetType()

```
KDBType elektraGetType (
            Elektra * elektra,
            const char * keyname )
```

Reads the type metadata of a given key.

**Parameters**

| | |
|---|---|
| *elektra* | An Elektra instance. |
| *keyname* | The name of the key whose type information shall be read. |

**Returns**

the KDBType of the key

#### 570.1.2.35 elektraGetUnsignedLong()

```
kdb_unsigned_long_t elektraGetUnsignedLong (
            Elektra * elektra,
            const char * keyname )
```

Gets a unsigned long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the unsigned long stored at the given key

#### 570.1.2.36 elektraGetUnsignedLongArrayElement()

```
kdb_unsigned_long_t elektraGetUnsignedLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a unsigned long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the unsigned long stored at the given array element

#### 570.1.2.37 elektraGetUnsignedLongLong()

```
kdb_unsigned_long_long_t elektraGetUnsignedLongLong (
            Elektra * elektra,
            const char * keyname )
```

Gets a long long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the unsigned long long stored at the given key

#### 570.1.2.38 elektraGetUnsignedLongLongArrayElement()

```
kdb_unsigned_long_long_t elektraGetUnsignedLongLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a unsigned long long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the unsigned long long stored at the given array element

#### 570.1.2.39 elektraGetUnsignedShort()

```
kdb_unsigned_short_t elektraGetUnsignedShort (
            Elektra * elektra,
            const char * keyname )
```

Gets a unsigned short value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the key to look up. |

**Returns**

the unsigned short stored at the given key

#### 570.1.2.40 elektraGetUnsignedShortArrayElement()

```
kdb_unsigned_short_t elektraGetUnsignedShortArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index )
```

Gets a unsigned short value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to look up. |
| *index* | The index of the array element to look up. |

**Returns**

the unsigned short stored at the given array element

#### 570.1.2.41 elektraHelpKey()

```
Key * elektraHelpKey (
            Elektra * elektra )
```

This function is only intended for use with code-generation.
It looks for the key proc:/elektra/gopts/help (absolute name) created by gopts, and returns it if found.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to check |

**Returns**

the help key if found, NULL otherwise The pointer returned may become invalid, when any `elektraSet*()` function or any other function that modifies the state of `elektra` is called. It will always become invalid, when elektraClose() is called on `elektra`.

#### 570.1.2.42 elektraOpen()

```
Elektra * elektraOpen (
            const char * application,
            KeySet * defaults,
            KeySet * contract,
            ElektraError ** error )
```

Initializes a new Elektra instance.
To free the memory allocated by this function call elektraClose(), once you are done using this instance.

**Parameters**

| | |
|---|---|
| *application* | Your application's base name. The simplest version for this string is "/sw/org/<appname>/#0/current", where '<appname>' is a unique name for your application. For more information see the man-page elektra-key-names(7). |
| *defaults* | A KeySet containing default values. If you pass NULL, trying to read a non-existent value will cause a fatal error. It is recommended, to only pass NULL, if you are using a specification, which provides default values inside of the KDB. If a key in this KeySet doesn't have a value, we will use the value of the "default" metakey of this key. |
| *contract* | Will be passed to kdbOpen() as the contract. |
| *error* | If an error occurs during initialization of the Elektra instance, this pointer will be used to report the error. |

**Returns**

An Elektra instance initialized for the application (free with elektraClose()).

**See also**

elektraClose

kdbOpen

#### 570.1.2.43 elektraSetBoolean()

```
void elektraSetBoolean (
            Elektra * elektra,
            const char * keyname,
            kdb_boolean_t value,
            ElektraError ** error )
```

Sets a boolean value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |

**Parameters**

| | |
|---|---|
| *keyname* | The (relative) name to write to. |
| *value* | The new boolean value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.44 elektraSetBooleanArrayElement()

```
void elektraSetBooleanArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_boolean_t value,
            ElektraError ** error )
```

Sets a boolean value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new boolean value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.45 elektraSetChar()

```
void elektraSetChar (
            Elektra * elektra,
            const char * keyname,
            kdb_char_t value,
            ElektraError ** error )
```

Sets a char value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new char value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.46 elektraSetCharArrayElement()

```
void elektraSetCharArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_char_t value,
            ElektraError ** error )
```

Sets a char value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new char value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.47 elektraSetDouble()

```
void elektraSetDouble (
            Elektra * elektra,
            const char * keyname,
            kdb_double_t value,
            ElektraError ** error )
```

Sets a double value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new double value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.48 elektraSetDoubleArrayElement()

```
void elektraSetDoubleArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_double_t value,
            ElektraError ** error )
```

Sets a double value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new double value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.49 elektraSetFloat()

```
void elektraSetFloat (
            Elektra * elektra,
            const char * keyname,
            kdb_float_t value,
            ElektraError ** error )
```

Sets a float value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new float value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.50 elektraSetFloatArrayElement()

```
void elektraSetFloatArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_float_t value,
            ElektraError ** error )
```

Sets a float value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new float value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.51 elektraSetLong()

```
void elektraSetLong (
            Elektra * elektra,
            const char * keyname,
            kdb_long_t value,
            ElektraError ** error )
```

Sets a long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.52 elektraSetLongArrayElement()

```
void elektraSetLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_long_t value,
            ElektraError ** error )
```

Sets a long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.53 elektraSetLongLong()

```
void elektraSetLongLong (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t value,
            ElektraError ** error )
```

Sets a long long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new long long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.54 elektraSetLongLongArrayElement()

```
void elektraSetLongLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_long_long_t value,
            ElektraError ** error )
```

Sets a long long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new long long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.55 elektraSetOctet()

```
void elektraSetOctet (
            Elektra * elektra,
            const char * keyname,
            kdb_octet_t value,
            ElektraError ** error )
```

Sets an octet value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new octet value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.56 elektraSetOctetArrayElement()

```
void elektraSetOctetArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_octet_t value,
            ElektraError ** error )
```

Sets an octet value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new octet value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.57 elektraSetRawString()

```
void elektraSetRawString (
            Elektra * elektra,
            const char * name,
            const char * value,
            KDBType type,
            ElektraError ** error )
```

Set the raw string value of a key.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the key. |
| *value* | The raw value to set. |
| *type* | The type to set in the metadata of the key. |
| *error* | Pointer to an ElektraError. Will be set in case saving fails. |

#### 570.1.2.58 elektraSetRawStringArrayElement()

```
void elektraSetRawStringArrayElement (
            Elektra * elektra,
            const char * name,
            kdb_long_long_t index,
            const char * value,
```

```
            KDBType type,
            ElektraError ** error )
```

Set the raw string value of an array element key.

**Parameters**

| | |
|---|---|
| *elektra* | The Elektra instance to use. |
| *name* | The (relative) name of the array. |
| *index* | The index of the array element. |
| *value* | The raw value to set. |
| *type* | The type to set in the metadata of the (array element) key. |
| *error* | Pointer to an ElektraError. Will be set in case saving fails. |

#### 570.1.2.59 elektraSetShort()

```
void elektraSetShort (
            Elektra * elektra,
            const char * keyname,
            kdb_short_t value,
            ElektraError ** error )
```

Sets a short value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new short value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.60 elektraSetShortArrayElement()

```
void elektraSetShortArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_short_t value,
            ElektraError ** error )
```

Sets a short value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new short value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.61 elektraSetString()

```
void elektraSetString (
```

```
            Elektra * elektra,
            const char * keyname,
            const char * value,
            ElektraError ** error )
```

Sets a string value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new string value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.62 elektraSetStringArrayElement()

```
void elektraSetStringArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            const char * value,
            ElektraError ** error )
```

Sets a string value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new string value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.63 elektraSetUnsignedLong()

```
void elektraSetUnsignedLong (
            Elektra * elektra,
            const char * keyname,
            kdb_unsigned_long_t value,
            ElektraError ** error )
```

Sets a unsigned long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new unsigned long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.64 elektraSetUnsignedLongArrayElement()

```
void elektraSetUnsignedLongArrayElement (
```

```
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_unsigned_long_t value,
            ElektraError ** error )
```

Sets a unsigned long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new unsigned long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.65 elektraSetUnsignedLongLong()

```
void elektraSetUnsignedLongLong (
            Elektra * elektra,
            const char * keyname,
            kdb_unsigned_long_long_t value,
            ElektraError ** error )
```

Sets a unsigned long long value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new unsigned long long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.66 elektraSetUnsignedLongLongArrayElement()

```
void elektraSetUnsignedLongLongArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_unsigned_long_long_t value,
            ElektraError ** error )
```

Sets a unsigned long long value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new unsigned long long value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.67 elektraSetUnsignedShort()

```
void elektraSetUnsignedShort (
            Elektra * elektra,
            const char * keyname,
            kdb_unsigned_short_t value,
            ElektraError ** error )
```

Sets a unsigned short value.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name to write to. |
| *value* | The new unsigned short value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

#### 570.1.2.68 elektraSetUnsignedShortArrayElement()

```
void elektraSetUnsignedShortArrayElement (
            Elektra * elektra,
            const char * keyname,
            kdb_long_long_t index,
            kdb_unsigned_short_t value,
            ElektraError ** error )
```

Sets a unsigned short value array element.

**Parameters**

| | |
|---|---|
| *elektra* | The elektra instance to use. |
| *keyname* | The (relative) name of the array to write to. |
| *index* | The index of the array element to write to. |
| *value* | The new unsigned short value. |
| *error* | Pass a reference to an ElektraError pointer. Will only be set in case of an error. |

## 570.2 I/O Bindings

Asynchronous I/O feature.

### Files

- file kdbio.h

  *Elektra-I/O structures for I/O bindings, plugins and applications.*
- file kdbioplugin.h

  *Elektra-I/O functions and declarations for the I/O binding test suite.*
- file kdbiotest.h

  *Elektra-I/O functions and declarations for the I/O binding test suite.*

### 570.2.1 Detailed Description

Asynchronous I/O feature.

## 570.2.2 Asynchronous I/O with Elektra

### 570.2.2.1 Overview

I/O bindings allow Elektra and its plugins to integrate into different main loop APIs using a thin abstraction layer. For example, this is used for notification transport plugins which receive notifications using ZeroMQ, D-Bus, etc.
I/O bindings are created using an initialization function for a specific main loop API. Please see `bindings` for available I/O bindings and their according READMEs for more details. After creating, an I/O binding is associated to a KDB instance using elektraIoSetBinding(). Having different I/O bindings (e.g. same or different main loop APIs) for different KDB instances is supported.
The remainder of this page contains useful details for creating I/O bindings and using the operations provided by these bindings. Application developers are normally not required to do those tasks. For more information about using I/O bindings from an application developer perspective please read the `Notification Tutorial`.

### 570.2.2.2 Introduction

An I/O binding needs to handle different types of operations. These operations are used by plugins that require asynchronous I/O. In this document we will call developers of these plugins "users". The three types of operations are:

- file descriptor watch operations
- timer operations
- idle operations

Each operation has a user callback that is called under the following conditions:

- For file descriptor watch operations a user callback shall be called whenever the descriptor status matches the "flags" bitmask (readable, writable or both).
- For timer operations a user callback shall be called whenever a given interval has elapsed.
- Idle operations shall call a user callback whenever possible without interfering other operations. These operations are used for bulk data transfers or expensive calculations that would stall other operations when done at once.

Each operation has different properties. The following properties are shared by all operations:

- callback (elektraIoFdGetCallback();set by elektraIoNewFdOperation()) user callback
- enabled (elektraIoFdIsEnabled();elektraIoFdSetEnabled()) indicates whether the operation is enabled. No callback shall be fired when an operation is disabled.
- privateData (elektraIoFdGetData();set by elektraIoNewFdOperation()) pointer to data owned by user
- bindingData (elektraIoFdGetBindingData();elektraIoFdSetBindingData()) pointer to data owned by I/O binding

For brevity only file descriptor operation variants are listed here. Variants for timer and idle operations are called `elektraIoTimer*` and `elektraIoIdle*`. All `elektraIo*` utility functions are provided by the `elektra-io` library.
File descriptor watch operations have the following additional properties:

- fd (elektraIoFdGetFd();set by elektraIoNewFdOperation()) file descriptor number to be watched
- flags (elektraIoFdGetFlags();elektraIoFdSetFlags()) flags bitmask

Timer operations have the following additional properties:

- interval (elektraIoTimerGetInterval();elektraIoTimerSetInterval()) minimum interval in milliseconds after the user callback shall be fired

Idle operations have no additional properties.

#### Creating a new I/O Binding

Every I/O binding needs to provide ten functions:

- file descriptor watch operations
  - ElektraIoBindingAddFd (e.g. ioDocBindingAddFd())
  - ElektraIoBindingUpdateFd (e.g. ioDocBindingUpdateFd())
  - ElektraIoBindingRemoveFd (e.g. ioDocBindingRemoveFd())
- timer operations
  - ElektraIoBindingAddTimer (e.g. ioDocBindingAddTimer())
  - ElektraIoBindingUpdateTimer (e.g. ioDocBindingUpdateTimer())
  - ElektraIoBindingRemoveTimer (e.g. ioDocBindingRemoveTimer())
- idle operations
  - ElektraIoBindingAddIdle (e.g. ioDocBindingAddIdle())
  - ElektraIoBindingUpdateIdle (e.g ioDocBindingUpdateIdle())
  - ElektraIoBindingRemoveIdle (e.g. ioDocBindingRemoveIdle())
- ElektraIoBindingCleanup (e.g. ioDocBindingCleanup())

In order to create a new I/O binding you have to create an entry point for your binding (e.g. elektraIoDocNew()). This entry point then calls elektraIoNewBinding() with pointers to the ten required functions.

```
        // Initialize I/O interface
        ElektraIoInterface * binding = elektraIoNewBinding (
                // file descriptors
                ioDocBindingAddFd, ioDocBindingUpdateFd, ioDocBindingRemoveFd,
                // timers
                ioDocBindingAddTimer, ioDocBindingUpdateTimer, ioDocBindingRemoveTimer,
                // idle
                ioDocBindingAddIdle, ioDocBindingUpdateIdle, ioDocBindingRemoveIdle,
                // cleanup
                ioDocBindingCleanup);
        if (binding == NULL)
        {
                ELEKTRA_LOG_WARNING ("elektraIoNewBinding failed");
                return NULL;
        }
```

If your I/O management library requires you to store additional data you can do so using elektraIoBindingSetData(). Let's assume you have the following data structure:

```
typedef struct DocBindingData
{
        char * foo;
        // Add additional members as required
} DocBindingData;
```

Then you can store your data with the I/O binding.

```
        // Store binding relevant data in the interface
        DocBindingData * bindingData = elektraMalloc (sizeof (*bindingData));
        if (bindingData == NULL)
        {
                ELEKTRA_LOG_WARNING ("elektraMalloc failed");
                return NULL;
        }
        elektraIoBindingSetData (binding, bindingData);
        bindingData->foo = foo;
```

Of course if you need to store only a single pointer (e.g. a handle) you can omit the struct and directly use elektraIoBindingSetData() with your pointer.

#### Implementing Operations

The next step is to implement operation functions. We'll walk through the implementation of the functions for managing file descriptor watch operations. Timer and idle variants are the same except for the operation properties. For reconstructing the user callback it is advisable to store a context for each operation in your I/O management library. Most I/O management libraries let you pass this context when adding an operation to the library. This

context is then passed by the library back to your callbacks. You can use the operation data itself as context and store additional data like handles from your I/O management library by using elektraIoFdSetBindingData().
Let's assume the data structure looks like this:

```
typedef struct DocOperationData
{
        char * bar;
        // Add additional members as required
} DocOperationData;
```

Using this struct's members you can store additional data like handles in operations. The member `bar` is just an example.
The following snippet from ioDocBindingAddFd() shows example code for ElektraIoBindingAddFd. Code for ElektraIoBindingAddTimer and ElektraIoBindingAddIdle is similar.

```
        DocOperationData * operationData = newOperationData ();
        if (operationData == NULL)
        {
                return 0;
        }
        // You can use private data stored in the I/O binding
        // e.g. MyData data = (MyData *)elektraIoBindingGetData (binding);
        elektraIoFdSetBindingData (fdOp, operationData);
        // You can store additional data for each operation in your operationData structure
        operationData->bar = "foo";
        // Here you need to add the operation to the I/O management library
        // ioDocBindingFdCallback() holds an example callback to pass to your I/O management library
        // assume SomeIoLibHandle * someIoLibAddFd (int fd, int flags, int enabled, void * privateData,
       callback)
        // operationData->handle = someIoLibAddFd (elektraIoFdGetFd (fdOp), elektraIoFdGetFlags (fdOp),
       elektraIoFdIsEnabled (fdOp), &fdOp,
        // ioDocBindingFdCallback)
        return 1;
```

In ElektraIoBindingUpdateFd or ElektraIoBindingRemoveFd you can access your binding operation data by using elektraIoFdGetBindingData().

```
        DocOperationData * operationData = (DocOperationData *) elektraIoFdGetBindingData (fdOp);
```

When your I/O management library detects a change of the file descriptor status it will call a callback supplied by your I/O binding. We will assume for file descriptor watch operations this is ioDocBindingFdCallback(). Your I/O binding's task is to call the operation callback supplied by the user with the correct arguments.

```
/*static*/ void ioDocBindingFdCallback (SomeIoLibHandle * handle, int bitmask)
{
        // For this example let's assume handle is passed as argument
        ELEKTRA_NOT_NULL (handle->data);
        ElektraIoFdOperation * fdOp = (ElektraIoFdOperation *) handle->data;
        // Convert bitmask to Elekta's flags
        elektraIoFdGetCallback (fdOp) (fdOp, someBitMaskToElektraIoFlags (bitmask));
}
```

We assumed `SomeIoLibHandle->data` let's you access your context. Since we have used the original operation data as context we directly obtain the operation data to retrieve the user callback using elektraIoFdGetCallback(). Additionally it is necessary to convert the I/O management library's bitmask to Elekta's I/O bitmask (ElektraIoFdFlags) and then call the user callback.
When implementing ElektraIoBindingRemoveFd (or the timer and idle equivalents) make sure to free data allocated in the add functions.

#### Cleanup

ElektraIoBindingCleanup is the place to free data allocated for your I/O binding.
At least you need to free the pointer returned from elektraIoNewBinding() in your I/O binding's entry point.

#### Linking

Make sure to link against the `elektra-io` library for the `elektraIo*` utility functions that create bindings or operations and allow access to their fields. This library is available via `pkg-config`.

#### Testing

Elektra provides a test suite for I/O bindings in order to make sure that transport plugins will work with all bindings. To run the test suite you need to execute elektraIoTestSuite() and provide the necessary callbacks for creating a new binding, starting and stopping asynchronous processing (ElektraIoTestSuiteCreateBinding, ElektraIoTestSuiteStart and ElektraIoTestSuiteStop).

```
int main (int argc, char ** argv)
{
        init (argc, argv);
        elektraIoTestSuite (createBinding, startLoop, stopLoop);
        print_result ("iowrapper_doc");
        return nbError;
}
```

The functions supplied to elektraIoTestSuite() are called for setup, starting and stopping of the tests.
For example ElektraIoTestSuiteCreateBinding of the "doc" binding:

```
static ElektraIoInterface * createBinding (void)
{
        return elektraIoDocNew ("foo");
}
```

Of course starting and stopping is specific to your I/O management library.

## 570.3 Invoke

Functionality to use plugins and invoke functions.

### Functions

- ElektraInvokeHandle * elektraInvokeInitialize (const char *elektraPluginName)
- ElektraInvokeHandle * elektraInvokeOpen (const char *elektraPluginName, KeySet *config, Key *errorKey)

  *Opens a new handle to invoke functions for a plugin.*
- const void * elektraInvokeGetFunction (ElektraInvokeHandle *handle, const char *elektraPluginFunction↩ Name)

  *Get a function pointer.*
- KeySet * elektraInvokeGetPluginConfig (ElektraInvokeHandle *handle)

  *Get the configuration the plugin uses.*
- const char * elektraInvokeGetPluginName (ElektraInvokeHandle *handle)

  *Get the name of the plugin.*
- void * elektraInvokeGetPluginData (ElektraInvokeHandle *handle)

  *Get the data of the plugin.*
- KeySet * elektraInvokeGetModules (ElektraInvokeHandle *handle)

  *Get the modules used for invoking.*
- KeySet * elektraInvokeGetExports (ElektraInvokeHandle *handle)

  *Get the exports from the plugin.*
- int elektraInvoke2Args (ElektraInvokeHandle *handle, const char *elektraPluginFunctionName, KeySet *ks, Key *k)

  *A convenience function to call a function with two arguments.*
- void elektraInvokeClose (ElektraInvokeHandle *handle, Key *errorKey)

  *Closes all affairs with the handle.*
- int elektraInvokeCallDeferable (ElektraInvokeHandle *handle, const char *elektraPluginFunctionName, Key↩ Set *parameters)

  *Invokes a deferable function on an invoke handle.*
- void elektraInvokeExecuteDeferredCalls (ElektraInvokeHandle *handle, ElektraDeferredCallList *list)

  *Execute deferred calls from list on given invoke handle.*
- int elektraDeferredCall (Plugin *handle, const char *elektraPluginFunctionName, KeySet *parameters)

  *Call a deferrable function on a plugin handle.*
- int elektraDeferredCallAdd (ElektraDeferredCallList *list, const char *name, KeySet *parameters)

  *Add a new deferred call to the deferred call list.*
- ElektraDeferredCallList * elektraDeferredCallCreateList (void)

  *Create new deferred call list.*
- void elektraDeferredCallDeleteList (ElektraDeferredCallList *list)

  *Delete deferred call list.*
- void elektraDeferredCallsExecute (Plugin *plugin, ElektraDeferredCallList *list)

  *Execute deferred calls on given plugin.*

### 570.3.1 Detailed Description

Functionality to use plugins and invoke functions.
Allows invoking functions of plugins as needed within applications and plugins inside and outside of the KDB.
To use this library, you need to include:

```
#include <kdbinvoke.h>`
```

and link against `libelektra-invoke`. Then you can use it:

```
ElektraInvokeHandle * handle = elektraInvokeOpen ("dini", 0);
elektraInvoke2Args (handle, "get", ks, k);
elektraInvokeClose (handle);
```

### 570.3.2 Function Documentation

#### 570.3.2.1 elektraDeferredCall()

```
int elektraDeferredCall (
            Plugin * handle,
            const char * elektraPluginFunctionName,
            KeySet * parameters )
```

Call a deferrable function on a plugin handle.
If the function is exported by the plugin it is directly invoked, if the plugin supports deferring calls, the call is deferred.
If both is possible (function is exported and deferred calls are supported), the function is directly called and the call is deferred (i.e. for nested plugins).

**Parameters**

| | |
|---|---|
| *handle* | invoke handle |
| *elektraPluginFunctionName* | function name |
| *parameters* | parameter key set. Can bee freed afterwards. |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | when the call failed (direct call and deferring not available) |

#### 570.3.2.2 elektraDeferredCallAdd()

```
int elektraDeferredCallAdd (
            ElektraDeferredCallList * list,
            const char * name,
            KeySet * parameters )
```

Add a new deferred call to the deferred call list.
Used internally by plugins.

**Parameters**

| | |
|---|---|
| *list* | deferred call list |
| *name* | function name |
| *parameters* | function parameters |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | when malloc failed |

#### 570.3.2.3 elektraDeferredCallCreateList()

```
ElektraDeferredCallList* elektraDeferredCallCreateList (
            void  )
```

Create new deferred call list.
The list needs to be deleted with elektraDeferredCallDeleteList(). Used internally by plugins.

**Returns**

new list

#### 570.3.2.4 elektraDeferredCallDeleteList()

```
void elektraDeferredCallDeleteList (
            ElektraDeferredCallList * list )
```

Delete deferred call list.
Used internally by plugins.

**Parameters**

| | |
|---|---|
| *list* | list |

#### 570.3.2.5 elektraDeferredCallsExecute()

```
void elektraDeferredCallsExecute (
            Plugin * plugin,
            ElektraDeferredCallList * list )
```

Execute deferred calls on given plugin.
Used internally by plugins.

**Parameters**

| | |
|---|---|
| *plugin* | plugin handle |
| *list* | list |

#### 570.3.2.6 elektraInvoke2Args()

```
int elektraInvoke2Args (
            ElektraInvokeHandle * handle,
            const char * elektraPluginFunctionName,
            KeySet * ks,
            Key * k )
```

A convenience function to call a function with two arguments.

**Parameters**

| | |
|---|---|
| *handle* | the handle to work with |
| *elektraPluginFunctionName* | the function to call, e.g. "get" |
| *ks* | the keyset to be used as first parameter |
| *k* | the key to be used as second parameter |

**Precondition**

handle must be as returned from elektraInvokeOpen()

**Returns**

the return value of the invoked function (i.e. -1, 0, or 1)

**Return values**

| | |
|---|---|
| *-2* | if the function was not found. |

### 570.3.2.7 elektraInvokeCallDeferable()

```
int elektraInvokeCallDeferable (
            ElektraInvokeHandle * handle,
            const char * elektraPluginFunctionName,
            KeySet * parameters )
```

Invokes a deferable function on an invoke handle.
If the function is exported by the plugin it is directly invoked, if the plugin supports deferring calls, the call is deferred.
The parameters key set can be freed afterwards.

**Parameters**

| | |
|---|---|
| *handle* | invoke handle |
| *elektraPluginFunctionName* | function name |
| *parameters* | parameter key set |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | when the call failed (direct call and deferring not available) |

### 570.3.2.8 elektraInvokeClose()

```
void elektraInvokeClose (
            ElektraInvokeHandle * handle,
            Key * errorKey )
```

Closes all affairs with the handle.
The close function of the plugin will be called.

**Parameters**

| | |
|---|---|
| *handle* | the handle to work with |
| *errorKey* | a key where error messages will be stored |

**Precondition**

handle must be as returned from elektraInvokeOpen()

#### 570.3.2.9 elektraInvokeExecuteDeferredCalls()

```
void elektraInvokeExecuteDeferredCalls (
            ElektraInvokeHandle * handle,
            ElektraDeferredCallList * list )
```

Execute deferred calls from list on given invoke handle.
Used internally by plugins holding invoke handles.

**Parameters**

| | |
|---|---|
| *handle* | invoke handle |
| *list* | list |

#### 570.3.2.10 elektraInvokeGetExports()

```
KeySet* elektraInvokeGetExports (
            ElektraInvokeHandle * handle )
```

Get the exports from the plugin.

**Parameters**

| | |
|---|---|
| *handle* | the handle to work with. |

**Precondition**

handle must be as returned from elektraInvokeOpen()

**Returns**

the exports of the plugin.

#### 570.3.2.11 elektraInvokeGetFunction()

```
const void* elektraInvokeGetFunction (
            ElektraInvokeHandle * handle,
            const char * elektraPluginFunctionName )
```

Get a function pointer.

**Parameters**

| | |
|---|---|
| *handle* | the handle to use |
| *elektraPluginFunctionName* | the name of the function to use, e.g., get/set |

**Precondition**

handle must be as returned from elektraInvokeOpen()

Example:

```
typedef int (*elektra2Args) (KeySet*, Key *);
elektra2Args func = *(elektra2Args *)elektraInvokeGetFunction (handle, elektraPluginFunctionName);
```

```
if (!func) exit(1);   // no function found, handle error
func (ks, k);         // otherwise, call function
```

**See also**

elektraInvoke2Args() a convenience function to be used for KeySet,Key arguments.

**Returns**

a function pointer for the specified function.

#### 570.3.2.12 elektraInvokeGetModules()

```
KeySet* elektraInvokeGetModules (
            ElektraInvokeHandle * handle )
```

Get the modules used for invoking.

**Warning**

The modules are closed within elektraInvokeClose(). It is *not* enough to ksDup() the keyset, you must not call elektraInvokeClose() if you want to reuse the modules.

**Parameters**

| | |
|---|---|
| *handle* | the handle to work with. |

**Precondition**

handle must be as returned from elektraInvokeOpen()

**Returns**

the modules used for invoking.

#### 570.3.2.13 elektraInvokeGetPluginConfig()

```
KeySet* elektraInvokeGetPluginConfig (
            ElektraInvokeHandle * handle )
```

Get the configuration the plugin uses.

**Parameters**

| | |
|---|---|
| *handle* | the handle to use |

**Precondition**

handle must be as returned from elektraInvokeOpen()

**Returns**

the config of the plugin.

#### 570.3.2.14 elektraInvokeGetPluginData()

```
void* elektraInvokeGetPluginData (
            ElektraInvokeHandle * handle )
```

Get the data of the plugin.

**Parameters**

| | |
|---|---|
| *handle* | the handle to work with. |

**Precondition**

handle must be as returned from elektraInvokeOpen()

**Returns**

a pointer to the plugin's data.

#### 570.3.2.15 elektraInvokeGetPluginName()

```
const char* elektraInvokeGetPluginName (
            ElektraInvokeHandle * handle )
```

Get the name of the plugin.
The name might differ from the name as passed with elektraInvokeOpen when symlinks are used.

**Parameters**

| | |
|---|---|
| *handle* | the handle to work with. |

**Precondition**

handle must be as returned from elektraInvokeOpen()

**Returns**

the name of the plugin

#### 570.3.2.16 elektraInvokeInitialize()

```
ElektraInvokeHandle* elektraInvokeInitialize (
            const char * elektraPluginName )
```

**Deprecated** Do not use.

Use `elektraInvokeOpen (name, 0, 0)` instead.

**See also**

elektraInvokeOpen()

#### 570.3.2.17 elektraInvokeOpen()

```
ElektraInvokeHandle* elektraInvokeOpen (
            const char * elektraPluginName,
            KeySet * config,
            Key * errorKey )
```

Opens a new handle to invoke functions for a plugin.
When opening the plugin, it calls the "open" function of the plugin.

**Parameters**

| | |
|---|---|
| *elektraPluginName* | the plugin on which we want to invoke functions. |
| *config* | the config to be passed to the plugin. |
| *errorKey* | a key where error messages will be stored |

**Returns**

the handle

**Return values**

| | |
|---|---|
| *0* | on errors |

## 570.4 KDB

General methods to access the Key database.

### Macros

- #define KDB_VERSION "x.y.z"

  *The version information in x.y.z format as string.*
- #define KDB_VERSION_MAJOR x

  *The version information of the major version as number.*
- #define KDB_VERSION_MINOR y

  *The version information of the minor version as number.*
- #define KDB_VERSION_PATCH z

  *The version information of the patch version as number.*

### Functions

- KeySet ∗ ksRenameKeys (KeySet ∗config, const char ∗name)

  *Takes the first key and cuts off this common part for all other keys, instead name will be prepended.*
- KDB ∗ kdbOpen (const KeySet ∗contract, Key ∗errorKey)

  *Opens the session with the Key database.*
- int kdbClose (KDB ∗handle, Key ∗errorKey)

  *Closes the session with the Key database.*
- int kdbGet (KDB ∗handle, KeySet ∗ks, Key ∗parentKey)

  *Retrieve Keys from the Key database in an atomic and universal way.*
- int kdbSet (KDB ∗handle, KeySet ∗ks, Key ∗parentKey)

  *Set Keys to the Key database in an atomic and universal way.*

### 570.4.1 Detailed Description

General methods to access the Key database.
To use them:

```
#include <kdb.h>
```

The kdb∗() methods are used to access the storage, to get and set KeySets.
Parameters common for all these functions are:

- *handle*, as returned by kdbOpen(), need to be passed to every call
- *parentKey* is used for every call to add warnings and set an error. For kdbGet() / kdbSet() it is used to specify which keys should be retrieved/stored.

**Note**

The parentKey is an obligation for you, but only an hint for KDB. KDB does not remember anything about the configuration. You need to pass the same configuration back to kdbSet(), otherwise parts of the configuration get lost. Only keys below the parentKey are subject for change, the rest must be left untouched.

KDB uses different backend implementations that know the details about how to access the storage. One backend consists of multiple plugins. See writing a new plugin for information about how to write a plugin. Backends are state-less regarding the configuration (because of that you must pass back the whole configuration for every backend), but have a state for:

- a two phase-commit
- a conflict detection (error C02000) and
- optimizations that avoid redoing already done operations.

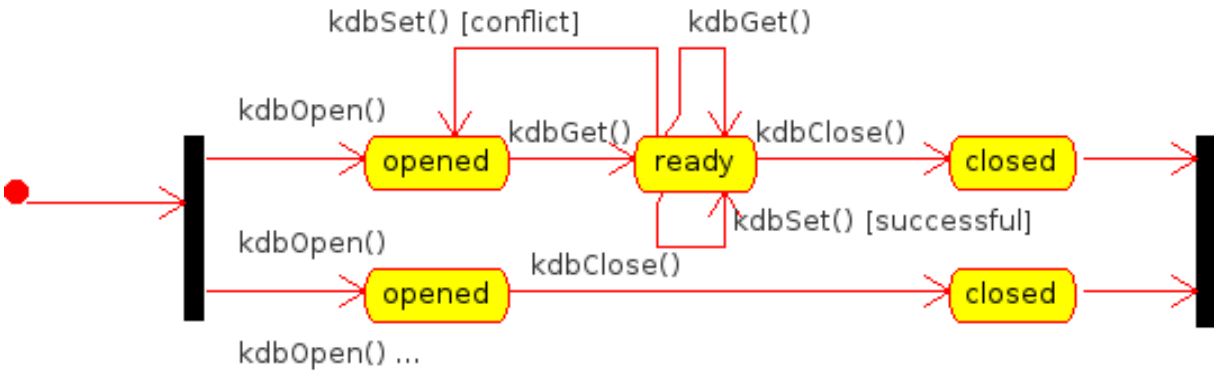



**Figure 570.1 State**

As we see in the figure, kdbOpen() can be called arbitrarily often in any number of threads.
For every handle you got from kdbOpen(), for every parentKey with a different name, *only* the shown state transitions are valid. From a freshly opened KDB, only kdbGet() and kdbClose() are allowed, because otherwise conflicts (error C02000) would not be detected.
Once kdbGet() was called (for a specific handle+parentKey), any number of kdbGet() and kdbSet() can be used with this handle respective parentKey, unless kdbSet() had a conflict (error C02000) with another application. Every affair with KDB needs to be finished with kdbClose().
The name of the parentKey in kdbOpen() and kdbClose() does not matter.
In the usual case we just have one parentKey and one handle. In these cases we just have to remember to use kdbGet() before kdbSet():

```
#include <kdb.h>
#include <stddef.h>
int main (void)
{
        KeySet * myConfig = ksNew (0, KS_END);
        Key * parentKey = keyNew ("/sw/MyApp", KEY_END);
        KDB * handle = kdbOpen (NULL, parentKey);
        kdbGet (handle, myConfig, parentKey); // kdbGet() must be first
        // now any number of any kdbGet()/kdbSet() calls are allowed, e.g.:
        kdbSet (handle, myConfig, parentKey);
        ksDel (myConfig); // delete the in-memory configuration
        kdbClose (handle, parentKey); // no more affairs with the key database.
        keyDel (parentKey);             // working with key/ks does not need kdb
}
```

To output warnings, you can use following code:

```
        Key * cutpoint = keyNew ("meta:/warnings", KEY_END);
        KeySet * warnings = ksCut (keyMeta (warningKey), cutpoint);
        if (!warningKey || ksGetSize (warnings) == 0)
        {
                ksDel (warnings);
                keyDel (cutpoint);
                return 1;
        }
        printf ("There are %zu warnings\n", ksGetSize (warnings));
        elektraCursor i = 1;
        while (i < ksGetSize (warnings))
        {
```

```
                ++i;
                Key * cur = ksAtCursor (warnings, i);
                while (!keyIsDirectlyBelow (cutpoint, cur))
                {
                        printf ("%s: %s\n", keyName (cur) + keyGetNameSize (cutpoint), keyString (cur));
                        ++i;
                        cur = ksAtCursor (warnings, i);
                }
                printf ("\n");
        }
        ksDel (warnings);
        keyDel (cutpoint);
```

To output the error, you can use following code:

```
        const Key * metaError = keyGetMeta (errorKey, "error");
        if (!metaError) return 1; /* There is no current error */
        printf ("number: %s\n", keyString (keyGetMeta (errorKey, "error/number")));
        printf ("description: : %s\n", keyString (keyGetMeta (errorKey, "error/description")));
        printf ("module: : %s\n", keyString (keyGetMeta (errorKey, "error/module")));
        printf ("at: %s:%s\n", keyString (keyGetMeta (errorKey, "error/file")), keyString (keyGetMeta
       (errorKey, "error/line")));
        printf ("reason: : %s\n", keyString (keyGetMeta (errorKey, "error/reason")));
        printf ("mountpoint: : %s\n", keyString (keyGetMeta (errorKey, "error/mountpoint")));
        printf ("configfile: : %s\n", keyString (keyGetMeta (errorKey, "error/configfile")));
```

## 570.4.2 Macro Definition Documentation

### 570.4.2.1 KDB_VERSION

```
#define KDB_VERSION "x.y.z"
```

The version information in x.y.z format as string.
To get the version at run-time, you can get the key system:/elektra/version/constants/KDB_VERSION

**See also**

VERSION.md.

KDB_VERSION_MAJOR

KDB_VERSION_MINOR

KDB_VERSION_PATCH

### 570.4.2.2 KDB_VERSION_MAJOR

```
#define KDB_VERSION_MAJOR x
```

The version information of the major version as number.
To get the version at run-time, you can get the key system:/elektra/version/constants/KDB_VERSION_MAJOR

**See also**

VERSION.md.

KDB_VERSION

### 570.4.2.3 KDB_VERSION_MINOR

```
#define KDB_VERSION_MINOR y
```

The version information of the minor version as number.
To get the version at run-time, you can get the key system:/elektra/version/constants/KDB_VERSION_MINOR

**See also**

VERSION.md.

KDB_VERSION

#### 570.4.2.4 KDB_VERSION_PATCH

```
#define KDB_VERSION_PATCH z
```

The version information of the patch version as number.
To get the version at run-time, you can get the key system:/elektra/version/constants/KDB_VERSION_PATCH

**See also**

VERSION.md.

KDB_VERSION

### 570.4.3 Function Documentation

#### 570.4.3.1 kdbClose()

```
int kdbClose (
            KDB * handle,
            Key * errorKey )
```

Closes the session with the Key database.

**Precondition**

The handle must be a valid handle as returned from kdbOpen()

errorKey must be a valid key, e.g. created with keyNew()

This is the counterpart of kdbOpen().
You must call this method when you are finished working with the Key database. You can manipulate Key and KeySet objects also after kdbClose(), but you must not use any kdb∗() call afterwards.
The `handle` parameter will be finalized and all resources associated to it will be freed. After a kdbClose(), the `handle` cannot be used anymore.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *errorKey* | the key which holds error/warning information |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

kdbOpen() for opening a session with a Key database

#### 570.4.3.2 kdbGet()

```
int kdbGet (
            KDB * handle,
            KeySet * ks,
            Key * parentKey )
```

Retrieve Keys from the Key database in an atomic and universal way.

#### Precondition

The `handle` must be a valid KDB handle as returned from kdbOpen().

The KeySet `returned` must be a valid KeySet, i.e., constructed with ksNew().

The KeySet `returned` must contain keys only from the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/` or `proc:/` namespaces.

The Key `parentKey` must be a valid Key, i.e., constructed with keyNew().

The Key `parentKey` must not have read-only name, value or metadata.

The Key `parentKey` must use the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/`, `proc:/` or cascading namespace.

If you pass `NULL` or a key with read-only metadata as `parentKey`, kdbGet() will fail immediately without doing anything. If you pass another invalid `parentKey`, or `NULL` as `ks` or `handle`, kdbGet() will set an error on `parentKey` and then return immediately.

#### Note

If you pass a non-NULL `parentKey` with writable metadata, kdbGet() will **always** remove any existing errors and warnings from `parentKey`.

#### Warning

If you later call kdbSet() with the same `handle` you must make sure to pass all keys from `ks`, which you do not want to remove.

#### Loadable Namespaces

Not all namespace can be loaded.

- `spec:/`, `dir:/`, `user:/` and `system:/` can be loaded via kdbGet().
- `proc:/` keys can be loaded via kdbGet(), but are not persisted or cached.
- `default:/` keys can be inserted by kdbGet() but they will always stem from a specification in `spec:/` keys.
- If `ks` contains a key with any other namespace, an error will be returned.

#### Parent Key

The `parentKey` defines which parts of `ks` will be loaded. Everything that is at or below `parentKey` wil be loaded together with any key that shares a backend with such a key. Backends are always loaded as an atomic unit.

#### Note

If `parentKey` is in the cascading namespace, keys of all loadable namespaces (see above) will be loaded. This is generally the recommended approach.

Upon sucessfully returning kdbGet() also sets the value of `parentKey` to the storage identifier used by the backend that contains (or would contain) `parentKey`. For file-based backends this is the absolute path of the underlying file. Other backends may use different identifiers, but it always uniquely identifies the underlying storage unit.

#### Note

If `parentKey` is in the cascading, `default:/` or `proc:/ namespace, the value of `parentKey` will be set to an empty string. This is done, because those namespaces are not persistable (see kdbSet()) and therfore have no storage identifier.

#### KeySet Modifications

Below or at `parentKey`, the KeySet `ks` will mostly contain keys loaded from backends. The only exception are `proc:/` and `spec:/` keys that were already present, before kdbGet() was called and do not overlap with an existing backend (for those namespaces). This can be used to provide a hard-coded fallback specifications and/or process-specific data.
Keys not below (or at) `parentKey` that were present when kdbGet() was called, may still be removed. For example, this could be because they overlap with a backend that also has keys below `parentKey` (backends are atomic units).

**Example:**

This example demonstrates the typical usecase within an application (without error handling).

```
#include <kdb.h>
#include <stdio.h>
int main (void)
{
        KeySet * myConfig = ksNew (0, KS_END);
        // for error handling see kdbget_error.c
        // clang-format off
Key * key = keyNew ("/sw/tests/myapp/#0/current/",  KEY_END);
KDB * handle = kdbOpen (NULL, key);
kdbGet (handle, myConfig, key);
Key * result = ksLookupByName (myConfig, "/sw/tests/myapp/#0/current/testkey1", 0);
        // clang-format on
        keyDel (key);
        const char * key_name = keyName (result);
        const char * key_value = keyString (result);
        const char * key_comment = keyString (keyGetMeta (result, "comment/#0"));
        printf ("key: %s value: %s comment: %s\n", key_name, key_value, key_comment);
        ksDel (myConfig); // delete the in-memory configuration
        // maybe you want kdbSet() myConfig here
        kdbClose (handle, 0); // no more affairs with the key database.
}
```

When a backend fails kdbGet() will return -1 with all error and warning information in the `parentKey`. The parameter `returned` will not be changed.

**Optimization:**

In the first run of kdbGet all requested (or more) Keys are retrieved. On subsequent calls only the Keys are retrieved where something was changed inside the Key database. The other Keys stay in the KeySet returned as passed.

It is your responsibility to save the original KeySet if you need it afterwards.
If you want to be sure to get a fresh KeySet again, you need to open a second handle to the Key database using kdbOpen().

#### Parameters

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *parentKey* | Keys below `parentKey` will be retrieved from `handle`. It is also used to add warnings and set error information. |
| *ks* | the (pre-initialized) KeySet returned with all keys found will not be changed on error or if no update is required |

#### Return values

| | |
|---|---|
| *2* | if only `proc:/` backends were executed. This means no data was loaded from storage. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as errors! |
| *1* | if the Keys were retrieved successfully. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as errors! |
| *0* | if there was no update at all - no changes are made to the KeySet then. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as erorrs! |

#### Return values

| | |
|---|---|
| *-1* | on failure - no changes are made to the KeySet then |

#### Since

1.0.0

#### See also

ksLookup(), ksLookupByName() for powerful lookups after the KeySet was retrieved

kdbOpen() which needs to be called before

kdbSet() to save the configuration afterwards

kdbClose() to finish affairs with the Key database.

### 570.4.3.3 kdbOpen()

```
KDB* kdbOpen (
              const KeySet * contract,
              Key * errorKey )
```

Opens the session with the Key database.

#### Precondition

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the `KDB` data structure will be initialized.
The pointer to the `KDB` structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a `KDB` handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

#### Precondition

errorKey must be a valid key, e.g. created with keyNew()

#### Parameters

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

#### Returns

handle to the newly created KDB on success

#### Return values

| | |
|---|---|
| *NULL* | on failure |

#### Since

1.0.0

#### See also

kdbClose() to close the session of a Key database opened by kdbOpen()

### 570.4.3.4 kdbSet()

```
int kdbSet (
            KDB * handle,
            KeySet * ks,
            Key * parentKey )
```

Set Keys to the Key database in an atomic and universal way.

#### Precondition

kdbGet() must be called before kdbSet():

- initially (after kdbOpen())
- after conflict errors in kdbSet().

The KeySet `ks` must be a valid KeySet, i.e., constructed with ksNew().

The KeySet `ks` must only contain only keys in the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/` or `proc:/` namespaces.

The Key `parentKey` must be a valid Key, e.g. constructed with keyNew().

The Key `parentKey` must not have read-only name, value or metadata.

The Key `parentKey` must use the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/`, `proc:/` or cascading namespace.

If you pass `NULL` or a key with read-only metadata as `parentKey`, kdbSet() will fail immediately without doing anything. If you pass another invalid `parentKey`, or `NULL` as `ks` or `handle`, kdbSet() will set an error on `parentKey` and then return immediately.

#### Note

If you pass a non-NULL `parentKey` with writable metadata, kdbSet() will **always** remove any existing errors and warnings from `parentKey`.

#### Persistable Namespaces

Not all namespace can be persisted.

- `spec:/`, `dir:/`, `user:/` and `system:/` will be persisted by kdbSet().
- `default:/` and `proc:/` keys are ignored by kdbSet().
- If `ks` contains a key with any other namespace, an error will be returned.

In general it is recommended to use a `parentKey` in the cascading namespace to cover all namespaces at once.

#### Parent Key

The `parentKey` defines which parts of `ks` will be stored. Everything that is at or below `parentKey` will be persisted together with any key that shares a backend with such a key. Backends are always stored as an atomic unit.

**Note**

If `parentKey` is in the cascading namespace, keys of all persistable namespaces (see above) will be stored. This is generally the recommended approach.

#### KeySet modifications

The contents of `ks` will mostly not be modified by kdbSet(). The only modifications made are those caused by applying the specification in `spec:/` to `dir:/`, `user:/` and `system:/`.

**Errors**

If `parentKey == NULL` or `parentKey` has read-only metadata, kdbSet() will immediately return the error code -1. In all other error cases the following happens:

- Error information will be written into the metadata of the parent key, if possible.
- None of the keys are actually committed in this situation, i.e. no configuration file will be modified.

In case of errors you should present the error message to the user and let the user decide what to do. Possible solutions are:

- remove the problematic key and use kdbSet() again (for validation or type errors)
- change the value of the problematic key and use kdbSet() again (for validation errors)
- do a kdbGet() (for conflicts, i.e. error C02000) and then
  - set the same keyset again (in favour of what was set by this user)
  - drop the old keyset (in favour of what was set from another application)
  - merge the original, your own and the other keyset
- export the configuration into a file (for unresolvable errors)
- repeating the same kdbSet() might be of limited use, if the user does not explicitly request it, because temporary errors are rare and its unlikely that they fix themselves (e.g. disc full, permission problems)

#### Optimization

Only backends that

- contain at least one changed key according to elektraDiffCalculate(),
- contain fewer keys than at the end of kdbGet() will be called. There won't be an unnecessary write for unchanged keys.

If none of the backends needs an update, kdbSet() returns 0 and does nothing.

```
KeySet * myConfig = ksNew (0, KS_END);
Key * parentKey = keyNew ("system:/sw/MyApp", KEY_END);
KDB * handle = kdbOpen (NULL, parentKey);
kdbGet (handle, myConfig, parentKey); // kdbGet needs to be called first!
KeySet * base = ksDup (myConfig);     // save a copy of original keyset
// change the keys within myConfig
ksAppendKey (myConfig, keyNew ("system:/sw/MyApp/Test", KEY_VALUE, "5", KEY_END));
KeySet * ours = ksDup (myConfig); // save a copy of our keyset
KeySet * theirs;                  // needed for 3-way merging
int ret = kdbSet (handle, myConfig, parentKey);
while (ret == -1) // as long as we have an error
{
        int strategy = showElektraErrorDialog (parentKey);
```

```
        theirs = ksDup (ours);
        kdbGet (handle, theirs, parentKey); // refresh key database
        KeySet * result = elektraMerge(
                ksCut(ours, parentKey), parentKey,
                ksCut(theirs, parentKey), parentKey,
                ksCut(base, parentKey), parentKey,
                parentKey, strategy, parentKey);
        int numberOfConflicts = elektraMergeGetConflicts (parentKey);
        ksDel (theirs);
        if (result != NULL) {
                ret = kdbSet (handle, result, parentKey);
        } else {
                // an error happened while merging
                if (numberOfConflicts > 0 && strategy == MERGE_STRATEGY_ABORT)
                {
                        // Error due to merge conflicts
                        ret = -1;
                }
                else
                {
                        // Internal errors, out of memory etc.
                        ret = -1;
                }
        }
}
ksDel (ours);
ksDel (base);
ksDel (myConfig); // delete the in-memory configuration
kdbClose (handle, parentKey); // no more affairs with the key database.
keyDel (parentKey);
```

showElektraErrorDialog() and doElektraMerge() need to be implemented by the user of Elektra. For doElektraMerge a 3-way merge algorithm exists in libelektra-tools.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *ks* | a KeySet which should contain changed keys, otherwise nothing is done |
| *parentKey* | Keys below `parentKey` will be set to `handle`. It is also used to add warnings and set error information. |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | if nothing had to be done, no changes in KDB |
| *-1* | on failure, no changes in KDB, an error will be set on `parentKey` if possible (see "Errors" above) |

**Since**

1.0.0

**See also**

kdbOpen() for getting `handle`

kdbClose() that must be called afterwards

ksCurrent() contains the error Key

### 570.4.3.5 ksRenameKeys()

```
KeySet* ksRenameKeys (
             KeySet * config,
             const char * name )
```

Takes the first key and cuts off this common part for all other keys, instead name will be prepended.

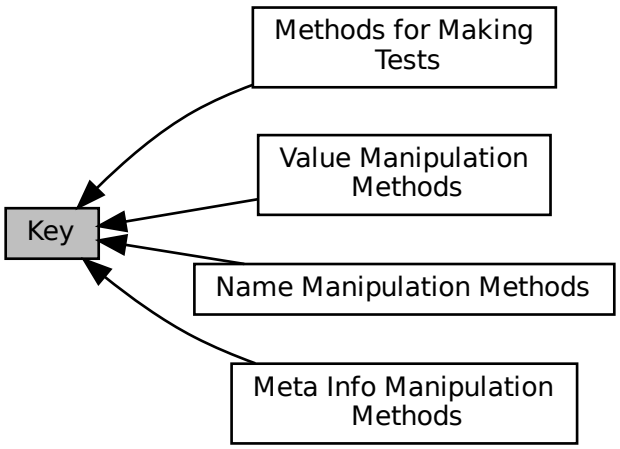
**Returns**

a new allocated keyset with keys in user namespace.

The first key is removed in the resulting keyset.

## 570.5 Key

Key is an essential class that encapsulates key name , value and metainfo .
Collaboration diagram for Key:




### Modules

- Meta Info Manipulation Methods

  *Methods to do various operations on Key metadata.*
- Methods for Making Tests

  *Methods to do various tests on Keys.*
- Name Manipulation Methods

  *Methods to do various operations on Key names.*
- Value Manipulation Methods

  *Methods to do various operations on Key values.*

### Macros

- #define KDB_PATH_SEPARATOR '/'

  */ is used to separate key names.*
- #define KDB_PATH_ESCAPE '\\'

  *\ is used as escape character in the key name.*

### Enumerations

- enum elektraKeyFlags {
  KEY_VALUE = 1 << 1 , KEY_FLAGS = 3 , KEY_BINARY = 1 << 4 , KEY_SIZE = 1 << 11 ,
  KEY_META = 1 << 15 , KEY_NULL = 1 << 16 , KEY_END = 0 }

  *Allows keyNew() to determine which information comes next.*
- enum elektraCopyFlags {
  KEY_CP_NAME = 1 << 0 , KEY_CP_STRING = 1 << 1 , KEY_CP_VALUE = 1 << 2 , KEY_CP_META = 1 << 3 ,
  KEY_CP_ALL = KEY_CP_NAME | KEY_CP_VALUE | KEY_CP_META }

  *Copy options.*

- enum elektraLockFlags { KEY_LOCK_NAME = 1 << 17 , KEY_LOCK_VALUE = 1 << 18 , KEY_LOCK_META = 1 << 19 }

  *Lock options.*
- enum elektraNamespace { KEY_NS_NONE = 0 , KEY_NS_CASCADING = 1 , KEY_NS_META = 2 , KEY_NS_SPEC = 3 , KEY_NS_PROC = 4 , KEY_NS_DIR = 5 , KEY_NS_USER = 6 , KEY_NS_SYSTEM = 7 , KEY_NS_DEFAULT = 8 }

  *Elektra currently supported Key namespaces.*

## Functions

- Key * keyNew (const char *name,...)

  *A practical way to fully create a Key object in one step.*
- Key * keyCopy (Key *dest, const Key *source, elektraCopyFlags flags)

  *Copy or clear a key.*
- int keyDel (Key *key)

  *A destructor for Key objects.*
- int keyClear (Key *key)

  *Will clear all internal data of a Key.*
- uint16_t keyIncRef (Key *key)

  *Increment the reference counter of a Key object.*
- uint16_t keyDecRef (Key *key)

  *Decrement the reference counter of a Key object.*
- uint16_t keyGetRef (const Key *key)

  *Return the current reference counter value of a Key object.*
- int keyLock (Key *key, elektraLockFlags what)

  *Permanently lock parts of a Key.*
- int keyIsLocked (const Key *key, elektraLockFlags what)

  *Checks which parts of a Key are locked.*

### 570.5.1 Detailed Description

Key is an essential class that encapsulates key name , value and metainfo .
To use it include:

```
#include <kdb.h>
```

Key properties are:

- Key name
- Key value
- Key metadata , including but not limited to:
  - Key comment
  - Key owner
  - UID, GID and filesystem-like mode permissions
  - Mode, change and modification times

**ABI**

Due to ABI compatibility, the `Key` structure is not defined in kdb.h, only declared. So you can only declare `pointers` to `Keys` in your program, and allocate and free memory for them with keyNew() and keyDel() respectively.

#### Reference Counting

Every key has its reference counter (see keyGetRef() for longer explanation) that will be initialized with 0, that means a subsequent call of keyDel() will delete the key. If you append the key to a keyset the reference counter will be incremented by one (see keyIncRef()) and the key can't be deleted by a keyDel().

As you can imagine this refcounting allows you to put the Key in your own data structures. It can be a very powerful feature, e.g. if you need your own-defined ordering or different Models of your configuration.

#### Copy-On-Write

Keys employ copy-on-write techniques to minimize memory footprint. If keys are copied or duplicated, they will point at the same name and value as the source key. Only if this data is changed, additional memory is allocated.

### 570.5.2 Macro Definition Documentation

#### 570.5.2.1 KDB_PATH_ESCAPE

```
#define KDB_PATH_ESCAPE '\\'
```

\ is used as escape character in the key name.

**See also**

description about key names .

#### 570.5.2.2 KDB_PATH_SEPARATOR

```
#define KDB_PATH_SEPARATOR '/'
```

/ is used to separate key names.

**See also**

description about key names .

### 570.5.3 Enumeration Type Documentation

#### 570.5.3.1 elektraCopyFlags

```
enum elektraCopyFlags
```

Copy options.

**See also**

keyCopy()

**Enumerator**

| | |
|---|---|
| KEY_CP_NAME | Flag for copying the key name |
| KEY_CP_STRING | Flag for copying the key value, if it is a string |
| KEY_CP_VALUE | Flag for copying the key value |
| KEY_CP_META | Flag for copying the key metadata |
| KEY_CP_ALL | Shorthand for copying name, value and metadata |

#### 570.5.3.2 elektraKeyFlags

`enum elektraKeyFlags`
Allows keyNew() to determine which information comes next.

**See also**

keyNew()

**Enumerator**

| | |
|---|---|
| KEY_VALUE | Flag for the key data |
| KEY_FLAGS | Allows to define multiple flags at once. |
| KEY_BINARY | Flag if the key is binary |
| KEY_SIZE | Flag for maximum size to limit value |
| KEY_META | Flag for metadata |
| KEY_NULL | Is *not* a flag, only as return value<br>**Deprecated** do not use |
| KEY_END | Used as a parameter terminator to keyNew() |

#### 570.5.3.3 elektraLockFlags

`enum elektraLockFlags`
Lock options.

**See also**

keyLock(), keyIsLocked()

**Enumerator**

| | |
|---|---|
| KEY_LOCK_NAME | lock the name of a key |
| KEY_LOCK_VALUE | lock the value of a key |
| KEY_LOCK_META | lock the meta data of a key |

#### 570.5.3.4 elektraNamespace

`enum elektraNamespace`
Elektra currently supported Key namespaces.

**See also**

kdbGet(), keyGetNamespace()

**Enumerator**

| | |
|---|---|
| KEY_NS_NONE | no key given as parameter to keyGetNamespace() |
| KEY_NS_CASCADING | cascading key, starts with /, abstract name for any of the namespaces below |
| KEY_NS_META | metakey, i.e. any key name not under other categories |

**Enumerator**

| | |
|---|---|
| KEY_NS_SPEC | spec contains the specification of the other namespaces |
| KEY_NS_PROC | proc contains process-specific configuration |
| KEY_NS_DIR | dir contains configuration from a specific directory |
| KEY_NS_USER | user key in the home directory of the current user |
| KEY_NS_SYSTEM | system key is shared for a computer system |
| KEY_NS_DEFAULT | default key used as a fallback if no other key is found |

## 570.5.4 Function Documentation

### 570.5.4.1 keyClear()

```
int keyClear (
            Key * key )
```

Will clear all internal data of a Key.
After this call you will receive a fresh Key - with no value, metadata or name.
The reference counter will stay unmodified.

**Note**

that you might also clear() all aliases with this operation.

```
int f (Key *k)
{
        keyClear (k);
        // you have a fresh Key k here
        keySetString (k, "value");
        // the caller will get an empty Key k with an value
}
```

**Postcondition**

`key's` name is "/"

`key's` metadata is empty

**Parameters**

| | |
|---|---|
| *key* | the Key that should be cleared |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

keyDel() for completely deleting a Key

#### 570.5.4.2 keyCopy()

```
Key* keyCopy (
            Key * dest,
            const Key * source,
            elektraCopyFlags flags )
```

Copy or clear a key.
Depending on the chosen `flags` keyCopy() only copies certain parts of `source` into `dest`.

- If KEY_CP_NAME is set, the key name will be copied from `source` to `dest`.
- If KEY_CP_META is set, the meta keys will be copied from `source` to `dest`.
- If KEY_CP_VALUE is set, the key value will be copied from `source` to `dest`. Additionally, if `source` is a binary key (keyIsBinary()), `dest` will also be marked as binary. This means that even if KEY_CP_META is not set, the `binary` meta key will be copied with KEY_CP_VALUE.
- If KEY_CP_STRING is set, the key value will be copied from `source` to `dest`, but only, if `source` is *not* a binary key (keyIsBinary()). If `source` is binary, keyCopy() fails. If `dest` is binary, it will still be marked as binary after the copy. This cannot be used together with KEY_CP_VALUE. The main purpose of KEY_CP_STRING is for copying *into* known string keys. It ensure that you don't accidentally convert string keys into binary keys.

There is also the shorthand KEY_CP_ALL. It is equivalent to `KEY_CP_NAME | KEY_CP_VALUE | KEY_CP↩_META`, i.e. all key data supported by keyCopy() will be copied from `source` to `dest`.
Use this function when you need to copy into an existing key, e.g. because it was passed by a pointer in a function you can do so:

```
keyCopy (copy, orig, KEY_CP_ALL);
```

Most often you will want to duplicate an existing key. For this purpose the alias keyDup() exists. Calling

```
copy = keyDup (orig, KEY_CP_ALL);
```

is equivalent to

```
copy = keyCopy (keyNew ("/", KEY_END), orig, KEY_CP_ALL);
```

The reference counter will not be changed for both keys. Affiliation to keysets are also not affected.
Since metadata uses copy-on-write semantics there is only a constant memory cost to copying metadata.
When you pass a NULL-pointer as `source` the pieces of `dest` specified by `flags` will be cleared.
Calling `keyCopy (dest, NULL, KEY_CP_ALL)` is different from calling keyClear(). The key will not be fully reset, the reference counter and internal flags will remain unchanged. Additionally, keyCopy() respects keyLock() state, while keyClear() always works.

```
keyCopy (k, NULL, KEY_CP_ALL);
// name, value and metadata of k have now been clear
// lock flags, reference count, etc. remain unchanged
```

**Precondition**

`dest` must be a valid Key (created with keyNew)

`dest` must not have read-only flags set

`source` must be a valid Key or NULL

**Invariant**

Key name stays valid until delete

**Postcondition**

Value from Key source is written to Key dest

**Parameters**

| | |
|---|---|
| *dest* | the key which will be written to |
| *source* | the key which should be copied or NULL to clear the data of `dest` |
| *flags* | specifies which parts of the key should be copied |

#### Returns

`dest`

#### Return values

| | |
|---|---|
| *NULL* | on memory allocation problems |
| *NULL* | when a part of `dest` that should be modified (e.g. name, value) was marked read-only, e.g. the name of `dest` will be read-only if `dest` is part of a KeySet |
| *NULL* | when `dest` is NULL |
| *NULL* | when both KEY_CP_VALUE and KEY_CP_STRING are set in `flags` |
| *NULL* | when both KEY_CP_STRING is set in `flags` and `source` is a binary key (keyIsBinary()) |

#### Since

0.9.5

#### See also

keyDup() for duplicating an existing Key

### 570.5.4.3 keyDecRef()

```
uint16_t keyDecRef (
            Key * key )
```

Decrement the reference counter of a Key object.
As long as the reference counter is non-zero, `keyDel()` operations on `key` will be a no-op and return an error code.

#### Postcondition

`key`'s reference counter is >= 0

`key`'s reference counter is < SSIZE_MAX

#### Parameters

| | |
|---|---|
| *key* | the Key object whose reference counter should get decreased |

#### Returns

the updated value of the reference counter

#### Return values

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *0* | when the reference counter already was the minimum value 0, the reference counter will not be modified in this case |

#### Since

1.0.0

**See also**

keyGetRef() to retrieve the current reference count

keyIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

keyDel() for deleting a Key

#### 570.5.4.4 keyDel()

```
int keyDel (
            Key * key )
```

A destructor for Key objects.
Every Key created by keyNew() must be deleted with keyDel().
When the reference counter of `key` is non-zero, this function will do nothing and simply return the current value of the reference counter.
It is therefore safe to call `keyDel (k)` on any `Key * k`.

**Postcondition**

all memory related to `key` will be freed

**Parameters**

| | |
|---|---|
| *key* | the Key object to delete |

**Return values**

| | |
|---|---|
| *0* | when the Key was freed |
| *-1* | on NULL pointers |

**Returns**

the value of the reference counter, if it was non-zero

**Since**

1.0.0

**See also**

keyNew() for creating a new Key

keyIncRef() for more information about the reference counter

#### 570.5.4.5 keyGetRef()

```
uint16_t keyGetRef (
            const Key * key )
```

Return the current reference counter value of a Key object.

**Parameters**

| | |
|---|---|
| *key* | the Key whose reference counter to retrieve |

**Returns**

the value of the `key's` reference counter

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

keyIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

keyDecRef() for decreasing the reference counter

#### 570.5.4.6 keyIncRef()

```
uint16_t keyIncRef (
            Key * key )
```

Increment the reference counter of a Key object.
As long as the reference counter is non-zero, `keyDel()` operations on `key` will be a no-op and return an error code.
Elektra's system for reference counting is not based on a concept of shared ownership. It is more similar to a shared lock, where the counter is used to keep track of how many clients hold the lock.
Initially, the reference counter will be 0. This is can be interpreted as the lock being unlocked. When you increment the reference counter, the lock becomes locked and `keyDel()` is blocked and fails. Only when the reference counter is fully decremented back down to 0 again, will `keyDel()` work again.

**Note**

The reference counter can never exceed `UINT16_MAX - 1`. `UINT16_MAX` is reserved as an error code.

**Postcondition**

`key's` reference counter is > 0

`key's` reference counter is <= UINT16_MAX - 1

**Parameters**

| | |
|---|---|
| *key* | the Key object whose reference counter should be increased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *UINT16_MAX* | when the reference counter already was the maximum value `UINT16_MAX - 1`, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

keyGetRef() to retrieve the current reference count

keyDecRef() for decreasing the reference counter

keyDel() for deleting a Key

#### 570.5.4.7 keyIsLocked()

```
int keyIsLocked (
            const Key * key,
            elektraLockFlags what )
```

Checks which parts of a Key are locked.

**Parameters**

| | |
|---|---|
| *key* | the Key that should be checked for locks |
| *what* | the parts of the Key that should checked for locks |

**Returns**

the bits that are locked

**Return values**

| | |
|---|---|
| *0* | if nothing is locked |
| *-1* | on error (NULL pointer) |

**Since**

1.0.0

**See also**

keyLock() for locking a Key

#### 570.5.4.8 keyLock()

```
int keyLock (
            Key * key,
            elektraLockFlags what )
```

Permanently lock parts of a Key.
This can be:

- KEY_LOCK_NAME to lock the name
- KEY_LOCK_VALUE to lock the value
- KEY_LOCK_META to lock the metadata

To unlock the Key, duplicate it.
It is also possible to lock the Key when it is created with keyNew().
Some data structures need to lock the Key (most likely its name), so that the ordering does not get confused.

#### Postcondition

keyIsLocked(key, what) == what

#### Parameters

| | |
|---|---|
| *key* | the Key that should be locked |
| *what* | the parts of the Key that should be locked (see above) |

#### Returns

the bits that were successfully locked

#### Return values

| | |
|---|---|
| *0* | if everything was locked before |
| *-1* | if it could not be locked (NULL pointer) |

#### Since

1.0.0

#### See also

keyIsLocked() for checking whether a Key is locked

keyNew() for creating a new Key

keyDup() for duplicating an existing Key

ksAppendKey() appends a Key to a KeySet (and locks it)

### 570.5.4.9 keyNew()

```
Key* keyNew (
              const char * name,
               ...  )
```

A practical way to fully create a Key object in one step.
To just get a key object, simple do:

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/some/example", KEY_END);
// work with it
keyDel (k);
```

keyNew() allocates memory for a key object and keyDel() cleans everything up.
If you want the key object to contain a name, value, comment and other meta info read on.

#### Note

When you already have a key with similar properties its easier to keyDup() the key.

You can call keyNew() in many different ways depending on the attribute tags you pass as parameters. Tags are represented as elektraKeyFlags values, and tell keyNew() which Key attribute comes next. The Key attribute tags are the following:

- KEY_VALUE
  Next parameter is a pointer to the value that will be used. If no KEY_BINARY was used before, a string is assumed.

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex0",
        KEY_VALUE, "some data",    // set a string value
        KEY_END);                  // end of args
```

- KEY_SIZE
  Define a maximum length of the value. This is only used when setting a binary key.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex1",
        KEY_SIZE, 4,                // has no effect on strings
        KEY_VALUE, "some data",     // set a string value
        KEY_END);                   // end of args
```

- KEY_META
  Next two parameter is a metaname and a metavalue. See keySetMeta().
```
Key *k=keyNew("user:/tmp/ex3",
        KEY_META, "comment/#0", "a comment",  // with a comment
        KEY_META, "owner", "root",          // and an owner
        KEY_META, "special", "yes",         // and any other metadata
        KEY_END);                   // end of args
```

- KEY_END
  Must be the last parameter passed to keyNew(). It is always required, unless the `keyName` is 0.

- KEY_FLAGS
  Bitwise disjunction of flags, which don't require one or more values. recommended way to set multiple flags. overrides previously defined flags.
```
Key *k=keyNew("user:/tmp/ex3",
        KEY_BINARY,                   // binary key
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",                 // value that will be truncated in 7 bytes
        KEY_END);                       // end of args
```

- KEY_BINARY
  Allows one to change the key to a binary key. Make sure that you also pass KEY_SIZE before you set the value. Otherwise it will be cut off with first \0 in the string. So this flag toggle from keySetString() to keySetBinary(). If no value (nor size) is given, it will be a NULL key.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex2",
        KEY_BINARY,
        KEY_SIZE, 4,                // now the size is important
        KEY_VALUE, "some data",     // sets the binary value ("some")
        KEY_END);                   // end of args
Key *k=keyNew("user:/tmp/ex4",
        KEY_BINARY,                   // key type
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",                 // value that will be truncated in 7 bytes
        KEY_META, "comment/#0", "value is truncated",
        KEY_END);                       // end of args
```

**Precondition**

`name` is a valid Key name

Variable arguments are a valid combination

**Postcondition**

returns a new, fully initialized Key object with the valid Key name and all data given by variable arguments

**Parameters**

| | |
|---|---|
| *name* | a valid name to the key (see keySetName()) |

**Returns**

a pointer to a new allocated and initialized Key object.

**Return values**

| | |
|---|---|
| *NULL* | on allocation error or if an invalid `name` was passed (see keySetName()). |

**Since**

1.0.0

**See also**

keyDel() for deallocating a created Key object
keySetName() for rules about which names are considered valid

## 570.6 KeySet

Methods to manipulate KeySets.

### Macros

- #define KS_END ((Key *) 0)

  *End of a list of keys.*

### Enumerations

- enum elektraLookupFlags { KDB_O_NONE = 0 , KDB_O_DEL = 1 , KDB_O_POP = 1 << 1 }

  *Options to change the default behavior of ksLookup() functions.*

### Functions

- KeySet * ksNew (size_t alloc,...)

  *Allocate, initialize and return a new KeySet object.*
- KeySet * ksVNew (size_t alloc, va_list va)

  *Allocate, initialize and return a new KeySet object.*
- KeySet * ksDup (const KeySet *source)

  *Return a duplicate of a KeySet.*
- int ksCopy (KeySet *dest, const KeySet *source)

  *Replace the content of a KeySet with another one.*
- int ksDel (KeySet *ks)

  *A destructor for KeySet objects.*
- int ksClear (KeySet *ks)

  *Empties a KeySet.*
- uint16_t ksIncRef (KeySet *ks)

  *Increment the reference counter of a KeySet object.*
- uint16_t ksDecRef (KeySet *ks)

  *Decrement the reference counter of a KeySet object.*
- uint16_t ksGetRef (const KeySet *ks)

  *Return the current reference counter value of a KeySet object.*
- ssize_t ksGetSize (const KeySet *ks)

  *Return the number of Keys that `ks` contains.*
- ssize_t ksSearch (const KeySet *ks, const Key *key)

  *Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.*
- ssize_t ksAppendKey (KeySet *ks, Key *toAppend)

  *Appends a Key to the end of `ks`.*
- ssize_t ksAppend (KeySet *ks, const KeySet *toAppend)

  *Append all Keys in `toAppend` to the end of the KeySet `ks`.*
- ssize_t ksRename (KeySet *ks, const Key *root, const Key *newRoot)

  *Moves all keys below `root` to below `newRoot`.*

- elektraCursor ksFindHierarchy (const KeySet *ks, const Key *root, elektraCursor *end)

  *Searches for the start and optionally end of the key hierarchy rooted at `root` in `ks`.*
- KeySet * ksBelow (const KeySet *ks, const Key *root)

  *Retrieves all Keys from KeySet `ks` that are below or at `root`.*
- KeySet * ksCut (KeySet *ks, const Key *cutpoint)

  *Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.*
- Key * ksPop (KeySet *ks)

  *Remove and return the last Key of `ks`.*
- int ksRewind (KeySet *ks)

  *Rewinds the KeySet internal cursor.*
- Key * ksNext (KeySet *ks)

  *Returns the next Key in a KeySet.*
- Key * ksCurrent (const KeySet *ks)

  *Return the current Key.*
- elektraCursor ksGetCursor (const KeySet *ks)

  *Get the internal cursor of the KeySet.*
- Key * ksAtCursor (const KeySet *ks, elektraCursor pos)

  *Return Key at given position `pos`.*
- int ksSetCursor (KeySet *ks, elektraCursor cursor)

  *Set the KeySet internal cursor to `cursor`.*
- Key * ksLookup (KeySet *ks, Key *key, elektraLookupFlags options)

  *Look for a Key contained in `ks` that matches the name of the `key`.*
- Key * ksLookupByName (KeySet *ks, const char *name, elektraLookupFlags options)

  *Convenience method to look for a Key contained in `ks` with name `name`.*
- ssize_t ksSubtract (KeySet *total, const KeySet *sub)

  *Remove all the keys in `sub` from `total`.*

### 570.6.1 Detailed Description

Methods to manipulate KeySets.
A KeySet is a set of keys.
Most important properties of a KeySet:

- Allows us to iterate over all keys (in any depth)
- Iteration is always sorted
- Fast key lookup
- A Key may be shared among many KeySets.

The most important methods of KeySet:

- With ksNew() you can create a new KeySet.
- You can append keys with ksAppendKey() or with ksAppend() you can append a whole keyset.
- Using ksLookup() you can lookup (or pop with KDB_O_POP) a key.
- With ksGetSize() and ksAtCursor() you can iterate through the keyset. Be assured that you will get every key of the set in a stable order (parents before children).

KeySet is the most important data structure in Elektra. It makes it possible to get and store many keys at once inside the database. In addition to that, the class can be used as high level datastructure in applications and it can be used in plugins to manipulate or check configuration.
With ksLookupByName() it is possible to fetch easily specific keys out of the list of keys.
You can easily create and iterate keys:

```
// create a new keyset with 3 keys
```

```
// with a hint that about 20 keys will be inside
KeySet * myConfig = ksNew (20, keyNew ("user:/name1", KEY_END), keyNew ("user:/name2", KEY_END), keyNew
       ("user:/name3", KEY_END), KS_END);
// append a key in the keyset
ksAppendKey (myConfig, keyNew ("user:/name4", KEY_END));
Key * current;
for (elektraCursor it = 0; it < ksGetSize (myConfig); ++it)
{
        current = ksAtCursor (myConfig, it);
        printf ("Key name is %s.\n", keyName (current));
}
ksDel (myConfig); // delete keyset and all keys appended
```

**Copy-on-Write**

Keysets employ copy-on-write techniques to minimize memory footprint. If you create a copy or a duplication of a keyset, the resulting keyset initially references the same data as the source keyset. Only if add or remove keys from a keyset additional memory is allocated.

## 570.6.2 Macro Definition Documentation

### 570.6.2.1 KS_END

```
#define KS_END ((Key *) 0)
```

End of a list of keys.
Use this macro to define the end of a variable-length list of keys.

**See also**

ksNew() and ksVNew()

## 570.6.3 Enumeration Type Documentation

### 570.6.3.1 elektraLookupFlags

```
enum elektraLookupFlags
```

Options to change the default behavior of ksLookup() functions.
These options can be ORed. That is the |-Operator in C.

**See also**

kdbGet(), kdbSet()

**Enumerator**

| | |
|---|---|
| KDB_O_NONE | No Option set.<br>**See also**<br>ksLookup() |
| KDB_O_DEL | Delete parentKey key in ksLookup().<br>**See also**<br>ksLookup() |
| KDB_O_POP | Pop Parent out of keyset key in ksLookup().<br>`@see ksPop().` |

## 570.6.4 Function Documentation

### 570.6.4.1 ksAppend()

```
ssize_t ksAppend (
            KeySet * ks,
            const KeySet * toAppend )
```

Append all Keys in `toAppend` to the end of the KeySet `ks`.
`toAppend` KeySet will be left unchanged.
If a Key is both in `toAppend` and `ks`, the Key in `ks` will be overwritten.

**Postcondition**

Sorted KeySet ks with all Keys it had before and additionally the Keys from toAppend

**Parameters**

| | |
|---|---|
| *ks* | the KeySet that will receive the Keys |
| *toAppend* | the KeySet that provides the Keys that will be transferred |

**Returns**

the size of the KeySet `ks` after transfer

**Return values**

| | |
|---|---|
| *-1* | on NULL pointers |

**Since**

1.0.0

**See also**

ksAppendKey()

### 570.6.4.2 ksAppendKey()

```
ssize_t ksAppendKey (
            KeySet * ks,
            Key * toAppend )
```

Appends a Key to the end of `ks`.
Hands the ownership of the Key `toAppend` to the KeySet `ks`. ksDel(ks) uses keyDel(k) to delete every Key unless it got its reference counter incremented by keyIncRef(), e.g. by another KeySet that contains this Key.
The reference counter of the Key will be incremented to indicate this ownership, and thus `toAppend` is not const.

**See also**

keyGetRef()

If the Key's name already exists in the KeySet, it will be replaced with the new Key.
ksAppendKey() will also lock the Key's name from `toAppend`. This is necessary so that the order of the KeySet cannot be destroyed via calls to keySetName().
The KeySet internal cursor will be set to the new Key.
It is safe to directly append newly created Keys:

```
KeySet * ks = ksNew (1, KS_END);
ksAppendKey (ks, keyNew ("user:/my/new/key", KEY_END));
ksDel (ks);
// key deleted, too!
```

If you want the key to outlive the KeySet, make sure to do proper ref counting:

```
KeySet * ks = ksNew (1, KS_END);
Key * k = keyNew ("user:/ref/key", KEY_END);
keyIncRef (k);
ksAppendKey (ks, k);
ksDel (ks);
// now we still can work with the key k!
keyDecRef (k);
keyDel (k);
```

You can duplicate the Key to avoid aliasing, but then the Key in the KeySet has another identity:

```
KeySet * ks = ksNew (1, KS_END);
Key * k = keyNew ("user:/ref/key", KEY_END);
ksAppendKey (ks, keyDup (k, KEY_CP_ALL));
ksDel (ks);
// now we still can work with the key k!
keyDel (k);
```

**Parameters**

| | |
|---|---|
| *ks* | KeySet where `toAppend` should be append |
| *toAppend* | Key that will be appended to `ks` or deleted |

**Returns**

the size of the KeySet after appending

**Return values**

| | |
|---|---|
| *-1* | on NULL pointers |
| *-1* | if appending failed (only on memory problems). The Key will be deleted then. |

**Since**

1.0.0

**See also**

ksAppend() for appending a KeySet to another KeySet

keyIncRef() for manually increasing a Key's reference counter

### 570.6.4.3 ksAtCursor()

```
Key* ksAtCursor (
                const KeySet * ks,
                elektraCursor pos )
```

Return Key at given position `pos`.
The position is a number starting from 0.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet to get the Key from |
| *pos* | the position of the Key that should be retrieved |

**Returns**

the Key at the cursor `pos` on success

**Return values**

| | |
|---|---|
| *NULL* | on NULL pointer, negative cursor position or a position that does not lie within the KeySet `ks` |

**Since**

1.0.0

#### 570.6.4.4 ksBelow()

```
KeySet* ksBelow (
            const KeySet * ks,
            const Key * root )
```

Retrieves all Keys from KeySet `ks` that are below or at `root`.
This function works very much like ksCut(). It returns an identical KeySet, but the Keys also remain in `ks`

**Parameters**

| | |
|---|---|
| *ks* | the Keyset to copy from |
| *root* | the point where to copy from the Keyset |

**Returns**

a new allocated KeySet which needs to be deleted with ksDel(). The KeySet consists of all Keys (of the original KeySet ks) below `root`. If `root` exists, it will also be appended.

#### 570.6.4.5 ksClear()

```
int ksClear (
            KeySet * ks )
```

Empties a KeySet.
This function

- **does not** check or modify the reference count of `ks`
- decrements the reference count of all keys contained in `ks`
- deletes all keys that where only referenced by `ks`
- resets size of `ks` to 0

**Parameters**

| | |
|---|---|
| *ks* | the KeySet to clear |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on failure (memory) or ks == NULL |

#### 570.6.4.6 ksCopy()

```
int ksCopy (
            KeySet * dest,
            const KeySet * source )
```

Replace the content of a KeySet with another one.
Most often you may want a duplicate of a KeySet, see ksDup() or append keys, see ksAppend(). In some situations you need to copy Keys from a KeySet to another KeySet, for which this function exists.

**Note**

You can also use it to clear a KeySet when you pass a NULL pointer as `source`.

**Implementation:**

First all Keys in `dest` will be deleted. Afterwards the content of `source` will be added to the destination.

A flat copy is made, so Keys will not be duplicated, but their reference counter is updated, so both KeySets need to be deleted via ksDel().

```
int f (KeySet *ks)
{
        KeySet *c = ksNew (20, ..., KS_END);
        // c receives keys
        ksCopy (ks, c); // pass the KeySet to the caller
        ksDel (c);
}       // caller needs to ksDel (ks)
```

**Parameters**

| | |
|---|---|
| *source* | an initialized KeySet or NULL |
| *dest* | an initialized KeySet, where the Keys from `source` get copied to |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | if `dest` was cleared successfully (`source` is NULL) |
| *-1* | when `dest` is a NULL pointer |

**Since**

1.0.0

**See also**

ksNew() for creating a new KeySet

ksDel() for deleting an existing KeySet

ksDup() for duplicating an existing KeySet

keyCopy() for copying Keys

#### 570.6.4.7 ksCurrent()

```
Key* ksCurrent (
            const KeySet * ks )
```

Return the current Key.
The returned pointer is NULL if you reached the end or after ksRewind().

**Note**

You must not delete the Key or change the Key, use ksPop() if you want to delete it.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object to get the current Key from |

**Returns**

pointer to the Key pointed by `ks's` cursor

**Return values**

| | |
|---|---|
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

ksNext() to get the next Key in the KeySet

ksRewind() for resetting the internal cursor of the KeySet

### 570.6.4.8 ksCut()

```
KeySet* ksCut (
            KeySet * ks,
            const Key * cutpoint )
```

Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.
Searches for the `cutpoint` inside the KeySet `ks`. If found, it cuts out this Key and everything which is below (see keyIsBelow()) this Key. These Keys will be missing in the keyset `ks`. Instead, they will be moved to the returned KeySet. If `cutpoint` is not found an empty KeySet is returned and `ks` is not changed.
The cursor will stay at the same Key as it was before. If the cursor was inside the region of cut (moved) Keys, the cursor will be set to the Key before the `cutpoint`.
If you use ksCut() on a KeySet you got from kdbGet() and plan to use kdbSet() later, make sure that you keep all Keys that should not be removed permanently. You have to keep the KeySet that was returned and the KeySet `ks`.

**Example:**

You have the keyset `ks`:

- `system:/mountpoint/interest`
- `system:/mountpoint/interest/folder`
- `system:/mountpoint/interest/folder/key1`
- `system:/mountpoint/interest/folder/key2`
- `system:/mountpoint/other/key1`

When you use

```
Key * parentKey = keyNew ("system:/mountpoint/interest", KEY_END);
KDB * kdb = kdbOpen (NULL, parentKey);
KeySet * ks = ksNew (0, KS_END);
kdbGet (kdb, ks, parentKey);
KeySet * returned = ksCut (ks, parentKey);
kdbSet (kdb, ks, parentKey); // all keys below cutpoint are now removed
kdbClose (kdb, parentKey);
```

Then in `returned` are:

- `system:/mountpoint/interest`
- `system:/mountpoint/interest/folder`
- `system:/mountpoint/interest/folder/key1`
- `system:/mountpoint/interest/folder/key2`

And in `ks` are:

- `system:/mountpoint/other/key1`

So kdbSet() permanently removes all keys at or below `system:/mountpoint/interest`.

**Parameters**

| | |
|---|---|
| *ks* | the Keyset to cut. It will be modified by removing all Keys at or below the cutpoint. |
| *cutpoint* | the point where to cut out the Keyset |

**Returns**

a new allocated KeySet which needs to be deleted with ksDel(). The KeySet consists of all Keys (of the original KeySet ks) below the cutpoint. If the Key cutpoint exists, it will also be appended.

**Return values**

| | |
|---|---|
| *0* | on NULL pointers, no Key name or allocation problems |

**Since**

1.0.0

**See also**

kdbGet() for an explanation on why you might get more Keys than you requested.

### 570.6.4.9 ksDecRef()

```
uint16_t ksDecRef (
            KeySet * ks )
```

Decrement the reference counter of a KeySet object.
As long as the reference counter is non-zero, `ksDel()` operations on `key` will be a no-op and return an error code.

**Parameters**

| | |
|---|---|
| *key* | the KeySet object whose reference counter should get decreased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *0* | when the reference counter already was the minimum value 0, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

ksGetRef() to retrieve the current reference count

ksIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

ksDel() for deleting a Key

### 570.6.4.10 ksDel()

```
int ksDel (
            KeySet * ks )
```

A destructor for KeySet objects.
Every KeySet created by ksNew() must be deleted with ksDel().
When the reference counter of `ks` is non-zero, this function will do nothing and simply return the current value of the reference counter.
It is therefore safe to call `ksDel (ks)` on any `KeySet * ks`.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object to delete |

**Return values**

| | |
|---|---|
| *0* | when the KeySet was freed |
| *-1* | on NULL pointers |

**Returns**

the value of the reference counter, if it was non-zero

**Since**

1.0.0

**See also**

ksNew() for creating a new KeySet

ksIncRef() for more information about the reference counter

### 570.6.4.11 ksDup()

```
KeySet* ksDup (
            const KeySet * source )
```

Return a duplicate of a KeySet.
Objects created with ksDup() must be destroyed with ksDel().
Memory will be allocated as needed for dynamic properties, so you need to ksDel() the returned pointer.
A flat copy is made, so the Keys will not be duplicated, but their reference counter is updated, so both KeySets need to be deleted via ksDel().

**Parameters**

| | |
|---|---|
| *source* | has to be an initialized KeySet |

**Returns**

a flat copy of source on success

**Return values**

| | |
|---|---|
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

ksNew() for creating a new KeySet

ksDel() for deleting a KeySet

keyDup() for Key duplication

### 570.6.4.12 ksFindHierarchy()

```
elektraCursor ksFindHierarchy (
              const KeySet * ks,
              const Key * root,
              elektraCursor * end )
```

Searches for the start and optionally end of the key hierarchy rooted at `root` in `ks`.
The hierarchy will only contain keys in the same namespace as `root`. If `root` is a cascading key, only cascading keys will be part of the hierarchy.
The main use-case for this function is this kind of loop:

```
elektraCursor end;
for (elektraCursor it = ksFindHierarchy (ks, root, &end); it < end; ++it)
{
        Key * cur = ksAtCursor (ks, it);
}
```

**Parameters**

| | |
|---|---|
| *ks* | The keyset to search in |
| *root* | The root of the hierachy to find |
| *end* | If this is not NULL, it will be set to position of the first key after `root` that is not below `root`. This is useful for loops like the one above. If not keys below `root` exist in `ks`, `end` will always be set to the size of `ks`. This way a loop like the one above will still work correctly. |

**Return values**

| | |
|---|---|
| *-1* | if `ks` or `root` are NULL |

**Returns**

the position of either `root` itself or the first key below `root` that is part of `ks`. If no keys below `root` exist in `ks`, the size of `ks` is returned. The snippet above shows why this is useful.

#### 570.6.4.13 ksGetCursor()

```
elektraCursor ksGetCursor (
              const KeySet * ks )
```

Get the internal cursor of the KeySet.

**Warning**

Cursors are getting invalid when the Key was ksPop()ed or ksLookup() with KDB_O_POP was used.

### 570.6.5 Read ahead

With the cursors it is possible to read ahead in a KeySet:

```
elektraCursor jump;
ksRewind (ks);
while ((key = keyNextMeta (ks))!=0)
{
        // now mark this key
        jump = ksGetCursor(ks);
        //code..
        keyNextMeta (ks); // now browse on
        // use ksCurrent(ks) to check the keys
        //code..
        // jump back to the position marked before
        ksSetCursor(ks, jump);
}
```

### 570.6.6 Restoring state

It can also be used to restore the state of a KeySet in a function

```
int f (KeySet *ks)
{
        elektraCursor state = ksGetCursor(ks);
        // work with keyset
        // now bring the keyset to the state before
        ksSetCursor (ks, state);
}
```

It is of course possible to make the KeySet const and cast its const away to set the cursor. Another way to achieve the same is to ksDup() the KeySet, but it is not as efficient.
An invalid cursor will be returned directly after ksRewind(). When you set an invalid cursor ksCurrent() is 0.

### 570.6.7 Using Cursor directly

You can also use the cursor directly by initializing it to some index in the KeySet and then incrementing or decrementing it, to iterate over the KeySet.

```
        Key * cur;
        for (elektraCursor cursor = 0; (cur = ksAtCursor (ks, cursor)) != NULL; ++cursor)
        {
                printf ("%s\n", keyName (cur));
        }
```

You can also use a while loop if you need access to the last cursor position.

```
        elektraCursor cursor = 0;
        Key * cur;
        while ((cur = ksAtCursor (ks, cursor)) != 0)
        {
                printf ("%s\n", keyName (cur));
                ++cursor;
        }
```

**Note**

Only use a cursor for the same KeySet which it was made for.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object to get the cursor from |

**Returns**

a valid cursor on success

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |
| *-1* | on an invalid internal cursor or after ksRewind |

**Since**

1.0.0

**See also**

ksNext() for moving the internal cursor forward

ksSetCursor() for setting the cursor to a specific position

ksAtCursor() for getting the Key at a specific position

#### 570.6.7.1 ksGetRef()

```
uint16_t ksGetRef (
            const KeySet * ks )
```

Return the current reference counter value of a KeySet object.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet whose reference counter to retrieve |

**Returns**

the value of the `key's` reference counter

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

ksIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

ksDecRef() for decreasing the reference counter

#### 570.6.7.2 ksGetSize()

```
ssize_t ksGetSize (
            const KeySet * ks )
```

Return the number of Keys that `ks` contains.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object to get the size from |

**Returns**

the number of Keys that `ks` contains.

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |

**Since**

1.0.0

### 570.6.7.3 ksIncRef()

```
uint16_t ksIncRef (
            KeySet * ks )
```

Increment the reference counter of a KeySet object.
As long as the reference counter is non-zero, `ksDel()` operations on `key` will be a no-op and return an error code.
Elektra's system for reference counting is not based on a concept of shared ownership. It is more similar to a shared lock, where the counter is used to keep track of how many clients hold the lock.
Initially, the reference counter will be 0. This can be interpreted as the lock being unlocked. When you increment the reference counter, the lock becomes locked and `ksDel()` is blocked and fails. Only when the reference counter is fully decremented back down to 0 again, will `ksDel()` work again.

**Note**

The reference counter can never exceed `UINT16_MAX - 1`. `UINT16_MAX` is reserved as an error code.

**Postcondition**

`ks's` reference counter is > 0

`ks's` reference counter is <= UINT16_MAX - 1

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object whose reference counter should be increased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *UINT16_MAX* | when the reference counter already was the maximum value `UINT16_MAX - 1`, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

ksGetRef() to retrieve the current reference count

ksDecRef() for decreasing the reference counter

ksDel() for deleting a Key

#### 570.6.7.4 ksLookup()

```
Key* ksLookup (
              KeySet * ks,
              Key * key,
              elektraLookupFlags options )
```

Look for a Key contained in `ks` that matches the name of the `key`.

**Note**

Applications should only use ksLookup() with cascading Keys (Key name starting with `/`). Furthermore, a lookup should be done for every Key (also when iterating over Keys) so that the specifications are honored correctly. Keys of all namespaces need to be present so that ksLookup() can work correctly, so make sure to also use kdbGet() with a cascading Key.

ksLookup() is designed to let you work with a KeySet containing all Keys of the application. The idea is to fully kdbGet() the whole configuration of your application and process it all at once with many `ksLookup()`.
This function is efficient (at least using binary search). Together with kdbGet(), which you can use to load the whole configuration, you can write very effective and short code for configuration:

```
Key * key = keyNew ("/sw/tests/myapp/#0/current/",  KEY_END);
KDB * handle = kdbOpen (NULL, key);
kdbGet (handle, myConfig, key);
Key * result = ksLookupByName (myConfig, "/sw/tests/myapp/#0/current/testkey1", 0);
```

This is the way programs should get their configuration and search for the values. It is guaranteed, that more namespaces can be added easily and that all values can be set by admin and user. Furthermore, using the kdb-tool, it is possible to introspect which values an application will get (by doing the same cascading lookup).
If found, a pointer to the Key is returned. If not found a NULL pointer is returned.
Cascading lookups will by default search in all namespaces (proc:/, dir:/, user:/ and system:/), but will also correctly consider the specification (=metadata) in spec:/:

- `override/#` will make sure that another Key is considered before
- `namespace/#` will change the number and/or order in which the namespaces are searched
- `fallback/#` will search for other Keys when the other possibilities up to now were not successful
- `default` to return the given value when not even `fallback` Keys were found.

**Note**

override and fallback work recursively, while default does not.

This process is very flexible, but it would be boring to manually follow all this links to find out which Key will be taken in the end. Use `kdb get -v` to trace the Keys.

**KDB_O_POP**

When KDB_O_POP is set the Key which was found will be ksPop()ed.

#### Note

Like in ksPop() the popped Key always needs to be keyDel() afterwards, even if it is appended to another KeySet.

```
void f (KeySet * iterator, KeySet * lookup)
{
        KeySet * append = ksNew (ksGetSize (lookup), KS_END);
        ssize_t ksSize = ksGetSize (iterator);
        for (elektraCursor it = 0; it < ksSize; ++it)
        {
                Key * current = ksAtCursor (iterator, it);
                Key * key = ksLookup (lookup, current, KDB_O_POP);
                // do something...
                ksAppendKey (append, key); // now append it to append, not lookup!
                keyDel (key);                // make sure to ALWAYS delete poped keys.
        }
        ksAppend (lookup, append);
        // now lookup needs to be sorted only once, append never
        ksDel (append);
}
```

This is also a nice example how a complete application with ksLookup() can look like.

#### KDB_O_DEL

Passing KDB_O_DEL will cause the deletion of the parameter `key` using keyDel().

#### Hybrid search

When Elektra is compiled with `ENABLE_OPTIMIZATIONS=ON` a hybrid search decides dynamically between the binary search and the OPMPHM. The hybrid search can be overruled by passing KDB_O_OPMPHM or KDB_O_BINSEARCH in the options to ksLookup().

#### Parameters

| | |
|---|---|
| *ks* | the KeySet that should be searched |
| *key* | the Key object you are looking for |
| *options* | of type elektraLookupFlags with some `KDB_O_*` option bits - as explained above |

#### Returns

pointer to the Key found

#### Return values

| | |
|---|---|
| *0* | if no Key has been found |
| *0* | on NULL pointers |

#### Since

1.0.0

#### See also

ksLookupByName() to search by a name given by a string

ksGetSize(), ksAtCursor() for iterating over a KeySet

### 570.6.7.5 ksLookupByName()

```
Key* ksLookupByName (
            KeySet * ks,
```

```
            const char * name,
            elektraLookupFlags options )
```

Convenience method to look for a Key contained in `ks` with name `name`.
There are several options that can be used in conjunction with this function. All possible option flags can be found in elektraLookupFlags

**Parameters**

| | |
|---|---|
| *ks* | the KeySet that should be searched |
| *name* | name of the Key you are looking for |
| *options* | some `KDB_O_*` option bits (KDB_O_POP, KDB_O_DEL):<br>• See ksLookup() or elektraLookupFlags for possible options |

**Returns**

pointer to the Key found

**Return values**

| | |
|---|---|
| *0* | if no Key has been found |
| *0* | on NULL pointers |

**Since**

1.0.0

**See also**

ksLookup() for explanation of the functionality and examples.

ksGetSize(), ksAtCursor() for iterating over a KeySet

### 570.6.7.6 ksNew()

```
KeySet* ksNew (
            size_t alloc,
             ...  )
```

Allocate, initialize and return a new KeySet object.
Objects created with ksNew() must be destroyed with ksDel().
You can use an arbitrary long list of parameters to preload the KeySet with a list of Keys. Either your first and only parameter is 0 or your last parameter must be KS_END.
So, terminate with ksNew(0, KS_END) or ksNew(20, ..., KS_END)

**Warning**

Never use ksNew(0, keyNew(...), KS_END). If the first parameter is 0, other arguments are ignored.

The first parameter `alloc` defines how many Keys can be added without reallocation. If you pass any alloc size greater than 0, but less than 16, it will default to 16.
For most uses

```
KeySet * keys = ksNew (1, KS_END);
// enough memory for up to 16 keys, without needing reallocation
ksDel (keys);
```

will be fine. The alloc size will be 16 and will double whenever size reaches alloc size, so it also performs well with large KeySets.
You can defer the allocation of the internal array that holds the Keys, by passing 0 as the alloc size. This is useful if it is unclear whether your KeySet will actually hold any Keys and you want to avoid a malloc call.

```
// Create KeySet without allocating memory for keys
KeySet * keys = ksNew (0, KS_END);
// The first allocation will happen in ksAppendKey
ksAppendKey(keys, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END));
// work with the KeySet
ksDel (keys);
```

If the size of the KeySet is known in advance, use the `alloc` parameter to hint the size of the KeySet.
If your application only needs up to 15 Keys you can request a KeySet of size 15:

```
KeySet * keys = ksNew (15, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key01", KEY_VALUE,
       "value01", KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key03", KEY_VALUE, "value03",
       KEY_END),
                       // ...
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key15", KEY_VALUE, "value15",
       KEY_END), KS_END);
// work with it
ksDel (keys);
```

If you start having 3 Keys, and your application needs approximately 200 up to 500 Keys, you can use:

```
KeySet * config = ksNew (500, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key1", KEY_VALUE,
       "value1", KEY_END),
                        keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key2", KEY_VALUE, "value2",
       KEY_END),
                        keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key3", KEY_VALUE, "value3",
       KEY_END),
                        KS_END); // don't forget the KS_END at the end!
// work with it
ksDel (config);
```

Alloc size is 500, the size of the KeySet will be 3 after ksNew. This means the KeySet will reallocate when appending more than 497 keys.
The main benefit of taking a list of variant length parameters is to be able to have one C-Statement for any possible KeySet. If you prefer, you can always create an empty KeySet and use ksAppendKey().

**Postcondition**

the KeySet is rewinded properly

**Parameters**

| | |
|---|---|
| *alloc* | gives a hint for how many Keys may be stored initially |

**Returns**

a ready to use KeySet object

**Return values**

| | |
|---|---|
| *0* | on memory error |

**Since**

1.0.0

**See also**

ksDel() to free the KeySet afterwards

ksDup() to duplicate an existing KeySet

ksAppendKey() to append individual Keys to a KeySet

### 570.6.7.7 ksNext()

```
Key* ksNext (
            KeySet * ks )
```

Returns the next Key in a KeySet.
KeySets have an internal cursor that can be reset with ksRewind(). Every time ksNext() is called, the cursor is incremented and the new current Key is returned.
You'll get a NULL pointer if the Key at the end of the KeySet has been reached. On subsequent calls of ksNext() it will still return the NULL pointer.
The `ks` internal cursor will be changed, so it is not const.

**Note**

You must not delete or change the Key, use ksPop() if you want to delete it.

That applications must do ksLookup() with an cascading Key for every single Key before using it, because specifications allow to hide or override Keys.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object to work with |

**Returns**

the new current Key

**Return values**

| | |
|---|---|
| *0* | when the end of the KeySet has been reached |
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

ksRewind() for resetting the internal cursor of the KeySet

ksCurrent() for getting the Key the cursor currently points at

ksLookup() to honor specifications

### 570.6.7.8 ksPop()

```
Key* ksPop (
              KeySet * ks )
```

Remove and return the last Key of `ks`.
The reference counter of the Key will be decremented by one.
The KeySet's cursor will not be affected if it did not point to the popped Key.

**Note**

You need to keyDel() the Key afterwards, if you don't append it to another KeySet. It has the same semantics like a Key allocated with keyNew() or keyDup().

```
ks1=ksNew(0, KS_END);
ks2=ksNew(0, KS_END);
k1=keyNew("user:/name", KEY_END); // ref counter 0
ksAppendKey(ks1, k1); // ref counter 1
ksAppendKey(ks2, k1); // ref counter 2
k1=ksPop (ks1); // ref counter 1
k1=ksPop (ks2); // ref counter 0, like after keyNew()
ksAppendKey(ks1, k1); // ref counter 1
ksDel (ks1); // key is deleted too
ksDel (ks2);
```

**Parameters**

| | |
|---|---|
| *ks* | KeySet to pop a Key from |

**Returns**

the last Key of `ks`

**Return values**

| | |
|---|---|
| *NULL* | if `ks` is empty or a NULL pointer |

**Since**

1.0.0

**See also**

ksLookup() to pop Keys by name

ksCopy() to pop all Keys

### 570.6.7.9 ksRename()

```
ssize_t ksRename (
              KeySet * ks,
              const Key * root,
              const Key * newRoot )
```

Moves all keys below `root` to below `newRoot`.
Only keys below `root` will be modified. The rest of `ks` remains untouched.
This functions is similar to the following snippet, but there are some differences.

```
KeySet * toRename = ksCut (ks, root);
for (elektraCursor cursor = 0; cursor < ksGetSize (toRename); cursor++)
{
    Key * cur = keyDup (ksAtCursor (ks, cursor));
    keyReplacePrefix (cur, root, newRoot);
    ksAppendKey (ks, cur);
}
ksDel (toRename);
```

Firstly, the optimizations only work, if `ks` doesn't contain any keys below `newRoot` that aren't below `root`. If such keys exist, ksRename() will still work, but it will fall back to code similar to the for-loop above.
The second difference is that ksRename() will modify the keys in `ks` directly, if they aren't referenced from anywhere else (if their reference count is 1 (see keyGetRef())). Normally, this shouldn't cause problems, but if you have a direct `Key *` pointer to a key in `ks` or hold a reference to some data within a key of `ks`, you may need to call keyIncRef() to ensure the key isn't modified.

**Parameters**

| | |
|---|---|
| *ks* | the keyset to manipulate |
| *root* | the old prefix that will be removed, must not be a cascading key |
| *newRoot* | the new prefix the will replace the old one, must not be a cascading key |

**Return values**

| | |
|---|---|
| *-1* | if any of `ks`, `root`, `newRoot` is NULL, or if `root` or `newRoot` are cascading keys |
| *-2* | if `ks` already contains keys below `newRoot` |
| *0* | if `ks` contains no keys below `root` (and also not `root` itself) |

#### Returns

otherwise, the number of keys that have been renamed

### 570.6.7.10 ksRewind()

```
int ksRewind (
            KeySet * ks )
```

Rewinds the KeySet internal cursor.
Use it to set the cursor to the beginning of the KeySet. ksCurrent() will always return NULL afterwards. So you want to use ksNext() first.

```
ksRewind (ks);
while ((key = ksNext (ks))!=0) {}
```

#### Parameters

| | |
|---|---|
| *ks* | the KeySet that should be rewound |

#### Return values

| | |
|---|---|
| *0* | on success |
| *-1* | on NULL pointer |

#### Since

1.0.0

#### See also

ksNext() for moving the cursor to the next entry in the KeySet

ksCurrent() for getting the current element in the KeySet

### 570.6.7.11 ksSearch()

```
ssize_t ksSearch (
            const KeySet * ks,
            const Key * key )
```

Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.

```
ssize_t result = ksSearch(ks, key);
if (result >= 0)
{
        ssize_t position = result;
        // The key already exist in key set.
} else {
        ssize_t insertpos = -result-1;
        // The key was not found in key set.
}
```

#### Parameters

| | |
|---|---|
| *ks* | the keyset to work with |
| *key* | the key to check |

#### Returns

position where the key is (>=0) if the key was found

-insertpos -1 (< 0) if the key was not found so to get the insertpos simple do: -insertpos -1

#### See also

ksLookup() for retrieving the found Key

### 570.6.7.12 ksSetCursor()

```
int ksSetCursor (
            KeySet * ks,
            elektraCursor cursor )
```

Set the KeySet internal cursor to `cursor`.
Use it to set the cursor to a stored position. ksCurrent() will then return the Key at the position of the supplied cursor.

#### Warning

Cursors may get invalid when the Key was ksPop()ed or ksLookup() was used together with KDB_O_POP.

```
elektraCursor cursor;
..
// key now in any position here
cursor = ksGetCursor (ks);
while ((key = keyNextMeta (ks))!=0) {}
ksSetCursor (ks, cursor); // reset state
ksCurrent(ks); // in same position as before
```

An invalid cursor will set the KeySet to its beginning like ksRewind(). When you set an invalid cursor ksCurrent() is 0.

#### Parameters

| | |
|---|---|
| *ks* | the KeySet object where the cursor should be set |
| *cursor* | the cursor to set for `ks` |

#### Return values

| | |
|---|---|
| *0* | when the KeySet has been ksRewind()ed |
| *1* | otherwise |
| *-1* | on NULL pointer |

#### Since

1.0.0

#### See also

ksGetCursor() for getting the cursor at the current position

ksNext() for moving the internal cursor forward

ksCurrent() for getting the Key at the current position

### 570.6.7.13 ksSubtract()

```
ssize_t ksSubtract (
            KeySet * total,
            const KeySet * sub )
```

Remove all the keys in `sub` from `total`.

**Parameters**

| | |
|---|---|
| *total* | the keyset where the keys should be removed from |
| *sub* | the keys to remove |

**Return values**

| | |
|---|---|
| *-1* | on `NULL` pointers |
| *the* | number of keys removed from `total` |

#### 570.6.7.14 ksVNew()

```
KeySet* ksVNew (
              size_t alloc,
              va_list va )
```

Allocate, initialize and return a new KeySet object.
Objects created with ksNew() must be destroyed with ksDel().
You can use an arbitrary long list of parameters to preload the KeySet with a list of Keys. Either your first and only parameter is 0 or your last parameter must be KS_END.
So, terminate with ksNew(0, KS_END) or ksNew(20, ..., KS_END)

**Warning**

Never use ksNew(0, keyNew(...), KS_END). If the first parameter is 0, other arguments are ignored.

The first parameter `alloc` defines how many Keys can be added without reallocation. If you pass any alloc size greater than 0, but less than 16, it will default to 16.
For most uses

```
KeySet * keys = ksNew (1, KS_END);
// enough memory for up to 16 keys, without needing reallocation
ksDel (keys);
```

will be fine. The alloc size will be 16 and will double whenever size reaches alloc size, so it also performs well with large KeySets.
You can defer the allocation of the internal array that holds the Keys, by passing 0 as the alloc size. This is useful if it is unclear whether your KeySet will actually hold any Keys and you want to avoid a malloc call.

```
// Create KeySet without allocating memory for keys
KeySet * keys = ksNew (0, KS_END);
// The first allocation will happen in ksAppendKey
ksAppendKey(keys, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END));
// work with the KeySet
ksDel (keys);
```

If the size of the KeySet is known in advance, use the `alloc` parameter to hint the size of the KeySet.
If your application only needs up to 15 Keys you can request a KeySet of size 15:

```
KeySet * keys = ksNew (15, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key01", KEY_VALUE,
       "value01", KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key03", KEY_VALUE, "value03",
       KEY_END),
                       // ...
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key15", KEY_VALUE, "value15",
       KEY_END), KS_END);
// work with it
ksDel (keys);
```

If you start having 3 Keys, and your application needs approximately 200 up to 500 Keys, you can use:

```
KeySet * config = ksNew (500, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key1", KEY_VALUE,
       "value1", KEY_END),
                         keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key2", KEY_VALUE, "value2",
       KEY_END),
                         keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key3", KEY_VALUE, "value3",
       KEY_END),
                         KS_END); // don't forget the KS_END at the end!
// work with it
ksDel (config);
```

Alloc size is 500, the size of the KeySet will be 3 after ksNew. This means the KeySet will reallocate when appending more than 497 keys.
The main benefit of taking a list of variant length parameters is to be able to have one C-Statement for any possible KeySet. If you prefer, you can always create an empty KeySet and use ksAppendKey().

**Postcondition**

the KeySet is rewinded properly

**Parameters**

| | |
|---|---|
| *alloc* | gives a hint for how many Keys may be stored initially |

**Returns**

a ready to use KeySet object

**Return values**

| | |
|---|---|
| *0* | on memory error |

**Since**

1.0.0

**See also**

ksDel() to free the KeySet afterwards

ksDup() to duplicate an existing KeySet

ksAppendKey() to append individual Keys to a KeySet

**Precondition**

caller must call va_start and va_end

**va the list of arguments**

**Parameters**

| | |
|---|---|
| *alloc* | the allocation size |
| *va* | the list of variable arguments |

## 570.7 Meta Data proposal+compatibility

Meta data proposal+compatibility methods.

### Functions

- const char * keyComment (const Key *key)

  *Return a pointer to the real internal `key` comment.*
- ssize_t keyGetCommentSize (const Key *key)

*Calculates number of bytes needed to store a key comment, including final NULL.*

- ssize_t keyGetComment (const Key ∗key, char ∗returnedComment, size_t maxSize)

  *Get the key comment.*
- ssize_t keySetComment (Key ∗key, const char ∗newComment)

  *Set a comment for a key.*
- int elektraKeyCmpOrder (const Key ∗ka, const Key ∗kb)

  *Compare the order metadata of two keys.*
- void elektraMetaArrayAdd (Key ∗key, const char ∗metaName, const char ∗value)

  *creates an metadata array or appends another element to an existing metadata array e.g.*
- KeySet ∗ elektraMetaArrayToKS (Key ∗key, const char ∗metaName)

  *Create a `KeySet` from a metakey array.*
- int elektraSortTopology (KeySet ∗ks, Key ∗∗array)

  *topological sorting*
- char ∗ elektraMetaArrayToString (const Key ∗key, const char ∗metaName, const char ∗delim)

  *returns the metakey array as a string separated by delim*

## 570.7.1 Detailed Description

Meta data proposal+compatibility methods.
In versions before Elektra 0.8 only limited metadata was available. Now any metadata can be added. These API methods are implementations of the 0.7 API using 0.8 metadata.
Additionally, new suggestions can be made here.
It is planned that these methods will be generated from doc/METADATA.ini and moved to a separate library. Currently, you should better avoid the methods and directly use metainfo instead.

## 570.7.2 Function Documentation

### 570.7.2.1 elektraKeyCmpOrder()

```
int elektraKeyCmpOrder (
            const Key * ka,
            const Key * kb )
```

Compare the order metadata of two keys.

**Returns**

a number less than, equal to or greater than zero if the order of k1 is found, respectively, to be less than, to match, or be greater than the order of k2. If one key is NULL, but the other isn't, the key which is not NULL is considered to be greater. If both keys are NULL, they are considered to be equal. If one key does have an order metadata but the other has not, the key with the metadata is considered greater. If no key has metadata, they are considered to be equal.

**Parameters**

| | |
|---|---|
| *ka* | key to compare with |
| *kb* | other key to compare with |

### 570.7.2.2 elektraMetaArrayAdd()

```
void elektraMetaArrayAdd (
            Key * key,
            const char * metaName,
```

```
            const char * value )
```

creates an metadata array or appends another element to an existing metadata array e.g.

```
Key *key = keyNew("user:/test", KEY_END);
elektraMetaArrayAdd(key, "test", "val0");
// key now has "test/#0" with value "val0" as metadata
elektraMetaArrayAdd(key, "test", "val1");
// appends "test/#1" with value "val1" to key
```

**Parameters**

| | |
|---|---|
| *key* | the key the metadata should be added to |
| *metaName* | the name of the metakey array parent |
| *value* | the value of the newly appended metakey |

#### 570.7.2.3 elektraMetaArrayToKS()

```
KeySet* elektraMetaArrayToKS (
            Key * key,
            const char * metaName )
```

Create a `KeySet` from a metakey array.

For example, the following function call

```
elektraMetaArrayToKS(
        keyNew ("/a", KEY_VALUE, "b, c",
                KEY_META, "dep",    "#1",
                KEY_META, "dep/#0", "/b",
                KEY_META, "dep/#1", "/c", KEY_END),
        "dep");
```

returns a `KeySet` containing the keys `dep` with value `#1`, `"dep/#0"` with value `"/b"` and `"dep/#1"` with value `"/c"`.

If no metakey array is found, null is returned. The returned `KeySet` must be freed with `ksDel`

**Returns**

a keyset containing all the metakeys of the metakey array or null if no metakey array is found

**Parameters**

| | |
|---|---|
| *key* | the key containing the metakey array |
| *metaName* | the name of the metakey array parent |

#### 570.7.2.4 elektraMetaArrayToString()

```
char* elektraMetaArrayToString (
            const Key * key,
            const char * metaName,
            const char * delim )
```

returns the metakey array as a string separated by delim

**Parameters**

| | |
|---|---|
| *key* | the key containing the metakey array |
| *metaName* | the name of the metakey array parent |
| *delim* | delimiter for the records in the returned string |

**Returns**

a string containing all metakey values separated by "delim"

#### 570.7.2.5 elektraSortTopology()

```
int elektraSortTopology (
            KeySet * ks,
            Key ** array )
```

topological sorting

**Parameters**

| | |
|---|---|
| *array* | the array where the sorted keys will be stored in topological order. Nothing will be written into an array if |
| *ks* | is the keyset that should be sorted. Dependencies and order is defined by metakeys. |

- the "dep/#" metakeys e.g. the Key *k = keyNew ("/a", KEY_VALUE, "b, c", KEY_META, "dep", "#1", KEY_META, "dep/#0", "/b", KEY_META, "dep/#1", "/c", KEY_END), "dep"); depends on Key "/b" and Key "/c".
- if "order" metakeys are defined for the keys the algorithm tries to resolves them by that order using lexical comparison. You should prefer `#0` array syntax.

Duplicated and reflexive dep entries are ignored.
The algorithm used is a mixture of Kahn and BFS. Furthermore the algorithm does not use recursion.
First a BFS with the keys sorted by "order" is used. Then all dependencies (recursively) of every key is collected.

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | for cycles |
| *-1* | for invalid dependencies |

#### 570.7.2.6 keyComment()

```
const char* keyComment (
            const Key * key )
```

Return a pointer to the real internal `key` comment.
This is a much more efficient version of keyGetComment() and you should use it if you are responsible enough to not mess up things. You are not allowed to change anything in the memory region the returned pointer points to.
keyComment() returns "" when there is no keyComment. The reason is

```
key=keyNew(0);
keySetComment(key,"");
keyComment(key); // you would expect "" here
keyDel(key);
```

See keySetComment() for more information on comments.

**Note**

Note that the Key structure keeps its own size field that is calculated by library internal calls, so to avoid inconsistencies, you must never use the pointer returned by keyComment() method to set a new value. Use keySetComment() instead.

**Parameters**

| | |
|---|---|
| *key* | the key object to work with |

**Returns**

a pointer to the internal managed comment

**Return values**

| | |
|---|---|
| *""* | when there is no comment |
| *0* | on NULL pointer |

**See also**

keyGetCommentSize() for size and keyGetComment() as alternative

#### 570.7.2.7 keyGetComment()

```
ssize_t keyGetComment (
            const Key * key,
            char * returnedComment,
            size_t maxSize )
```

Get the key comment.

### 570.7.3 Comments

A Key comment is description for humans what this key is for. It may be a textual explanation of valid values, when and why a user or administrator changed the key or any other text that helps the user or administrator related to that key.
Don't depend on a comment in your program. A user is always allowed to remove or change it in any way they want to. But you are allowed or even encouraged to always show the content of the comment to the user and allow him to change it.

**Parameters**

| | |
|---|---|
| *key* | the key object to work with |
| *returnedComment* | pre-allocated memory to copy the comments to |
| *maxSize* | number of bytes that will fit returnedComment |

**Returns**

the number of bytes actually copied to `returnedString`, including final NULL

**Return values**

| | |
|---|---|
| *1* | if the string is empty |
| *-1* | on NULL pointer |
| *-1* | if maxSize is 0, not enough to store the comment or when larger then SSIZE_MAX |

**See also**

keyGetCommentSize(), keySetComment()

#### 570.7.3.1 keyGetCommentSize()

```
ssize_t keyGetCommentSize (
```

```
            const Key * key )
```

Calculates number of bytes needed to store a key comment, including final NULL.
Use this method to know to size for allocated memory to retrieve a key comment.
See keySetComment() for more information on comments.
For an empty key name you need one byte to store the ending NULL. For that reason 1 is returned.

```
 char *buffer;
 buffer = elektraMalloc (keyGetCommentSize (key));
// use this buffer to store the comment
// pass keyGetCommentSize (key) for maxSize
```

**Parameters**

| | |
|---|---|
| *key* | the key object to work with |

**Returns**

number of bytes needed

**Return values**

| | |
|---|---|
| *1* | if there is no comment |
| *-1* | on NULL pointer |

**See also**

keyGetComment(), keySetComment()

### 570.7.3.2 keySetComment()

```
ssize_t keySetComment (
            Key * key,
            const char * newComment )
```

Set a comment for a key.
A key comment is like a configuration file comment. See keySetComment() for more information.

**Parameters**

| | |
|---|---|
| *key* | the key object to work with |
| *newComment* | the comment, that can be freed after this call. |

**Returns**

the number of bytes actually saved including final NULL

**Return values**

| | |
|---|---|
| *0* | when the comment was freed (newComment NULL or empty string) |
| *-1* | on NULL pointer or memory problems |

**See also**

keyGetComment()

## 570.8 Meta Info Manipulation Methods

Methods to do various operations on Key metadata.
Collaboration diagram for Meta Info Manipulation Methods:


### Functions

- const Key * keyNextMeta (Key *key)

  *Get the next metadata entry of a Key.*
- int keyCopyMeta (Key *dest, const Key *source, const char *metaName)

  *Do a shallow copy of metadata with name `metaName` from source to dest.*
- int keyCopyAllMeta (Key *dest, const Key *source)

  *Do a shallow copy of all metadata from source to dest.*
- const Key * keyGetMeta (const Key *key, const char *metaName)

  *Returns the Key for a metadata entry with name `metaName`.*
- ssize_t keySetMeta (Key *key, const char *metaName, const char *newMetaString)

  *Set a new metadata Key.*
- KeySet * keyMeta (Key *key)

  *Returns the KeySet holding the given Key's metadata.*

### 570.8.1 Detailed Description

Methods to do various operations on Key metadata.
To use them:

```
#include <kdb.h>
```

Next to Name (key and owner) and value (data and comment) there is the so called meta information inside every key.
Key meta information are an unlimited number of key/value pairs strongly related to a key. It main purpose is to give keys special semantics, so that plugins can treat them differently.
Metakey, as opposed to Key, deliberately has following limitations:

- no null values
- no binary data
- no modification of references (COW)
- no guarantee of ordering

#### 570.8.1.1 Examples for metadata

File system information (see stat(2) for more information):

- uid: the user id (positive number)
- gid: the group id (positive number)
- mode: filesystem-like mode permissions (positive octal number)
- atime: When was the key accessed the last time.
- mtime: When was the key modified the last time.
- ctime: When the uid, gid or mode of a key changes. (times are represented through a positive number as unix timestamp)

The comment can contain userdata which directly belong to that key. The name of the meta information is "comment" for a general purpose comment about the key. Multi-Language comments are also supported by appending [LANG] to the name.
Validators are regular expressions which are tested against the key value. The metakey "validator" can hold a regular expression which will be matched against.
Types can be expressed with the meta information "type".
The relevance of the key can be tagged with a value from -20 to 20. Negative numbers are the more important and must be present in order to start the program.
A version of a key may be stored with "version". Its format is full.major.minor where all of these are integers.
The order inside a persistent storage can be described with the tag "order" which contains a positive number.
The metakey "app" describes to which application a key belongs. It can be used to remove keys from an application no longer installed.
The metakey "path" describes where the key is physically stored.
The "owner" is the user that owns the key. It only works for the user:/ hierarchy. It rather says where the key is stored and says nothing about the filesystem properties.

### 570.8.2 Function Documentation

#### 570.8.2.1 keyCopyAllMeta()

```
int keyCopyAllMeta (
              Key * dest,
              const Key * source )
```

Do a shallow copy of all metadata from source to dest.
The key dest will additionally have all metadata the source had. Metadata not present in source will not be changed. Metadata which was present in source and dest will be overwritten. If the `dest` Key is read-only it will not be changed.
For example the metadata type is copied into the Key k:

```
void l (Key * k)
{
        // receive copy
        keyCopyAllMeta (k, copy);
        // the caller will see the changed key k
        // with all the metadata from copy
}
```

The main purpose of this function is for plugins or applications which want to add the same metadata to n keys. When you do that with keySetMeta() it will take n times the memory for the key. This can be considerable amount of memory for many keys with some metadata for each.
To avoid that problem you can use keyCopyAllMeta() or keyCopyMeta():

```
void o (KeySet * ks)
{
        Key * current;
        Key * shared = keyNew ("/", KEY_END);
        keySetMeta (shared, "shared1", "this metadata should be shared among many keys");
        keySetMeta (shared, "shared2", "this metadata should be shared among many keys also");
        keySetMeta (shared, "shared3", "this metadata should be shared among many keys too");
        for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
        {
```

```
                current = ksAtCursor (ks, it);
                if (needsSharedData (current)) keyCopyAllMeta (current, shared);
        }
        keyDel (shared);
}
```

**Precondition**

`dest's` metadata is not read-only

**Postcondition**

for every metaName present in source: keyGetMeta(source, metaName) == keyGetMeta(dest, metaName)

**Parameters**

| | |
|---|---|
| *dest* | the destination where the metadata should be copied too |
| *source* | the key where the metadata should be copied from |

**Return values**

| | |
|---|---|
| *1* | if metadata was successfully copied |
| *0* | if source did not have any metadata |
| *-1* | on null pointer of dest or source |
| *-1* | on memory problems |

**Since**

1.0.0

**See also**

keyCopyMeta() for copying one metadata Key from `dest` to `source`

### 570.8.2.2 keyCopyMeta()

```
int keyCopyMeta (
            Key * dest,
            const Key * source,
            const char * metaName )
```

Do a shallow copy of metadata with name `metaName` from source to dest.
Afterwards `source` and `dest` will have the same metadata referred with `metaName`. If the Key with name `metaName` doesn't exist in `source` - it gets deleted in `dest`.
For example the metadata type is copied into the Key k.

```
void l(Key *k)
{
        // receive c
        keyCopyMeta(k, c, "type");
        // the caller will see the changed key k
        // with the metadata "type" from c
}
```

The main purpose of this function is for plugins or applications, which want to add the same metadata to n keys. When you do that keySetMeta() will take n times the memory for the key. This can be a considerable amount of memory for many keys with some metadata for each.
To avoid that problem you can use keyCopyAllMeta() or keyCopyMeta().

```
void o(KeySet *ks)
{
        Key *shared = keyNew ("/", KEY_END);
        keySetMeta(shared, "shared", "this metadata should be shared among many keys");
        for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
        {
```

```
                Key * current = ksAtCursor (ks, it);
                if (needs_shared_data(current)) keyCopyMeta(current, shared, "shared");
        }
}
```

**Precondition**

`dest's` metadata is not read-only

**Postcondition**

keyGetMeta(source, metaName) == keyGetMeta(dest, metaName)

**Parameters**

| | |
|---|---|
| *dest* | the destination where the metadata should be copied to |
| *source* | the key where the metadata should be copied from |
| *metaName* | the name of the metadata Key which should be copied |

**Return values**

| | |
|---|---|
| *1* | if was successfully copied |
| *0* | if the metadata in dest was removed too |
| *-1* | on null pointers (source or dest) |
| *-1* | on memory problems |
| *-1* | if metadata is read-only |

**Since**

1.0.0

**See also**

keyCopyAllMeta() copies all metadata from `dest` to `src`

### 570.8.2.3 keyGetMeta()

```
const Key* keyGetMeta (
             const Key * key,
             const char * metaName )
```

Returns the Key for a metadata entry with name `metaName`.
You are not allowed to modify the resulting key.
If `metaName` does not start with 'meta:/', it will be prefixed with 'meta:/'.

```
Key metaData = keyGetMeta(k, "type")
// keyType == "boolean"
char keyType[] = keyValue(metaData)
```

**Note**

You must not delete or change the returned key, use keySetMeta() if you want to delete or change it.

**Precondition**

`key` contains metadata

`metaName` is prefixed with "meta:/"

**Parameters**

| | |
|---|---|
| *key* | the Key from which to get metadata |
| *metaName* | the name of the meta information you want the Key from. |

**Returns**

value of meta-information if meta-information is found

**Return values**

| | |
|---|---|
| *0* | if key or metaName is NULL |
| *0* | if no such metaName is found |

**Since**

1.0.0

**See also**

keySetMeta() for setting metadata

keyMeta() for getting the KeySet containing metadata

### 570.8.2.4 keyMeta()

```
KeySet* keyMeta (
            Key * key )
```

Returns the KeySet holding the given Key's metadata.

Use keySetMeta() to populate the metadata KeySet of a Key.

```
Key * key = keyNew ("user:/test/key", KEY_END);
keySetMeta (key, "meta1", "value1");
keySetMeta (key, "meta2", "value2");
```

Iterate the returned metadata KeySet like any other KeySet.

```
Key * cur;
KeySet * metaKeys;
ssize_t ksSize;
metaKeys = keyMeta (key);
ksSize = ksGetSize (metaKeys);
for (elektraCursor it = 0; it < ksSize; ++it)
{
        cur = ksAtCursor (metaKeys, it);
        printf ("meta name: %s, meta value: %s\n", keyName (cur), keyString (cur));
}
```

Use ksLookup() or keyGetMeta() to retrieve a single value for a given Key.

```
Key * lookupKey = ksLookupByName (keyMeta (key), "meta2", 0);
printf ("meta name: %s, meta value: %s\n", keyName (lookupKey), keyString (lookupKey));
keyDel (key);
```

**Note**

You are not allowed to modify the name of KeySet's Keys or delete them.

You must not delete the returned KeySet.

Adding a key with metadata to the KeySet is an error.

If the key has not metadata keyset, it will be allocated

**Postcondition**

for the returned KeySet ks: keyGetMeta(key, metaName) == ksLookupByName(ks, metaName)

`key` contains a KeySet for the metadata

**Parameters**

| | |
|---|---|
| *key* | the Key from which to get the metadata KeySet |

**Returns**

the KeySet holding the metadata

**Return values**

| | |
|---|---|
| *0* | if the Key is 0 |

**Since**

1.0.0

**See also**

keySetMeta() for setting a metadata Key

keyGetMeta() for getting a metadata Key

#### 570.8.2.5 keyNextMeta()

```
const Key* keyNextMeta (
            Key * key )
```

Get the next metadata entry of a Key.
Keys have an internal cursor. Every time keyNextMeta() is called the cursor is incremented and the new current Name of Meta Information is returned.
You'll get a NULL pointer if the metadata after the end of the Key was reached. On subsequent calls of keyNextMeta() it will still return the NULL pointer.
The `key` internal cursor will be changed, so it is not const.

**Note**

That the resulting key is guaranteed to have a value, because meta information has no binary or null pointer semantics.

You must not delete or change the returned key, use keySetMeta() if you want to delete or change it.

**Parameters**

| | |
|---|---|
| *key* | the Key object to work with |

**Returns**

a key containing metadata

**Return values**

| | |
|---|---|
| *0* | when the last Key has been reached |
| *0* | when Key is a NULL pointer |

**Since**

1.0.0

#### 570.8.2.6 keySetMeta()

```
ssize_t keySetMeta (
            Key * key,
            const char * metaName,
            const char * newMetaString )
```

Set a new metadata Key.
Will set a new metadata pair with name `metaName` and value `newMetaString`.
Will add a new metadata Key, if `metaName` was unused until now.
It will modify an existing Pair of metadata if `metaName` was already present.
It will remove a metadata Key if `newMetaString` is 0.
If `metaName` does not start with 'meta:/', it will be prefixed with 'meta:/'.

**Precondition**

`metaName` is prefixed with "meta:/"

`key's` metadata is not read-only

**Postcondition**

The value in `key's` metadata Keyset for `metaName` is `newMetaString`

**Parameters**

| | |
|---|---|
| *key* | Key whose metadata should be set |
| *metaName* | name of the metadata Key that should be set |
| *newMetaString* | new value for the metadata Key |

**Returns**

size (>0) of `newMetaString` if metadata has been successfully added

**Return values**

| | |
|---|---|
| *0* | if the meta-information for metaName was removed |
| *-1* | if key or metaName is 0 |
| *-1* | if system is out of memory |
| *-1* | if `metaName` is not a valid metadata name |

**Since**

1.0.0

**See also**

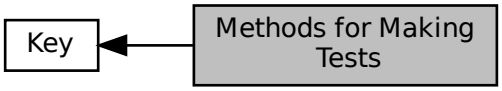
keyGetMeta() for getting the value of a metadata Key

keyMeta() for getting the KeySet containing metadata

## 570.9 Methods for Making Tests

Methods to do various tests on Keys.

Collaboration diagram for Methods for Making Tests:




## Functions

- int keyCmp (const Key ∗k1, const Key ∗k2)

  *Compare the name of two Keys.*
- int keyIsBelow (const Key ∗key, const Key ∗check)

  *Check if the Key `check` is below the Key `key` or not.*
- int keyIsDirectlyBelow (const Key ∗key, const Key ∗check)

  *Check whether the Key `check` is directly below the Key `key`.*
- int keyIsBinary (const Key ∗key)

  *Check if the value of a `key` is of binary type.*
- int keyIsString (const Key ∗key)

  *Check if the value of `key` is of string type.*

### 570.9.1 Detailed Description

Methods to do various tests on Keys.
With exception of the parameters of keyCmp(), the following contract holds for all parameters of type Key:

**Precondition**

The Key has been properly initialized via keyNew()

**Invariant**

All parts of the Key remain unchanged

**Postcondition**

All parts of the Key are unchanged

To use them:
```
#include <kdb.h>
```

### 570.9.2 Function Documentation

#### 570.9.2.1 keyCmp()

```
int keyCmp (
            const Key * k1,
            const Key * k2 )
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

**Note**

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

**Precondition**

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

**Invariant**

All parts of the Keys remain unchanged

**Postcondition**

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

**Parameters**

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

**Return values**

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

**Since**

1.0.0

**See also**

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

### 570.9.2.2 keyIsBelow()

```
int keyIsBelow (
            const Key * key,
            const Key * check )
```

Check if the Key `check` is below the Key `key` or not.
Example:

```
key user:/sw/app
check user:/sw/app/key
```

returns true because `check` is below `key`
Example:

```
key user:/sw/app
check user:/sw/app/folder/key
```

returns also true because `check` is indirectly below `key`
Obviously, there is no Key above a namespace (e.g. user, system, /):

```
key *
check user
```

**Parameters**

| | |
|---|---|
| *key* | the Key object to check against |
| *check* | the Key object for which it should be checked whether it is below `key` |

**Return values**

| | |
|---|---|
| *1* | if `check` is below `key` |
| *0* | if it is not below or if it is the same key |
| *-1* | if key or check is null |

**Since**

1.0.0

**See also**

keyIsDirectlyBelow() for checking whether a Key is directly below another

keyGetName(), keySetName() for getting / setting the Key's name

### 570.9.2.3 keyIsBinary()

```
int keyIsBinary (
            const Key * key )
```

Check if the value of a `key` is of binary type.
The function checks if the value of `key` is binary. Contrary to string values binary values can have '\0' inside the value and may not be terminated by a null character. Their disadvantage is that you need to pass their size.
Make sure to use this function and don't test the binary type another way to ensure compatibility and to write less error prone programs.

**Parameters**

| | |
|---|---|
| *key* | the Key to check |

**Return values**

| | |
|---|---|
| *1* | if the value of `key` is binary |
| *0* | if the value of `key` is not binary |

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |

**See also**

keyGetBinary(), keySetBinary() for getting / setting a Key's value as binary

### 570.9.2.4 keyIsDirectlyBelow()

```
int keyIsDirectlyBelow (
            const Key * key,
            const Key * check )
```

Check whether the Key `check` is directly below the Key `key`.

```
Example:
key user:/sw/app
check user:/sw/app/key
```

returns true because check is directly below key

```
Example:
key user:/sw/app
check user:/sw/app/folder/key
```

does not return true, because it is only indirectly below

**Parameters**

| | |
|---|---|
| *key* | the Key object to check against |
| *check* | the Key object for which it should be checked whether it is directly below `key` |

**Return values**

| | |
|---|---|
| *1* | if `check` is directly below `key` |
| *0* | if `check` is not directly below `key` or if it is the same |
| *-1* | on null pointer |

**Since**

1.0.0

**See also**

keyIsBelow() for checking whether a Key is below another

keyGetName(), keySetName() for getting / setting the Key's name

### 570.9.2.5 keyIsString()

```
int keyIsString (
            const Key * key )
```

Check if the value of `key` is of string type.
String values are null terminated and are not allowed to have any '\0' characters inside the string.
Make sure to use this function and don't test the string type another way to ensure compatibility and to write less error prone programs.

**Parameters**

| | |
|---|---|
| *key* | the Key to check |

**Return values**

| | |
|---|---|
| *1* | if the value of `key` is string |
| *0* | if the value of `key` is not string |
| *-1* | on NULL pointer |

**See also**

keyGetString(), keySetString() for getting / setting a Key's value as string

## 570.10 Modules

Loading Modules for Elektra.

### Functions

- int elektraModulesInit (KeySet *modules, Key *error)
  *Initialises the module loading system.*
- elektraPluginFactory elektraModulesLoad (KeySet *modules, const char *name, Key *error)
  *Load a library with the given name.*
- int elektraModulesClose (KeySet *modules, Key *error)
  *Close all modules.*

### 570.10.1 Detailed Description

Loading Modules for Elektra.
Unfortunately there is no portable way to load modules, plugins or libraries. So Elektra needed a framework which abstracts the loading of modules. Depending of the operating system the build system chooses different source files which actually implement the loading of a module.
The goals are:

- to have a list of all loaded modules
- writing module loaders should be easy
- handle and report errors well
- avoid loading of modules multiple times (maybe the OS can't handle that well)
- hide the OS dependent handle inside a Key (handle is needed to close module afterwards)

### 570.10.2 Function Documentation

#### 570.10.2.1 elektraModulesClose()

```
int elektraModulesClose (
            KeySet * modules,
            Key * error )
```

Close all modules.
Iterates over all modules and closes each of them.
Finish all affairs with the modules. Delete all keys where the appropriate module could be closed.
If it is not possible to close a module, still try to close all other modules, but report the error with the error key.

**Parameters**

| | |
|---|---|
| *modules* | all modules in this keyset will be closed |
| *error* | a key to append the error information if it is not null |

**Returns**

-1 on error

>=0 otherwise

**Postcondition**

all error information is stored in the key 'error'

#### 570.10.2.2 elektraModulesInit()

```
int elektraModulesInit (
            KeySet * modules,
            Key * error )
```

Initialises the module loading system.
Most operating systems will have to do nothing here. Anyway you are required to add the key system↩ :/elektra/modules if it was successful.
On error -1 is returned and if error != 0 error information is added to it.

**Parameters**

| | |
|---|---|
| *modules* | an empty keyset |
| *error* | a key to append the error information if it is not null |

**Returns**

-1 on error

>=0 otherwise

**Postcondition**

the key system:/elektra/modules is added in case of success

#### 570.10.2.3 elektraModulesLoad()

```
elektraPluginFactory elektraModulesLoad (
            KeySet * modules,
            const char * name,
            Key * error )
```

Load a library with the given name.

**Returns**

a pointer to the factory which can create the plugin.

Make sure that you first lookup if this module was already loaded. If it was, just return the pointer and you are done. Otherwise load the module/library given by name. You need to take care that a proper name is used. The name does not have any path, pre- or postfixes.
The next step is to fetch the symbol elektraPluginFactory.

If everything was successful append all information to the keyset modules and return the pointer. Take care that you can close the module with that information. All information needs to be stored within system:/elektra/modules/name You might want to use an struct and store it there as binary key.
If anything goes wrong don't append anything to modules. Instead report the error to the error key and return with 0.

**Precondition**

the name is not null, empty and has at least one character different to '/'. It is suitable to be used as keyAdd↩ BaseName without any further error checking.

**Parameters**

| | |
|---|---|
| *modules* | where to get existing modules from a new module will be added there |
| *name* | the name for the plugin to load. Note that it does not have any prefixes or postfixes, you need to add them yourself. |
| *error* | the key to add warnings or report errors |

**Returns**

a pointer which can create a Plugin

0 on error

## 570.11 Name Manipulation Methods

Methods to do various operations on Key names.
Collaboration diagram for Name Manipulation Methods:


### Functions

- const char * keyName (const Key *key)

  *Returns a pointer to the abbreviated real internal `key` name.*
- ssize_t keyGetNameSize (const Key *key)

  *Bytes needed to store the Key's name (excluding owner).*
- const void * keyUnescapedName (const Key *key)

  *Returns a Key's name, separated by NULL bytes and without backslashes for escaping.*
- ssize_t keyGetUnescapedNameSize (const Key *key)

  *Returns the size of the Key's unescaped name including embedded and terminating NULL characters.*
- ssize_t keyGetName (const Key *key, char *returnedName, size_t maxSize)

  *Get abbreviated Key name (excluding owner).*
- ssize_t keyGetUnescapedName (const Key *key, char *returnedName, size_t maxSize)

  *Copies the unescaped name of a Key into `returnedName`.*
- ssize_t keySetName (Key *key, const char *newName)

  *Set a new name to a Key.*

- ssize_t keyAddName (Key *key, const char *newName)

  *Add an already escaped name part to the Key's name.*
- int keyReplacePrefix (Key *key, const Key *oldPrefix, const Key *newPrefix)

  *Replaces a prefix of the key name of `key`.*
- bool elektraKeyNameValidate (const char *name, bool isComplete)

  *Takes an escaped key name and validates it.*
- void elektraKeyNameCanonicalize (const char *name, char **canonicalName, size_t *canonicalSizePtr, size_t offset, size_t *usizePtr)

  *Takes a valid (non-)canonical key name and produces its canonical form.*
- void elektraKeyNameUnescape (const char *canonicalName, char *unescapedName)

  *Takes a canonical key name and unescapes it.*
- const char * keyBaseName (const Key *key)

  *Returns a pointer to the unescaped Key's name where the basename starts.*
- ssize_t keyGetBaseNameSize (const Key *key)

  *Calculates number of bytes needed to store basename of `key` (including NULL terminator).*
- ssize_t keyGetBaseName (const Key *key, char *returned, size_t maxSize)

  *Copy the Key's basename to `returned`.*
- size_t elektraKeyNameEscapePart (const char *part, char **escapedPart)

  *Takes a single key name part and produces its escaped form.*
- ssize_t keyAddBaseName (Key *key, const char *baseName)

  *Adds `baseName` to the name of `key`.*
- ssize_t keySetBaseName (Key *key, const char *baseName)

  *Sets `baseName` as the new basename for `key`.*
- elektraNamespace keyGetNamespace (const Key *key)

  *Returns the elektraNamespace for a Key.*
- ssize_t keySetNamespace (Key *key, elektraNamespace ns)

  *Changes the namespace of a Key.*
- int elektraIsArrayPart (const char *namePart)

  *Checks if the given key name part is an array part.*

### 570.11.1 Detailed Description

Methods to do various operations on Key names.
To use them:

```
#include <kdb.h>
```

These functions make it easier for C programmers to work with key names.

#### Terminology of Key Names

- A *key name* (see keySetName() and keyName()) defines the place of a key within the key database. To be unique, it is always absolute and canonical.
- Key names are composed out of many *key name parts* split by a separator. These *key name parts* do not contain an unescaped separator.
- A *key base name* (see keySetBaseName() and keyAddBaseName()) is the last part of the key name.
- A *C-String* is a null terminated sequence of characters. So \0 (null-character) must not occur within a C-String.

#### Namespaces

A namespace denotes the place the key comes from:

- `spec:/something` for specification of other keys.

`proc:/something` for in-memory keys, e.g. commandline.

- `dir:/something` for dir keys in current working directory
- `system:/something` for system keys in /etc or /
- `user:/something` for user keys in home directory
- `user:username/something` for other users (deprecated: kdbGet() + kdbSet() currently unsupported)
- `/something` for cascading keys (actually refers to one of the above, see also ksLookup())

##### Note

The rules are currently not formally specified and are subject of change in the next major release. So, always prefer:

- To use keySetName() and keyAddName() to get the canonified version of the keyname
- To use keySetBaseName() and keyAddBaseName() to get an escaped key name part.
- Not to escape or canonify with your own algorithms!
- To use keyUnescapedName() and keyBaseName() to have access to the key name without escape sequences (key name parts are null terminated)
- Not to unescape the strings yourself!

#### Syntax for Key Names

Key names and key name parts have following goals:

- The C-String passed to keySetName() and keyAddName() may be any C-String.
- The *key name parts* (e.g. keySetBaseName(), keyBaseName()) may be any C-String. Escaping is needed to achieve both goals.

#### Semantics for Key Name Parts

- % denotes an empty key name part.

#### Canonicalization for Key Names

- / (slash) is the separator between key name parts.
- // is shortened to /
- trailing / (slashes) are removed
- . (dot) and .. (dot-dot) is removed in an canonical key name, with following rules:
  - /./ is shortened to /
  - _/../ is shortened to _

#### Conventions for key names

- Key name parts starting with # are array elements. Then only _ (underscore) followed by 0-9 is allowed. So we have the regular expression #[_]*[0-9]+ with the further limitation that the number of _ is defined by the number of digits-1.
- Key name parts starting with _ are reserved for special purposes (if you use this within a plugin you still have to make sure _ is escaped properly)
- Key name parts starting with @ are reserved for special purposes (if you use this within a plugin you still have to make sure @ is escaped properly)

**Escaping rules**

- \ (backslash) is the escape character for the situations as described here (and only these). The \ character must only be escaped, when one of the following rules apply.
- Stray escape characters are only possible in the end of the string.
- \/ allows one to escape / (any uneven number of \). Does not introduce a new part.
- Any uneven number N of \ before / allows you to escape / with the N/2 of \ prefixed. Does not introduce a new part.
- \\/ allows one to use \ as character before / and introduces a new part.
- Any even number N of \ before / allows you to have N/2 of \ prefixed before a / which introduces a new part.
- Use \. and \.. if you want your key name part to represent . and ..
- \\. and \\.. allows us to use \ as character before . and .. (and so on)
- Use \% if you want your key name part to start with % (and does not represent an empty name)
- Using \\% allows one to use \ as character before % (and so on)

**Semantics for Key Name Specifications**

- _ denotes that the key name part is arbitrary (syntax as described above).
- # denotes that the key name part has array syntax.
- names surrounded by % (e.g. %profile%) denotes a placeholder.

## 570.11.2 Function Documentation

### 570.11.2.1 elektraIsArrayPart()

```
int elektraIsArrayPart (
            const char * namePart )
```

Checks if the given key name part is an array part.
The return value of this function can safely be treated as a boolean.

**Parameters**

| | |
|---|---|
| *namePart* | an arbitrary string that shall be checked |

**Returns**

if `namePart` is an array part, the index of the first digit in `namePart`, or 0 otherwise

### 570.11.2.2 elektraKeyNameCanonicalize()

```
void elektraKeyNameCanonicalize (
            const char * name,
            char ** canonicalName,
            size_t * canonicalSizePtr,
            size_t offset,
            size_t * usizePtr )
```

Takes a valid (non-)canonical key name and produces its canonical form.
As a side-effect it can also calculate the size of the corresponding unescaped key name.

**Parameters**

| | |
|---|---|
| *name* | The key name that is processed |
| *canonicalName* | Output buffer for the canonical name |

**Parameters**

| | |
|---|---|
| *canonicalSizePtr* | Pointer to size of `canonicalName` |
| *offset* | Offset into `canonicalName` |
| *usizePtr* | Output variable for the size of the unescaped name |

**Precondition**

`name` MUST be a valid (non-)canonical key name. If it is not, the result is undefined

`canonicalName` MUST be a valid first argument for elektraRealloc() when cast to void**

`canonicalSizePtr` >= `offset`

`offset` MUST be 0 or `*canonicalName + offset` MUST point to the zero-termintor of a valid canonical key name that starts at `*canonicalName`

if `offset` is 0 then `*usizePtr` MUST 0, otherwise `*usizePtr` MUST be the correct unescaped size of the existing canonical name in `*canonicalName`

**See also**

elektraKeyNameValidate

### 570.11.2.3 elektraKeyNameEscapePart()

```
size_t elektraKeyNameEscapePart (
            const char * part,
            char ** escapedPart )
```

Takes a single key name part and produces its escaped form.

**Parameters**

| | |
|---|---|
| *part* | A single key name part, i.e. contained '/' will be escaped, '\0' terminates part |
| *escapedPart* | Output buffer for the escaped form |

**Precondition**

`escapedPart` MUST be a valid first argument for elektraRealloc() when cast to `void**`

**Returns**

The size of the escaped form excluding the zero terminator

### 570.11.2.4 elektraKeyNameUnescape()

```
void elektraKeyNameUnescape (
            const char * canonicalName,
            char * unescapedName )
```

Takes a canonical key name and unescapes it.

**Parameters**

| | |
|---|---|
| *canonicalName* | The canonical name to unescape |
| *unescapedName* | Output buffer for the unescaped name |

**Precondition**

`canonicalName` MUST be a canonical key name. If this is not the case, the result is undefined.

`unescapedName` MUST be allocated to the correct size.

**See also**

elektraKeyNameCanonicalize

### 570.11.2.5 elektraKeyNameValidate()

```
bool elektraKeyNameValidate (
            const char * name,
            bool isComplete )
```

Takes an escaped key name and validates it.
Complete key names must inlcude a namespace or a leading slash.

**Parameters**

| | |
|---|---|
| *name* | The escaped key name to check |
| *isComplete* | Whether or not `name` is supposed to be a complete key name |

**Return values**

| | |
|---|---|
| *#true* | If `name` is a valid key name. |
| *#false* | Otherwise |

### 570.11.2.6 keyAddBaseName()

```
ssize_t keyAddBaseName (
            Key * key,
            const char * baseName )
```

Adds `baseName` to the name of `key`.
`baseName` will be escaped before adding it to the name of `key`. No other part of the Key's name will be affected. Assumes that `key` is a directory and will append `baseName` to it. The function adds the path separator for concatenating.
If `key` has the name `"system:/dir1/dir2"` and this method is called with `baseName` `"mykey"`, the resulting key will have the name `"system:/dir1/dir2/mykey"`.
When `baseName` is 0, nothing will happen and the size of the name is returned.
The escaping rules apply as in above .
A simple example is:

```
Key * k = keyNew ("user:/my/long", KEY_END);
keyAddBaseName (k, "myname");
printf ("%s\n", keyName (k)); // will print user:/my/long/myname
keyDel (k);
```

E.g. if you add . it will be escaped:

```
        keySetName (k, "system:/valid");
        succeed_if (keyAddBaseName (k, ".") >= 0, "could not add a base name");
        succeed_if_same_string (keyName (k), "system:/valid/\\.");
        succeed_if_same_string (keyBaseName (k), ".");
```

**Parameters**

| | |
|---|---|
| *key* | the Key to add the basename to |
| *baseName* | the string to append to the Key's name |

#### Returns

the size in bytes of the Key's new name including the NULL terminator

#### Return values

| | |
|---|---|
| *-1* | if the Key has no name |
| *-1* | on NULL pointers |
| *-1* | if Key was inserted into KeySet before |
| *-1* | if the Key was read-only |
| *-1* | on memory allocation errors |

#### Since

1.0.0

#### See also

keySetBaseName() for setting the basename of a Key

keySetName() for setting the name of a Key

### 570.11.2.7 keyAddName()

```
ssize_t keyAddName (
            Key * key,
            const char * newName )
```

Add an already escaped name part to the Key's name.
The same way as in keySetName() this method finds the canonical pathname:

- it will ignore /./
- it will remove a level when /../ is used
- it will remove multiple slashes ////

For example:

```
Key * k = keyNew ("user:/x/r", KEY_END);
keyAddName (k, "../y/a//./z");
assert (!strcmp (keyName (k), "user:/x/y/a/z"));
keyDel (k);
```

Unlike keySetName() it adds relative to the previous name and cannot change the namespace of a Key. For example:

```
Key * n = keyNew ("user:/away", KEY_END);
keyAddName (n, "../../../new/name");
assert (!strcmp (keyName (n), "user:/new/name"));
keyDel (n);
```

The passed name needs to be valid according the key name rules . It is not allowed to:

- be empty
- end with unequal number of \

#### Precondition

`key` MUST be a valid #Key

#### Parameters

| | |
|---|---|
| *key* | the Key where a name should be added |
| *newName* | the new name to add to the name of `key` |

**Returns**

new size of the escaped name of `key`

**Return values**

| | |
|---|---|
| *-1* | if `key == NULL` or `newName == NULL` |
| *-1* | `newName` is not a valid escaped name |
| *-1* | `key` is read-only |

**Since**

1.0.0

**See also**

keySetName() for setting a Key's name

keyAddBaseName() for adding a basename to a Key

### 570.11.2.8 keyBaseName()

```
const char* keyBaseName (
            const Key * key )
```

Returns a pointer to the unescaped Key's name where the basename starts.
This is a much more efficient version of keyGetBaseName() and you should use it if you are responsible enough to not mess up things. The name might change or even point to a wrong place after a keySetName(). So make sure to copy the memory before the name changes.
keyBaseName() returns "" when the Key has no basename. The reason is

```
keySetName (k, "");
succeed_if_same_string (keyBaseName (k), "");
keySetName (k, "user:/");
succeed_if_same_string (keyBaseName (k), "");
```

There is also support for really empty basenames:

```
keySetName (k, "system:/valid");
succeed_if (keyAddBaseName (k, "") >= 0, "could not add a base name");
succeed_if_same_string (keyName (k), "system:/valid/%");
succeed_if_same_string (keyBaseName (k), "");
```

**Note**

You must never use the pointer returned by keyBaseName() method to change the name. You should use keySetBaseName() instead.

Do not assume that keyBaseName() points to the same region as keyName() does.

**Parameters**

| | |
|---|---|
| *key* | the Key to obtain the basename from |

**Returns**

a pointer to the Key's basename

**Return values**

| | |
|---|---|
| *""* | when the Key has no (base)name |
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

keyGetBaseName() for getting a copy of the Key's basename

keyGetBaseNameSize() for getting the size of the Key's basename

keyName() for getting a pointer to the Key's name

### 570.11.2.9 keyGetBaseName()

```
ssize_t keyGetBaseName (
            const Key * key,
            char * returned,
            size_t maxSize )
```

Copy the Key's basename to `returned`.
The copy will include a NULL terminator which will be considered for the returned size. Nothing will be copied if `maxSize` is smaller than the size of the basename.
Some examples:

- basename of `system:/some/keyname` is `keyname`
- basename of `"user:/tmp/some key"` is `"some key"`

**Parameters**

| | |
|---|---|
| *key* | the Key to extract basename from |
| *returned* | a pre-allocated buffer for storing the basename |
| *maxSize* | size of the buffer `returned` |

**Returns**

number of bytes copied to `returned`

**Return values**

| | |
|---|---|
| *1* | when Key's name is empty |
| *-1* | on NULL pointers |
| *-1* | when maxSize is 0 or larger than SSIZE_MAX |
| *-1* | when maxSize is smaller than the size of the Key's basename |

**Since**

1.0.0

**See also**

keyBaseName() for getting a pointer to the Key's basename

keyGetBaseNameSize() for getting the size of a Key's basename

keyName(), keyGetName() for getting a pointer / copy of the whole name

keySetName() for setting a Key's name

#### 570.11.2.10 keyGetBaseNameSize()

```
ssize_t keyGetBaseNameSize (
            const Key * key )
```

Calculates number of bytes needed to store basename of `key` (including NULL terminator).
Key names consisting of only root names (e.g. "`system:/`" or "`user:/`" or "`user:domain`" ) do not have basenames. In this case the function will return 1, because only a NULL terminator is needed for storage.
Basenames are denoted as:

- `system:/some/thing/basename` -> `basename`
- `user:domain/some/thing/base\/name` > `base\/name`

**Parameters**

| | |
|---|---|
| *key* | the Key to get the size of the basename from |

**Returns**

size in bytes of the Key's basename including NULL terminator

**Return values**

| | |
|---|---|
| *-1* | if the Key or the Key's basename is NULL |

**Since**

1.0.0

**See also**

keyBaseName() for getting a pointer to a Key's basename

keyGetBaseName() for getting a copy of a Key's basename

keyName(), keyGetName() for getting a pointer / copy of the whole name

keySetName() for setting a Key's name

#### 570.11.2.11 keyGetName()

```
ssize_t keyGetName (
            const Key * key,
            char * returnedName,
            size_t maxSize )
```

Get abbreviated Key name (excluding owner).
When there is not enough space to write the name, nothing will be written and -1 will be returned.
`maxSize` is limited to SSIZE_MAX. When this value is exceeded, -1 will be returned. The reason for that is, that any value higher is just a negative return value passed by accident. elektraMalloc() is not as failure tolerant and would try to allocate memory accordingly.

```
char *getBack = elektraMalloc (keyGetNameSize(key));
keyGetName(key, getBack, keyGetNameSize(key));
```

**Parameters**

| | |
|---|---|
| *key* | the Key to get the name from |
| *returnedName* | pre-allocated buffer to write the Key's name |
| *maxSize* | maximum number of bytes that will fit in returnedName, including the NULL terminator |

**Returns**

number of bytes written to `returnedName`

**Return values**

| | |
|---|---|
| *1* | when only NULL terminator was written |
| *-1* | when Key's name is longer than maxSize or maxSize is 0 or maxSize is greater than SSIZE_MAX |
| *-1* | `key` or `returnedName` is NULL pointer |

**Since**

1.0.0

**See also**

keyGetNameSize() for getting the size of a Key's name

keyName() for getting a pointer to a Key's name

keyGetBaseName() for getting a Key's base name

keyGetNamespace() for getting the namespace of a Key's name

### 570.11.2.12 keyGetNameSize()

```
ssize_t keyGetNameSize (
            const Key * key )
```

Bytes needed to store the Key's name (excluding owner).
For an empty Key name you need one byte to store the ending NULL. For that reason, 1 is returned when the name is empty.

**Parameters**

| | |
|---|---|
| *key* | the Key to get the name size from |

**Returns**

number of bytes needed, including NULL terminator, to store Key's name (excluding owner)

**Return values**

| | |
|---|---|
| *1* | if Key has no name |
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

keyGetName() for getting the Key's name

keyGetUnescapedNameSize() for getting the size of the unescaped name

### 570.11.2.13 keyGetNamespace()

```
elektraNamespace keyGetNamespace (
            const Key * key )
```

Returns the elektraNamespace for a Key.

To handle every namespace a Key could have, you can use the following snippet:

```
switch (keyGetNamespace (k))
{
case KEY_NS_SPEC:
        printf ("spec namespace\n");
        break;
case KEY_NS_PROC:
        printf ("proc namespace\n");
        break;
case KEY_NS_DIR:
        printf ("dir namespace\n");
        break;
case KEY_NS_USER:
        printf ("user namespace\n");
        break;
case KEY_NS_SYSTEM:
        printf ("system namespace\n");
        break;
case KEY_NS_NONE:
        printf ("no key\n");
        break;
case KEY_NS_META:
        printf ("metakey\n");
        break;
case KEY_NS_CASCADING:
        printf ("cascading key\n");
        break;
}
```

To loop over all valid namespaces use:

```
for (elektraNamespace ns = KEY_NS_FIRST; ns <= KEY_NS_LAST; ++ns)
{
        // work with namespace
        printf ("%d\n", ns);
}
```

#### Note

This method might be extended. There is no guarantee that a Key with a specific namespace will retain that namespace after recompilation. Make sure that your compiler gives you a warning for unhandled switches (gcc: -Wswitch or -Wswitch-enum if you want to handle default) and look out for those warnings when recompiling.

#### Parameters

| | |
|---|---|
| *key* | the Key to get the namespace from |

#### Returns

the namespace of the Key

#### Return values

| | |
|---|---|
| *KEY_NS_NONE* | if Key is NULL |

#### Since

1.0.0

#### See also

keySetNamespace() for setting a Key's namespace

#### 570.11.2.14 keyGetUnescapedName()

```
ssize_t keyGetUnescapedName (
            const Key * key,
            char * returnedName,
            size_t maxSize )
```

Copies the unescaped name of a Key into `returnedName`.
It will only copy the whole name. If the buffer is too small, an error code will be returned.
To ensure the buffer is big enough, you can use keyGetUnescapedNameSize() to get the correct size.

**Parameters**

| | |
|---|---|
| *key* | the Key to extract the unescaped name from |
| *returnedName* | the buffer to write the unescaped name into |
| *maxSize* | maximum number of bytes that can be copied into `returnedName` |

**Precondition**

`key` MUST be a valid #Key and `key != NULL`

`returnedName` MUST be allocated to be at least `maxSize` bytes big

`returnedName` must not be NULL

**Returns**

the actual size of the Key's unescaped name, i.e. the number of bytes copied into `returnedName`

**Return values**

| | |
|---|---|
| *-1* | Precondition error |
| *-2* | the size of the unescaped name is greater than `maxSize` |

**Since**

1.0.0

**See also**

keyGetUnescapedNameSize() for getting the size of the unescaped name

keyGetName() for getting the Key's escaped name

#### 570.11.2.15 keyGetUnescapedNameSize()

```
ssize_t keyGetUnescapedNameSize (
            const Key * key )
```

Returns the size of the Key's unescaped name including embedded and terminating NULL characters.

**Parameters**

| | |
|---|---|
| *key* | the Key where to get the size of the unescaped name from |

**Returns**

The size of the Key's unescaped name

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |
| *0* | if Key has no name |

**Since**

1.0.0

**See also**

keyUnescapedName() for getting a pointer to the unescaped name

keyGetUnescapedName() for getting a copy of the unescaped name

keyGetNameSize() for getting the size of the escaped name

### 570.11.2.16 keyName()

```
const char* keyName (
            const Key * key )
```

Returns a pointer to the abbreviated real internal `key` name.
This is a much more efficient version of keyGetName() and can use it if you are responsible enough to not mess up things. You are not allowed to change anything in the returned array. The content of that string may change after keySetName() and similar functions. If you need a copy of the name, consider using keyGetName().

**Return values**

| | |
|---|---|
| *""* | when there is no keyName. The reason is |

```
key=keyNew(0);
keySetName(key,"");
keyName(key); // you would expect "" here
keyDel(key);
```

Valid key names are:

- `spec:/something` for specification of other keys.
- `proc:/something` for in-memory keys, e.g. commandline.
- `dir:/something` for dir keys in current working directory
- `system:/something` for system keys in /etc or /
- `user:/something` for user keys in home directory
- `user:username/something` for other users (deprecated: kdbGet() + kdbSet() currently unsupported)
- `/something` for cascading keys (actually refers to one of the above, see also ksLookup())

**Note**

Note that the Key structure keeps its own size field that is calculated by library internal calls, so to avoid inconsistencies, you must never use the pointer returned by keyName() method to set a new value. Use keySetName() instead.

**Parameters**

| | |
|---|---|
| *key* | the Key you want to get the name from |

#### Returns

a pointer to the Key's name which must not be changed.

#### Return values

| | |
|---|---|
| *""* | when Key's name is empty |
| *0* | on NULL pointer |

#### Since

1.0.0

#### See also

keyGetNameSize() for the string length

keyGetName() as alternative to get a copy

keyUnescapedName to get an unescaped Key name

### 570.11.2.17 keyReplacePrefix()

```
int keyReplacePrefix (
              Key * key,
              const Key * oldPrefix,
              const Key * newPrefix )
```

Replaces a prefix of the key name of `key`.
The function only modifies `key`, if is is below (or same as) `oldPrefix` (see keyIsBelowOrSame()) and they both have the same namespace (this is not always the case with keyIsBelowOrSame()).
In simple terms this function operates as follows:

1. If before calling this function `key` and `oldPrefix` had the same name, then afterwards `key` will have the same name as `newPrefix`.
2. If `key` was in the same namespace as and below `oldPrefix`, then after calling this function `key` will be in the same namespace as and below `newPrefix`.
3. Otherwise `key` will not be modified.

Note: We use `const Key *` arguments for the prefixes instead of `const char *` to ensure only valid key names can be passed as arguments.

#### Parameters

| | |
|---|---|
| *key* | The key that will be manipulated. |
| *oldPrefix* | The name of this key will be removed from the front of the name of `key`. |
| *newPrefix* | The name of this key will be added to the front of `key`, after the name of `oldPrefix` is removed. |

#### Return values

| | |
|---|---|
| *-1* | if `key`, `oldPrefix` or `newPrefix` are NULL or the name of `key` is marked as read-only |
| *0* | if `key` is not below (or same as) `oldPrefix`, i.e. there is no prefix to replace |
| *1* | if the prefix was sucessfully replaced |

### 570.11.2.18 keySetBaseName()

```
ssize_t keySetBaseName (
            Key * key,
            const char * baseName )
```

Sets `baseName` as the new basename for `key`.
Only the basename of the Key will be affected.
A simple example is:

```
Key * k = keyNew ("user:/my/long/name", KEY_END);
keySetBaseName (k, "myname");
printf ("%s\n", keyName (k)); // will print user:/my/long/myname
keyDel (k);
```

All text after the last `'/'` in the Key's name is erased and `baseName` is appended. If `baseName` is 0 (NULL), then the last part of the Key's name is removed without replacement. The root name of the Key will not be removed though.
Let us suppose `key` has name `"system:/dir1/dir2/key1"`. If `baseName` is `"key2"`, the resulting key name will be `"system:/dir1/dir2/key2"`. If `baseName` is 0 (NULL), the resulting key name will be `"system:/dir1/dir2"`. If `baseName` is empty, the resulting key name will be `"system:/dir1/dir2/%"`, where `"%"` denotes an empty base name, as also shown in the following code:

```
keySetName (k, "system:/valid");
keySetBaseName (k, "");
succeed_if_same_string (keyName (k), "system:/%");
succeed_if_same_string (keyBaseName (k), "");
```

keySetBaseName() does proper escaping on the supplied name argument.
You can use character sequences as `baseName` (e.g. `"."` (dot), `".."` (dot-dot), `"%"` (empty basename)). They will be properly escaped and will not have their usual meaning.
If you want to add to the basename instead of changing it, use keyAddBaseName(). If you do not want any escaping, use keyAddName().

#### Parameters

| | |
|---|---|
| *key* | the Key whose basename to set |
| *baseName* | the new basename for the Key |

#### Returns

the size in bytes of the new key name

#### Return values

| | |
|---|---|
| *-1* | if Key is NULL |
| *-1* | if Key was inserted into KeySet before |
| *-1* | if Key is read-only |
| *-1* | on allocation errors |

#### Since

1.0.0

#### See also

keyAddBaseName() for adding a basename instead of changing it

keyAddName() for adding a name without escaping

keySetName() for setting a completely new name

Name Manipulation Methods for more details on special names

#### 570.11.2.19 keySetName()

```
ssize_t keySetName (
            Key * key,
            const char * newName )
```

Set a new name to a Key.
A valid name is one of the forms:

- `spec:/something` for specification of other keys.
- `proc:/something` for in-memory keys, e.g. commandline.
- `dir:/something` for dir keys in current working directory
- `system:/something` for system keys in /etc or /
- `user:/something` for user keys in home directory
- `user:username/something` for other users (deprecated: kdbGet() + kdbSet() currently unsupported)
- `/something` for cascading keys (actually refers to one of the above, see also ksLookup())

An invalid name either has an invalid namespace or a wrongly escaped \ at the end of the name.
See key names for the exact rules.
The last form has explicitly set the owner, to let the library know in which user folder to save the Key. A owner is a user name. If it is not defined (the second form), current user is used.
You should always follow the guidelines for Key tree structure creation.
A private copy of the Key name will be stored, and the `newName` parameter can be freed after this call.
.., . and / will be handled as in filesystem paths. A valid name will be build out of the (valid) name what you pass, e.g. user:///sw/../sw//./../MyApp -> user:/sw/MyApp
Trailing slashes will be stripped.
On invalid names, the name stays unchanged.

**Returns**

size of the new Key name in bytes, including NULL terminator

**Return values**

| | |
|---|---|
| *-1* | if `key` or `keyName` is NULL or `keyName` is empty or invalid |
| *-1* | if Key was inserted to a KeySet before |
| *-1* | if Key name is read-only |

**Parameters**

| | |
|---|---|
| *key* | the Key whose name to set |
| *newName* | the new name for the Key |

**Since**

1.0.0

**See also**

keyGetName() for getting a copy of the Key's name

keyName() for getting a pointer to the Key's name

keySetBaseName(), keyAddBaseName() for manipulating the base name

### 570.11.2.20 keySetNamespace()

```
ssize_t keySetNamespace (
            Key * key,
            elektraNamespace ns )
```

Changes the namespace of a Key.
The rest of the Key's name remains unchanged.

#### Precondition

`ns` MUST be a valid namespace and not KEY_NS_NONE

`key` MUST be a valid #Key, especially `key != NULL`

#### Parameters

| | |
|---|---|
| *key* | The #Key whose namespace will be changed |
| *ns* | The new namespace of for `key` |

#### Returns

the new size in bytes of the Key's namespace

#### Return values

| | |
|---|---|
| *-1* | precondition error |

#### Since

1.0.0

#### See also

keyGetNamespace() for getting a Key's namespace

### 570.11.2.21 keyUnescapedName()

```
const void* keyUnescapedName (
            const Key * key )
```

Returns a Key's name, separated by NULL bytes and without backslashes for escaping.
Slashes are replaced with NULL bytes. Therefore unescaped names of cascading Keys start with a NULL byte. Otherwise escaped characters, e.g. non-hierarchy slashes, will be unescaped.
This name is essential if you want to iterate over parts of the Key's name, compare Key names or check relations of Keys in the hierarchy.

#### Parameters

| | |
|---|---|
| *key* | the Key whose unescaped name to get |

#### Returns

the name in its unescaped form

#### Return values

| | |
|---|---|
| *0* | on NULL pointers |

**Return values**

| | |
|---|---|
| "" | if Key's name is empty |

**Since**

1.0.0

**See also**

keyGetUnescapedName() for getting a copy of the unescaped name

keyGetUnescapedNameSize() for getting the size of the unescaped name

keyName() for getting the escaped name of the Key

## 570.12 Notification

Notification feature.

### Files

- file kdbnotification.h

  *Elektra-Notification structures and declarations for application developers.*
- file kdbnotificationinternal.h

  *Elektra-Notification structures and declarations for developing notification and transport plugins.*

### Typedefs

- typedef void(∗ ElektraNotificationConversionErrorCallback) (Key ∗key, void ∗context)

  *Callback function called when string to number conversion failed.*
- typedef void(∗ ElektraNotificationChangeCallback) (Key ∗key, void ∗context)

  *Callback function for key changes.*

### Functions

- int elektraNotificationRegisterCallback (KDB ∗kdb, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

  *Subscribe for updates via callback when a given key value is changed.*
- int elektraNotificationRegisterCallbackSameOrBelow (KDB ∗kdb, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

  *Subscribe for updates via callback when a given key or a key below changed.*
- int elektraNotificationRegisterInt (KDB ∗kdb, Key ∗key, int ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterUnsignedInt (KDB ∗kdb, Key ∗key, unsigned int ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterLong (KDB ∗kdb, Key ∗key, long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterUnsignedLong (KDB ∗kdb, Key ∗key, unsigned long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterLongLong (KDB ∗kdb, Key ∗key, long long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterUnsignedLongLong (KDB ∗kdb, Key ∗key, unsigned long long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterFloat (KDB ∗kdb, Key ∗key, float ∗variable)

*Subscribe for automatic updates to a given variable when the given key value is changed.*

- int elektraNotificationRegisterDouble (KDB *kdb, Key *key, double *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbBoolean (KDB *kdb, Key *key, kdb_boolean_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbChar (KDB *kdb, Key *key, kdb_char_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbOctet (KDB *kdb, Key *key, kdb_octet_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbShort (KDB *kdb, Key *key, kdb_short_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbUnsignedShort (KDB *kdb, Key *key, kdb_unsigned_short_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbLong (KDB *kdb, Key *key, kdb_long_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbUnsignedLong (KDB *kdb, Key *key, kdb_unsigned_long_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbLongLong (KDB *kdb, Key *key, kdb_long_long_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbUnsignedLongLong (KDB *kdb, Key *key, kdb_unsigned_long_long_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbFloat (KDB *kdb, Key *key, kdb_float_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbDouble (KDB *kdb, Key *key, kdb_double_t *variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*

### 570.12.1 Detailed Description

Notification feature.
For an introduction to notifications please see the `Notification Tutorial`.
Examples:

- `Basic notifications using polling`
- `Using asynchronous I/O bindings`
- `Reload KDB when Elektra's configuration has changed`

#### Hook Setup

elektraNotificationContract() returns a contract for use with kdbOpen(). The contract ensures that the internalnotification plugin is registered into the correct hook and configured correctly.

#### Transport Plugins

Notification transport plugins (or simply transport plugins) need access to an I/O binding, as well as a notification callback and context.
All of these can be retrieved from the global keyset. The keys are as follows:

- I/O binding: `system:/elektra/io/binding` Type: `ElektraIoInterface *`
- Callback: `system:/elektra/notification/callback` Type: `ElektraNotificationCallback`

- Context: `system:/elektra/notification/context` Type: `ElektraNotification↩ CallbackContext *`

All of these keys are binary and store a pointer that can be read via `*(TYPE **) keyValue (key)`.
The I/O binding can be accessed at any time. It is recommended, plugins read the key once during their `open` function and store the pointer in their plugin data struct.
The notification callback and context are provided by the `internalnotification` plugin. Since it might be initialised after the transport plugin, it is not recommended to read the callback and context in the `open` function. Instead the plugin should read and store the values when the first notification is processed.
Transport plugins should handle missing I/O bindings, notification callbacks and notification contexts gracefully. The plugin should not report and error and instead simply log a debug or warning message.

## 570.12.2 Typedef Documentation

### 570.12.2.1 ElektraNotificationChangeCallback

```
typedef void(* ElektraNotificationChangeCallback) (Key *key, void *context)
```

Callback function for key changes.

**Parameters**

| | |
|---|---|
| *key* | changed key |
| *context* | user supplied callback context |

### 570.12.2.2 ElektraNotificationConversionErrorCallback

```
typedef void(* ElektraNotificationConversionErrorCallback) (Key *key, void *context)
```

Callback function called when string to number conversion failed.

**Parameters**

| | |
|---|---|
| *key* | key with invalid value |
| *context* | user supplied callback context |

## 570.12.3 Function Documentation

### 570.12.3.1 elektraNotificationRegisterCallback()

```
int elektraNotificationRegisterCallback (
            KDB * kdb,
            Key * key,
            ElektraNotificationChangeCallback callback,
            void * context )
```

Subscribe for updates via callback when a given key value is changed.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *callback* | callback function |
| *context* | user supplied context passed to callback function |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.2 elektraNotificationRegisterCallbackSameOrBelow()

```
int elektraNotificationRegisterCallbackSameOrBelow (
            KDB * kdb,
            Key * key,
            ElektraNotificationChangeCallback callback,
            void * context )
```

Subscribe for updates via callback when a given key or a key below changed.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *callback* | callback function |
| *context* | user supplied context passed to callback function |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.3 elektraNotificationRegisterDouble()

```
int elektraNotificationRegisterDouble (
            KDB * kdb,
            Key * key,
            double * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.4 elektraNotificationRegisterFloat()

```
int elektraNotificationRegisterFloat (
```

```
            KDB * kdb,
            Key * key,
            float * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.5 elektraNotificationRegisterInt()

```
int elektraNotificationRegisterInt (
            KDB * kdb,
            Key * key,
            int * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

Subscribe for automatic updates to a given variable when the given key value is changed. On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.6 elektraNotificationRegisterKdbBoolean()

```
int elektraNotificationRegisterKdbBoolean (
            KDB * kdb,
            Key * key,
            kdb_boolean_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.7 elektraNotificationRegisterKdbChar()

```
int elektraNotificationRegisterKdbChar (
            KDB * kdb,
            Key * key,
            kdb_char_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.8 elektraNotificationRegisterKdbDouble()

```
int elektraNotificationRegisterKdbDouble (
            KDB * kdb,
            Key * key,
            kdb_double_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.9 elektraNotificationRegisterKdbFloat()

```
int elektraNotificationRegisterKdbFloat (
            KDB * kdb,
            Key * key,
            kdb_float_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.10 elektraNotificationRegisterKdbLong()

```
int elektraNotificationRegisterKdbLong (
            KDB * kdb,
            Key * key,
            kdb_long_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.11 elektraNotificationRegisterKdbLongLong()

```
int elektraNotificationRegisterKdbLongLong (
            KDB * kdb,
            Key * key,
            kdb_long_long_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.12 elektraNotificationRegisterKdbOctet()

```
int elektraNotificationRegisterKdbOctet (
            KDB * kdb,
            Key * key,
            kdb_octet_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.13 elektraNotificationRegisterKdbShort()

```
int elektraNotificationRegisterKdbShort (
            KDB * kdb,
            Key * key,
            kdb_short_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.14 elektraNotificationRegisterKdbUnsignedLong()

```
int elektraNotificationRegisterKdbUnsignedLong (
             KDB * kdb,
             Key * key,
             kdb_unsigned_long_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.15 elektraNotificationRegisterKdbUnsignedLongLong()

```
int elektraNotificationRegisterKdbUnsignedLongLong (
             KDB * kdb,
             Key * key,
             kdb_unsigned_long_long_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.16 elektraNotificationRegisterKdbUnsignedShort()

```
int elektraNotificationRegisterKdbUnsignedShort (
            KDB * kdb,
            Key * key,
            kdb_unsigned_short_t * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.17 elektraNotificationRegisterLong()

```
int elektraNotificationRegisterLong (
            KDB * kdb,
            Key * key,
            long * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.18 elektraNotificationRegisterLongLong()

```
int elektraNotificationRegisterLongLong (
            KDB * kdb,
            Key * key,
            long long * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.19 elektraNotificationRegisterUnsignedInt()

```
int elektraNotificationRegisterUnsignedInt (
            KDB * kdb,
            Key * key,
            unsigned int * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.20 elektraNotificationRegisterUnsignedLong()

```
int elektraNotificationRegisterUnsignedLong (
            KDB * kdb,
            Key * key,
            unsigned long * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 570.12.3.21 elektraNotificationRegisterUnsignedLongLong()

```
int elektraNotificationRegisterUnsignedLongLong (
            KDB * kdb,
            Key * key,
            unsigned long long * variable )
```

Subscribe for automatic updates to a given variable when the given key value is changed.
On kdbGet iff the key is present and its content is valid, the registered variable is updated.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *variable* | variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

## 570.13 Plugins

Elektra plugin framework.

### Macros

- #define ELEKTRA_PLUGIN_FUNCTION(function) ELEKTRA_PLUGIN_FUNCTION2 (ELEKTRA_PLUGIN↩ _NAME_C, function)

  *Declare a plugin's function name suitable for compilation variants (see doc/tutorials).*
- #define ELEKTRA_README ELEKTRA_README2 (ELEKTRA_PLUGIN_NAME_C)

  *The filename for inclusion of the readme for compilation variants (see doc/tutorials).*
- #define ELEKTRA_SET_ERROR(number, key, text)

  *Sets the error in the keys metadata.*
- #define ELEKTRA_SET_ERRORF(number, key, formatstring, ...)

  *Sets the error in the keys metadata.*
- #define ELEKTRA_ADD_WARNINGF(number, key, formatstring, ...)

  *Adds a warning in the keys metadata.*
- #define ELEKTRA_ADD_WARNING(number, key, text)

  *Adds a warning in the keys metadata.*
- #define ELEKTRA_SET_ERROR_GET(parentKey)

  *Set error in kdbGet() when opening the file failed.*
- #define ELEKTRA_SET_ERROR_SET(parentKey)

  *Set error in kdbSet() when opening the file failed.*

### Functions

- Plugin ∗ elektraPluginExport (const char ∗pluginName,...)

  *Allows one to Export Methods for a Plugin.*
- KeySet ∗ elektraPluginGetConfig (Plugin ∗handle)

  *Returns the configuration of that plugin.*

- void elektraPluginSetData (Plugin *plugin, void *data)

  *Store a pointer to plugin specific data.*
- void * elektraPluginGetData (Plugin *plugin)

  *Get a pointer to the plugin specific data stored before.*
- KeySet * elektraPluginGetGlobalKeySet (Plugin *plugin)

  *Get a pointer to the global keyset.*
- int elektraDocOpen (Plugin *handle, Key *warningsKey)

  *Initialize data for the plugin.*
- int elektraDocClose (Plugin *handle, Key *warningsKey)

  *Finalize the plugin.*
- int elektraDocGet (Plugin *handle, KeySet *returned, Key *parentKey)

  *Get data from storage to application.*
- int elektraDocSet (Plugin *handle, KeySet *returned, Key *parentKey)

  *Set data from application to storage.*
- int elektraDocCommit (Plugin *handle, KeySet *returned, Key *parentKey)

  *Make changes to storage final.*
- int elektraDocError (Plugin *handle, KeySet *returned, Key *parentKey)

  *Rollback in case of errors.*
- int elektraDocCheckConf (Key *errorKey, KeySet *conf)

  *Validate plugin configuration at mount time.*

### 570.13.1 Detailed Description

Elektra plugin framework.

**Since**

version 0.4.9, Elektra can dynamically load different key storage plugins.

version 0.7.0 Elektra can have multiple backends, mounted at any place in the key database.

version 0.8.0 Elektra backends are composed out of multiple plugins.

To get started with writing plugins, first read our `Plugin Tutorial`.
A plugin can implement any functionality related to configuration. There are 6 possible entry points for a plugin.

- elektraDocGet() will be called when configuration or the plugin's contract is retrieved from the key database
- elektraDocSet() will be called when configuration is written to the key database
- elektraDocOpen() will be called before any other method of the plugin is called
- elektraDocClose() will be called as last method
- elektraDocError() will be called when kdbSet() failed (to give the plugin a chance to recover/undo its actions)
- elektraPluginExport() exports all methods for the plugin.

Additionally, make sure that you write a contract in the README.md. It is used by the build system and the mounting tools.
Plugins should not change the keyname of the key that is passed to the entry points (warningsKey and parentKey in this documentation). These keys might be members in keysets.
The names described here contain "Doc" within the method's name just because the plugin described in this document is called doc (the doxygen source was generated from src/plugins/doc/doc.h). Always replace Doc with the name of the plugin you are going to implement or use ELEKTRA_PLUGIN_FUNCTION.

#### Overview

There are different types of plugins for different concerns. They all only have the entry points as defined above. The types of plugins handled in this document:

- A storage plugin gets an empty keyset in elektraDocGet() and constructs the information out from a file. In elektraDocSet() the keyset is written to a file.
  Other persistent storage then a file is not handled within this document because it involves many other issues. For files the resolver plugin already takes care of transactions and rollback. So the storage plugin is the source and dump as known from pipes and filters.
- A filter plugin is a plugin which operates on existing keys. It may process or change the keyset. Or it may reject specific keysets which do not meet some criteria.

Use following include to have the functions that are not implemented by you available:

```
#include <kdbplugin.h>
```

#### Error and Warnings

In case of trouble, in some methods you can use the macro ELEKTRA_SET_ERROR (in other methods it is not allowed). You might add warnings with the macro ELEKTRA_ADD_WARNING. You can also use their pedants that accept a format string as known by printf: ELEKTRA_SET_ERRORF and ELEKTRA_ADD_WARNINGF. Make sure to define and use a macro in the error specification (/src/error/specification) so that you can easily renumber your error/warning codes:

```
number:60
description:Invalid Line encountered
severity:error
macro:NOEOF
module:simpleini
```

Use following include to have the macros for setting the error and adding the warnings available:

```
// using namespace ckdb; // for C++
#include <kdberrors.h>
```

and then you can use:

```
ELEKTRA_SET_VALIDATION_SYNTACTIC_ERROR ( parentKey, "Not at the end of file");
```

Note that you also need to return -1 in the case of error. See individual description of entry points to implement below.

#### Global KeySet Handle

This keyset allows plugins to exchange information with other plugins.
The keyset is initialized by the KDB for all plugins, except for manually created plugins with `elektraPluginOpen()`.
The global keyset is tied to a KDB handle, initialized on `kdbOpen()` and deleted on `kdbClose()`.
Obtain a handle to the global keyset and work with it:

```
KeySet * globalKS = elektraPluginGetGlobalKeySet (plugin);
// now we can read something from the global keyset
// or add something for us or others to read
Key * important = keyNew ("user:/global/myDocKey", KEY_VALUE, "global plugins can see me", KEY_END);
ksAppendKey (globalKS, important);
```

Clean up keys which you do not need any more, to keep the global keyset compact:

```
// clean up parts of the global keyset which we do not need
Key * cutKey = keyNew ("user:/global/myDocKey", KEY_END);
KeySet * notNeeded = ksCut (globalKS, cutKey);
ksDel (notNeeded);
```

#### Further help

Do not hesitate to open an issue if anything is unclear.

### 570.13.2 Macro Definition Documentation

#### 570.13.2.1 ELEKTRA_ADD_WARNING

```
#define ELEKTRA_ADD_WARNING(
                number,
```

```
                key,
                text )
```

Adds a warning in the keys metadata.
Include kdberrors.h to make it work:

```
// using namespace ckdb; // for C++
#include <kdberrors.h>
```

**Parameters**

| | |
|---|---|
| *number* | the warning number from src/error/specification |
| *key* | to write the error to |
| *text* | additional text for the user |

#### 570.13.2.2 ELEKTRA_ADD_WARNINGF

```
#define ELEKTRA_ADD_WARNINGF(
                number,
                key,
                formatstring,
                ...  )
```

Adds a warning in the keys metadata.
Include kdberrors.h to make it work:

```
// using namespace ckdb; // for C++
#include <kdberrors.h>
```

**Parameters**

| | |
|---|---|
| *number* | the warning number from src/error/specification |
| *key* | to write the error to |
| *formatstring* | a format string as in printf |
| *...* | further arguments as in printf |

#### 570.13.2.3 ELEKTRA_PLUGIN_FUNCTION

```
#define ELEKTRA_PLUGIN_FUNCTION(
                function ) ELEKTRA_PLUGIN_FUNCTION2 (ELEKTRA_PLUGIN_NAME_C, function)
```

Declare a plugin's function name suitable for compilation variants (see doc/tutorials).
It can be used in the same way as elektraPluginExport().

**See also**

ELEKTRA_PLUGIN_EXPORT

**Parameters**

| | |
|---|---|
| *plugin* | the name of the plugin |
| *function* | which function it is (open, close, get, set, error, commit) |

#### 570.13.2.4 ELEKTRA_README

```
#define ELEKTRA_README ELEKTRA_README2 (ELEKTRA_PLUGIN_NAME_C)
```

The filename for inclusion of the readme for compilation variants (see doc/tutorials).

**Parameters**

| | |
|---|---|
| *plugin* | the name of the plugin |

#### 570.13.2.5 ELEKTRA_SET_ERROR

```
#define ELEKTRA_SET_ERROR(
              number,
              key,
              text )
```

Sets the error in the keys metadata.
Include kdberrors.h to make it work. Only a single error can be written to the key.

```
// using namespace ckdb; // for C++
#include <kdberrors.h>
```

**Parameters**

| | |
|---|---|
| *number* | the error number from src/error/specification |
| *key* | to write the error to |
| *text* | additional text for the user |

#### 570.13.2.6 ELEKTRA_SET_ERROR_GET

```
#define ELEKTRA_SET_ERROR_GET(
              parentKey )
```

Set error in kdbGet() when opening the file failed.
Assumes that error reason is in `errno`.

**Parameters**

| | |
|---|---|
| *parentKey* | key to append error to |

To use it include:

```
#include <kdbplugin.h>
// using namespace ckdb; // for C++
#include <kdberrors.h>
```

**Returns**

#### 570.13.2.7 ELEKTRA_SET_ERROR_SET

```
#define ELEKTRA_SET_ERROR_SET(
              parentKey )
```

Set error in kdbSet() when opening the file failed.
Assumes that error reason is in `errno`.

**Parameters**

| | |
|---|---|
| *parentKey* | key to append error to |

To use it include:

```
#include <kdbplugin.h>
// using namespace ckdb; // for C++
```

```
#include <kdberrors.h>
```

**Returns**

#### 570.13.2.8 ELEKTRA_SET_ERRORF

```
#define ELEKTRA_SET_ERRORF(
              number,
              key,
              formatstring,
              ...  )
```

Sets the error in the keys metadata.
Include kdberrors.h to make it work. Only a single error can be written to the key.

```
// using namespace ckdb; // for C++
#include <kdberrors.h>
```

**Parameters**

| | |
|---|---|
| *number* | the error number from src/error/specification |
| *key* | to write the error to |
| *formatstring* | a format string as in printf |
| *...* | further arguments as in printf |

### 570.13.3 Function Documentation

#### 570.13.3.1 elektraDocCheckConf()

```
int elektraDocCheckConf (
             Key * errorKey,
             KeySet * conf )
```

Validate plugin configuration at mount time.
During the mount phase the BackendBuilder calls this method, if it is provided by the plugin.
In this method the plugin configuration can be checked for validity or integrity. Missing items can be added to complete the configuration.

**Parameters**

| | |
|---|---|
| *errorKey* | is used to propagate error messages to the caller |
| *conf* | contains the plugin configuration to be validated |

**Return values**

| | |
|---|---|
| *0* | on success: the configuration was OK and has not been changed. |
| *1* | on success: the configuration has been changed and now it is OK. |
| *-1* | on failure: the configuration was not OK and could not be fixed. Set an error using ELEKTRA_SET_ERROR to inform the user what went wrong. Additionally you can add any number of warnings with ELEKTRA_ADD_WARNING. |

#### 570.13.3.2 elektraDocClose()

```
int elektraDocClose (
            Plugin * handle,
            Key * warningsKey )
```

Finalize the plugin.

Called prior to unloading the plugin dynamic module. After this function is called, it is ensured that no functions from your plugin will ever be accessed again.

Make sure to free all memory that your plugin requested at runtime. Also make sure to free what you stored by elektraPluginSetData() before.

So for the Doc plugin we need to:

```
int elektraDocClose (Plugin * handle, Key * warningsKey ELEKTRA_UNUSED)
{
        elektraFree (elektraPluginGetData (handle));
        return 0; /* success */
}
```

After this call, libelektra.so will unload the plugin library, so this is the point to shutdown any affairs with the storage.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of the plugin |
| *warningsKey* | can be used to to add warnings using ELEKTRA_ADD_WARNING (Do not add errors!) |

**Return values**

| | |
|---|---|
| *1* | on success (no other return value currently allowed) |
| *-1* | on problems (only use ELEKTRA_ADD_WARNING, but never set an error). |

**See also**

kdbClose()

elektraPluginGetData(), elektraPluginSetData() and elektraPluginGetConfig()

#### 570.13.3.3 elektraDocCommit()

```
int elektraDocCommit (
            Plugin * handle,
            KeySet * returned,
            Key * parentKey )
```

Make changes to storage final.

Once the content of `returned` has been stored, the changes need to be made final and visible to other users, which is done by this function. After this function has been called, no further changes can be made by elektra↩ PluginSet() functions within this invocation of kdbSet().

The function is called by kdbSet() if the plugin implementing it fulfills the `commit` role.

**Precondition**

The keyset `returned` holds all stored keys which must be made final for this keyset. The keyset is sorted and rewinded.

The `parentKey` is the key which is the ancestor for all other keys in the keyset. The first key of the keyset `returned` has the same keyname. The name of the parentKey marks the mountpoint.

**Postcondition**

the storage changes made by the plugins previously called by kdbSet() will be made final.

**See also**

kdbSet() for caller.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of the plugin |
| *returned* | contains a keyset with relevant keys |
| *parentKey* | contains the location of the relevant keys within the key database. |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on success without any changes |
| *-1* | on failure. The cause of the error needs to be entered into parentKey. The error can be specified using ELEKTRA_SET_ERROR. ELEKTRA_ADD_WARNING can be used to add warnings for the user. |

#### 570.13.3.4 elektraDocError()

```
int elektraDocError (
            Plugin * handle,
            KeySet * returned,
            Key * parentKey )
```

Rollback in case of errors.
First for all plugins elektraDocSet() will be called. If any plugin had problems before the commit (done by the resolver plugin), we can safely rollback our changes.
This method is rarely used by plugins, it is mainly used for resolvers (to implement rollback) or by logging plugins. It is not needed for storage plugins, because they only operate on temporary files created by the resolver.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of the plugin |
| *returned* | contains a keyset with relevant keys |
| *parentKey* | contains the information where to set the keys. can be used to add warnings with the macro ELEKTRA_ADD_WARNING, but do not add errors! |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on success with no action |
| *-1* | on failure (you can add warnings, but we are already in an error state, so do not set the error). |

#### 570.13.3.5 elektraDocGet()

```
int elektraDocGet (
            Plugin * handle,
            KeySet * returned,
            Key * parentKey )
```

Get data from storage to application.
Retrieve information from a permanent storage to construct a keyset.

### 570.13.4 Introduction

The elektraDocGet() function handle everything related to receiving keys.

#### 570.13.4.1 Contract Handling

The contract is a keyset that needs to be returned if the parentKey is system:/elektra/modules/yourpluginname.
Which keys and their meaning is specified in doc/CONTRACT.ini
Here is an example for our doc plugin:

```
int elektraDocGet (Plugin * plugin ELEKTRA_UNUSED, KeySet * returned, Key * parentKey)
{
        if (!strcmp (keyName (parentKey), "system:/elektra/modules/doc"))
        {
                KeySet * contract =
                        ksNew (30, keyNew ("system:/elektra/modules/doc", KEY_VALUE, "doc plugin waits for
       your orders", KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports", KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/open", KEY_FUNC, elektraDocOpen,
       KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/close", KEY_FUNC,
       elektraDocClose, KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/get", KEY_FUNC, elektraDocGet,
       KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/set", KEY_FUNC, elektraDocSet,
       KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/commit", KEY_FUNC,
       elektraDocCommit, KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/error", KEY_FUNC,
       elektraDocError, KEY_END),
                                keyNew ("system:/elektra/modules/doc/exports/checkconf", KEY_FUNC,
       elektraDocCheckConf, KEY_END),
#include ELEKTRA_README
                                keyNew ("system:/elektra/modules/doc/infos/version", KEY_VALUE,
       PLUGINVERSION, KEY_END), KS_END);
                ksAppend (returned, contract);
                ksDel (contract);
                return 1; /* success */
        }
```

Some clauses of the contract, especially the description of the plugin can be done more conveniently directly in a README.md that is included by ELEKTRA_README.

#### 570.13.4.2 Storage Plugins

For storage plugins the filename is written in the value of the parentKey. So the first task of the plugin is to open that file. Then it should parse its content and construct a keyset with all information of that file.
You need to be able to reconstruct the same file with the information of the keyset. So be sure to copy all comments, whitespaces and so on into some metadata of the keys. Otherwise the information is lost after writing the file the next time.
Now lets look at an example how the typical elektraDocGet() might be implemented. To explain we introduce some pseudo functions which do all the work with the storage (which is of course 90% of the work for a real plugin):

- parse_key will parse a key and a value from an open file handle

The typical loop for a storage plugin will be like:

```
        FILE * fp = fopen (keyString (parentKey), "r");
        char * key = 0;
        char * value = 0;
        while (parseKey (fp, &key, &value) >= 1)
        {
                Key * read = keyNew (keyName (parentKey), KEY_END);
                if (keyAddName (read, key) == -1)
                {
                        ELEKTRA_ADD_VALIDATION_SYNTACTIC_WARNINGF (parentKey, "Key name %s is not valid,
       discarding key", key);
                        keyDel (read);
                        continue;
                }
                keySetString (read, value);
                ksAppendKey (returned, read);
        }
        if (feof (fp) == 0)
        {
                fclose (fp);
                ELEKTRA_SET_VALIDATION_SYNTACTIC_ERROR (parentKey, "Invalid line encountered: not at the end
       of file");
                return -1;
        }
        fclose (fp);
```

When opening files, make sure to use the macros ELEKTRA_SET_ERROR_SET and ELEKTRA_SET_ERROR_GET for errors, as shown here:

```
        FILE * fp = fopen (keyString (parentKey), "w");
        if (!fp)
```

```
        {
                ELEKTRA_SET_ERROR_SET (parentKey);
                return -1;
        }
```

#### 570.13.4.3 Filter Plugins

For filter plugins the actual task is rather unspecified. You basically can do anything with the keyset. To get roundtrip properties you might want to undo any changes you did in elektraDocSet().
The pseudo functions (which do the real work) are:

- do_action() which processes every key in this filter

```
        for (elektraCursor it = 0; it < ksGetSize (returned); ++it)
        {
                Key * k = ksAtCursor (returned, it);
                doAction (k);
        }
        return 1; // success
}
```

**Precondition**

The caller kdbGet() will make sure before you are called that the parentKey:

- is a valid key (means that it is a system or user key).
- is your mountpoint and that your plugin is responsible for it.

and that the returned:

- is a valid keyset.
- your plugin is only called when needed (e.g. only if file was modified)
- has `all` keys related to your plugin.
- contains only valid keys below (see keyIsBelow()) your parentKey.
- is in a sorted order (given implicit by KeySet)

and that the handle:

- is valid for your plugin.
- that elektraPluginGetData() contains the same handle for lifetime of your plugin.

The caller kdbGet() will make sure that afterwards you were called, that:

- other plugins below your plugin will be called again recursively.
- that all keys are merged to one keyset the user gets
- that all keys (that should not be removed) are passed to kdbSet() if writing to disc is needed.

**Invariant**

There are no global variables and elektraPluginSetData() stores all information. The handle is to be guaranteed to be the same if it is the same plugin.

**Postcondition**

The keyset `returned` has the `parentKey` and all keys below (keyIsBelow()) with all information from the storage. Make sure to return all keys, all directories and also all hidden keys. If some of them are not wished, the caller kdbGet() will drop these keys with additional plugins.

**Updating**

To get all keys out of the storage over and over again can be very inefficient. You might know a more efficient method to know if the key needs update or not, e.g. by stating it or by an external time stamp info. For file storage plugins this is automatically done for you by the resolver. For other types (e.g. databases) you need to implement your own resolver doing this.

**See also**

kdbGet() for caller.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *returned* | contains a keyset where the function need to append the keys got from the storage. There might be also some keys inside it, see conditions. You may use them to support efficient updating of keys, see updating. |
| *parentKey* | contains the information below which key the keys should be gotten. |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | when nothing was to do |
| *-1* | on failure, the current key in returned shows the position. use ELEKTRA_SET_ERROR of kdberrors.h to define the error code. You additionally can add as many warnings as you would like to add. |

#### 570.13.4.4 elektraDocOpen()

```
int elektraDocOpen (
            Plugin * handle,
            Key * warningsKey )
```

Initialize data for the plugin.
This is the first method called after dynamically loading this plugin. It is guaranteed, that this method will be called before any other method.
This method is responsible for:

- plugin's specific configuration gathering
- initialization of all plugin's internal structs
- initial setup of all I/O details such as opening a file, connecting to a database, setup connection to a server, iff this cannot be done per invocation in elektraDocGet() and elektraDocSet().

You may also read the configuration you can get with elektraPluginGetConfig() and transform it into other structures used by your plugin.

**Note**

> The plugin must not have any global variables. If you have one Elektra will not be threadsafe. Do not assume that your plugin will be opened only once or will not be reopened at a later time.

Instead of global variables the methods elektraPluginGetData() and elektraPluginSetData() exist to store and get any information related to your plugin.
The correct substitute for global variables will be:

```
typedef struct
{
        int global;
} GlobalData;
```

and then initialize it using:

```
int elektraDocOpen (Plugin * handle, Key * warningsKey ELEKTRA_UNUSED)
{
        GlobalData * data;
        KeySet * config = elektraPluginGetConfig (handle);
        Key * kg = ksLookupByName (config, "/global", 0);
        data = elektraMalloc (sizeof (GlobalData));
        data->global = 0;
        if (kg) data->global = atoi (keyString (kg));
        elektraPluginSetData (handle, data);
```

Make sure to free everything you allocate within elektraDocClose().
If your plugin has no useful way to startup without config, the module loader would not be able to load the module. We need, however, to still load the plugin to get the contract.

To solve that problem the module loader adds the configuration key /module. Even if your plugin is basically not able to startup successfully, it should still provide a fallback when /module is present, so that elektraDocGet() on system:/elektra/modules can be called successfully later on.

```
        if (ksLookupByName (config, "/module", 0))
        {
                return 0;
        }
        // do some setup that will fail without configuration
```

Note that for plugins where the contract will be altered based on configuration this specific configuration should be considered. In fact the idea of /module is to get the correct contract.

#### Return values

| | |
|---|---|
| *-1* | on error, your plugin will be removed then and the missing plugin added instead. Use ELEKTRA_ADD_WARNING to indicate the problem. The system will automatically add the information that the plugin was removed, so you do not need the user give that information. |
| *1* | on success |

#### Parameters

| | |
|---|---|
| *handle* | contains internal information of the plugin |
| *warningsKey* | can be used to add warnings with the macro ELEKTRA_ADD_WARNING (Do not add errors!) |

#### See also

elektraPluginGetData(), elektraPluginSetData() and elektraPluginGetConfig()

elektraDocClose()

### 570.13.4.5 elektraDocSet()

```
int elektraDocSet (
              Plugin * handle,
              KeySet * returned,
              Key * parentKey )
```

Set data from application to storage.

This function does everything related to set and remove keys in a plugin. There is only one function for that purpose to make implementation of file based plugins much easier.

The keyset `returned` was filled in with information from the application using elektra and the task of this function is to store it in a permanent way so that a subsequent call of elektraPluginGet() can rebuild the keyset as it was before. See the live cycle to understand:

```
static void usercode (KDB * handle, KeySet * keyset, Key * key)
{
        // some more user code
        keySetString (key, "mycomment"); // the user changes the key
        ksAppendKey (keyset, key);    // append the key to the keyset
        kdbSet (handle, keyset, 0);        // and syncs it to disc
}
// so now kdbSet is called
int elektraKdbSet (KDB * handle, KeySet * keyset, Key * parentKey)
{
        int ret = 0;
        // find appropriate plugin and then call it:
        Plugin * plugin = findPlugin (handle);
        ret = elektraDocSet (plugin, keyset, parentKey);
        // the keyset with the key (and others for this plugin)
        // will be passed to this function
        return ret;
}
// so now elektraPluginSet(), which is the function described here,
// is called:
int elektraPluginSet (Plugin * plugin ELEKTRA_UNUSED, KeySet * returned, Key * parentKey ELEKTRA_UNUSED)
{
        // the task of elektraPluginSet is now to store the keys
        for (elektraCursor it = 0; it < ksGetSize (returned); ++it)
        {
```

```
                Key * k = ksAtCursor (returned, it);
                saveToDisc (k);
        }
        return 1; /* success */
}
```

Of course all information of every key in the keyset `returned` need to be stored permanently. So this specification needs to give an exhaustive list of information present in a key.

**Precondition**

The keyset `returned` holds all keys which must be saved permanently for this keyset. The keyset is sorted and rewinded.

The `parentKey` is the key which is the ancestor for all other keys in the keyset. The first key of the keyset `returned` has the same keyname. The name of the parentKey marks the mountpoint. The string of the parentKey is the filename to write to.

Make sure to set all keys, all directories and also all hidden keys. If some of them are not wished, the caller kdbSet() and plugins will sort them out.

**Invariant**

There are no global variables, but instead elektraPluginGetData() will be used. The handle is the same when it is the same plugin.

**Postcondition**

The information of the keyset `returned` is stored permanently.

**See also**

kdbSet() for caller.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of the plugin |
| *returned* | contains a keyset with relevant keys |
| *parentKey* | contains the information where to set the keys (name is mountpoint your plugin is mounted, string is the file to write to) |

**Returns**

When everything works gracefully return the number of keys you set. The cursor position and the keys remaining in the keyset are not important.

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on success with no changed key in database |
| *-1* | on failure. The cause of the error needs to be added in parentKey Set an error using ELEKTRA_SET_ERROR to inform the user what went wrong. Additionally you can add any number of warnings with ELEKTRA_ADD_WARNING. |

### 570.13.4.6 elektraPluginExport()

```
Plugin* elektraPluginExport (
            const char * pluginName,
             ...  )
```

Allows one to Export Methods for a Plugin.
This function must be called within ELEKTRA_PLUGIN_EXPORT. It define the plugin's methods that will be exported.
All KDB methods implemented by the plugin basically could have random names (convention is elektraName∗), except ELEKTRA_PLUGIN_EXPORT.
This is the single symbol that will be looked up when loading the plugin, and the first method of the backend implementation that will be called.
You need to use a macro so that both dynamic and static loading of the plugin works. For example for the doc plugin:

```
Plugin * ELEKTRA_PLUGIN_EXPORT
{
        // clang-format off
        return elektraPluginExport(DOC_PLUGIN_NAME,
                ELEKTRA_PLUGIN_OPEN, &elektraDocOpen,
                ELEKTRA_PLUGIN_CLOSE,       &elektraDocClose,
                ELEKTRA_PLUGIN_GET,   &elektraDocGet,
                ELEKTRA_PLUGIN_SET,   &elektraDocSet,
                ELEKTRA_PLUGIN_ERROR,       &elektraDocError,
                ELEKTRA_PLUGIN_COMMIT,     &elektraDocCommit,
                ELEKTRA_PLUGIN_END);
}
```

The first parameter is the name of the plugin. Then every plugin should have: ELEKTRA_PLUGIN_OPEN, ELEKTRA_PLUGIN_CLOSE, ELEKTRA_PLUGIN_GET, ELEKTRA_PLUGIN_SET and optionally ELEKTRA↩ _PLUGIN_ERROR and ELEKTRA_PLUGIN_COMMIT.
The list is terminated with ELEKTRA_PLUGIN_END.
You must use static "char arrays" in a read only segment. Don't allocate storage, it won't be freed.

**Parameters**

| | |
|---|---|
| *pluginName* | the name of this plugin |

**Returns**

an object that contains all plugin information needed by libelektra.so

#### 570.13.4.7 elektraPluginGetConfig()

```
KeySet* elektraPluginGetConfig (
             Plugin * handle )
```

Returns the configuration of that plugin.

- The user:/ config holds plugin specific configuration
- The system:/ config holds backend specific configuration

So prefer cascading lookups to honor both.

**Parameters**

| | |
|---|---|
| *handle* | a pointer to the plugin |

**Returns**

keyset to the configuration for that plugin

#### 570.13.4.8 elektraPluginGetData()

```
void* elektraPluginGetData (
             Plugin * plugin )
```

Get a pointer to the plugin specific data stored before.
If elektraPluginSetData() was not called earlier, NULL will be returned.
This data is private to one instance of a plugin.

**See also**

elektraPluginSetData

**Parameters**

| | |
|---|---|
| *plugin* | a pointer to the plugin |

**Returns**

a pointer to the data

#### 570.13.4.9 elektraPluginGetGlobalKeySet()

```
KeySet* elektraPluginGetGlobalKeySet (
            Plugin * plugin )
```

Get a pointer to the global keyset.
Initialized for all plugins by the KDB, except for manually created plugins with `elektraPluginOpen()`. The global keyset is tied to a KDB handle, initialized on `kdbOpen()` and deleted on `kdbClose()`.
Plugins using this keyset are responsible for cleaning up their parts of the keyset which they do not need any more.

**Parameters**

| | |
|---|---|
| *plugin* | a pointer to the plugin |

**Returns**

a pointer to the global keyset

#### 570.13.4.10 elektraPluginSetData()

```
void elektraPluginSetData (
            Plugin * plugin,
            void * data )
```

Store a pointer to plugin specific data.
This data is private to one instance of a plugin.

**See also**

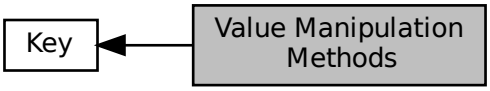
elektraPluginGetData

**Parameters**

| | |
|---|---|
| *plugin* | a pointer to the plugin |
| *data* | the pointer to the data |

## 570.14 Value Manipulation Methods

Methods to do various operations on Key values.
Collaboration diagram for Value Manipulation Methods:




### Functions

- const void * keyValue (const Key *key)

  *Return a pointer to the real internal `key` value.*
- const char * keyString (const Key *key)

  *Get a pointer to the c-string representing the value.*
- ssize_t keyGetValueSize (const Key *key)

  *Returns the number of bytes needed to store the key value, including the NULL terminator.*
- ssize_t keyGetString (const Key *key, char *returnedString, size_t maxSize)

  *Copy the string value of a Key into `returnedString`.*
- ssize_t keySetString (Key *key, const char *newStringValue)

  *Set the value for `key` as `newStringValue`.*
- ssize_t keyGetBinary (const Key *key, void *returnedBinary, size_t maxSize)

  *Copy the binary value of a Key into `returnedBinary`.*
- ssize_t keySetBinary (Key *key, const void *newBinary, size_t dataSize)

  *Set the value of a Key to the binary value `newBinary`.*

### 570.14.1 Detailed Description

Methods to do various operations on Key values.
A key can contain a value in different format. The most likely situation is, that the value is interpreted as text. Use keyGetString() for that. You can save any Unicode Symbols and Elektra will take care that you get the same back, independent of your current environment.
In some situations this idea fails. When you need exactly the same value back without any interpretation of the characters, there is keySetBinary(). If you use that, its very likely that your Configuration is not according to the standard. Also for Numbers, Booleans and Date you should use keyGetString(). To do so, you might use strtod() strtol() and then atol() or atof() to convert back.
To use them:

```
#include <kdb.h>
```

### 570.14.2 Function Documentation

#### 570.14.2.1 keyGetBinary()

```
ssize_t keyGetBinary (
            const Key * key,
            void * returnedBinary,
            size_t maxSize )
```

Copy the binary value of a Key into `returnedBinary`.
If the type is not binary -1 will be returned.
When the binary data is empty (this is not the same as ""!) 0 will be returned and `returnedBinary` will not be changed.
For string values see keyGetString() and keyIsString().
When `returnedBinary` is too small to hold the data (maximum size is given by maxSize), the returnedBinary will not be changed and -1 is returned.

**Example:**

```
Key *key = keyNew ("user:/keyname", KEY_BINARY, KEY_END);
char buffer[300];
if (keyGetBinary(key,buffer,sizeof(buffer)) == -1)
{
        // handle error
}
```

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the binary value from |
| *returnedBinary* | pre-allocated memory to store a copy of the Key's value |
| *maxSize* | number of bytes of pre-allocated memory in `returnedBinary` |

**Returns**

the number of bytes copied to `returnedBinary`

**Return values**

| | |
|---|---|
| *0* | if the binary is empty |
| *-1* | on NULL pointers |
| *-1* | if maxSize is 0, too small for the value or larger than SSIZE_MAX |
| *-1* | if the Key's value is a string |

**Since**

1.0.0

**See also**

keyValue() for getting a raw pointer to the Key's value

keyGetValueSize() for getting the size of the Key's value

keySetBinary() for setting the binary value of a Key

keyIsBinary() for checking whether a Key's value is binary

keyGetString(), keySetString() for working with string values

### 570.14.2.2 keyGetString()

```
ssize_t keyGetString (
            const Key * key,
            char * returnedString,
            size_t maxSize )
```

Copy the string value of a Key into `returnedString`.
When there is no value inside the string, 1 will be returned and the returnedString will be empty ("") to avoid programming errors where old strings are shown to the user.

**Example:**

```
Key *key = keyNew ("user:/keyname", KEY_END);
char buffer[300];
if (keyGetString(key,buffer,sizeof(buffer)) == -1)
{
        // handle error
} else {
        printf ("buffer: %s\n", buffer);
}
```

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the string from |
| *returnedString* | pre-allocated memory to store a copy of the Key's value |
| *maxSize* | number of bytes of allocated memory in `returnedString` |

**Returns**

the number of bytes actually copied to `returnedString`, including final NULL

**Return values**

| | |
|---|---|
| *1* | if the string is empty |
| *-1* | on any NULL pointers |
| *-1* | if the Key's value is binary |
| *-1* | maxSize is 0, too small for the string or larger than SSIZE_MAX |

**Since**

1.0.0

**See also**

keyGetValueSize() for getting the size of the Key's value

keyValue() for getting a raw pointer to the Key's value

keyString() for getting a raw char pointer to the Key's value

keyGetBinary(), keyIsBinary() for working with binary data

### 570.14.2.3 keyGetValueSize()

```
ssize_t keyGetValueSize (
            const Key * key )
```

Returns the number of bytes needed to store the key value, including the NULL terminator.
It returns the correct size, independent of the Key Type. If the value is binary there might be '\0' values in it.
For an empty string you need one byte to store the ending NULL. For that reason 1 is returned. This is not true for binary data, so 0 will be returned.
A binary key has no '\0' termination. String types are null-terminated, and the terminator will be considered for the length.
This method can be used with elektraMalloc() before keyGetString() or keyGetBinary() is called.

```
char *buffer;
buffer = elektraMalloc (keyGetValueSize (key));
// use this buffer to store the value (binary or string)
// pass keyGetValueSize (key) for maxSize
```

**Postcondition**

returns the exact amount of bytes needed to store `key's` value (including NULL terminators)

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the size of the value from |

**Returns**

the number of bytes needed to store the Key's value

**Return values**

| | |
|---|---|
| *1* | when there is no data and type is a string |
| *0* | when there is no data and type is binary |
| *-1* | on null pointer |

**Since**

1.0.0

**See also**

keyGetString() for getting the Key's value as a string

keyGetBinary() for getting the Key's value as a binary

keyValue() for getting a pointer to the Key's value

### 570.14.2.4 keySetBinary()

```
ssize_t keySetBinary (
            Key * key,
            const void * newBinary,
            size_t dataSize )
```

Set the value of a Key to the binary value `newBinary`.
A private copy of `newBinary` will be allocated and saved inside `key`, so the parameter can be deallocated after the call.
Binary values might be encoded in another way than string values depending on the plugin. Typically character encodings should not take place on binary data. Consider using a string Key instead, if encoding should occur.
When `newBinary` is a NULL pointer the value will be freed and 0 will be returned.
Read-only keys will stay unchanged after calling this function.

**Note**

The metadata "binary" will be set to mark that the key is binary from now on. When the Key is already binary the metadata won't be changed. This will only happen in the successful case, but not when -1 is returned.

**Precondition**

`dataSize` matches the length of `newBinary`

`newBinary` is not NULL and `dataSize` > 0

`key` is not read-only

**Postcondition**

`key`'s value set exactly to the data in `newBinary`

"binary" key set in `key`'s metadata

**Parameters**

| | |
|---|---|
| *key* | the Key object where the value should be set |
| *newBinary* | a pointer to any binary data or NULL (to clear the stored value) |
| *dataSize* | number of bytes to copy from `newBinary` |

**Returns**

the number of bytes actually copied to internal struct storage

**Return values**

| | |
|---|---|
| *0* | when the internal binary was freed and is now a null pointer |
| *-1* | if `key` is NULL |
| *-1* | when `dataSize` is 0 (and newBinary not NULL) or larger than SSIZE_MAX |
| *-1* | if `key` is read-only |

**Since**

1.0.0

**See also**

keyGetBinary() for getting a Key's value as binary

keyIsBinary() to check if the Key's value is binary

keyGetString() and keySetString() for working with string values

### 570.14.2.5 keySetString()

```
ssize_t keySetString (
            Key * key,
            const char * newStringValue )
```

Set the value for `key` as `newStringValue`.
The function will allocate and save a private copy of `newStringValue`, so the parameter can be freed after the call.
String values will be saved in backend storage in UTF-8 universal encoding, regardless of the program's current encoding (if the iconv plugin is available).

**Precondition**

`newStringValue` is a NULL terminated string

**Postcondition**

Value of the Key is set to the UTF-8 encoded value of `newStringValue`

Metakey `meta:/binary` is cleared

**Parameters**

| | |
|---|---|
| *key* | the Key for which to set the string value |
| *newStringValue* | NULL-terminated string to be set as `key's` value |

**Returns**

the number of bytes actually saved in private struct including final NULL

**Return values**

| | |
|---|---|
| *1* | if `newStringValue` is a NULL pointer, this will make the string empty (string only containing null termination) |
| *-1* | if `key` is a NULL pointer |

**Since**

1.0.0

**See also**

keyString() for getting a pointer to the Key's value

keyGetString() for getting a copy of the Key's value

keySetBinary() for setting binary data

### 570.14.2.6 keyString()

```
const char* keyString (
            const Key * key )
```

Get a pointer to the c-string representing the value.
Will return "(null)" on null pointers. Will return "(binary)" on binary data not ended with a null byte.

**Note**

You must not change or delete the returned value. Use the respective functions for that (keySetString(), keyGetString())

It is not checked if it is actually a string, only if it terminates for security reasons.

**Postcondition**

returned pointer points to the real internal value

value is NULL terminated

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the string value from |

**Returns**

pointer to the c-string representing the Key's value

**Return values**

| | |
|---|---|
| *""* | if no data found |
| *(null)* | on null Key |
| *(binary)* | on binary Key |

**Since**

1.0.0

**See also**

keyGetString() for getting a copy of the Key's value as string

keyGetBinary() for getting a copy of the Key's value as binary

keyValue() for getting a pointer to the Key's value as binary

#### 570.14.2.7 keyValue()

```
const void* keyValue (
            const Key * key )
```

Return a pointer to the real internal `key` value.
This is a much more efficient version of keyGetString() keyGetBinary(). You should use it if you are responsible enough to not mess up things. You are not allowed to modify anything in the returned string. If you need a copy of the Value, consider to use keyGetString() or keyGetBinary() instead.

### 570.14.3 String Handling

If `key` is string (keyIsString()), you may cast the returned as a "`char *`" because you'll get a NULL terminated regular string.
keyValue() returns "" in string mode when there is no value. The reason is

```
key=keyNew(0);
keySetString(key,"");
keyValue(key); // you would expect "" here
keyDel(key);
```

### 570.14.4 Binary Data Handling

If the data is binary, the size of the value must be determined by keyGetValueSize(), any strlen() operations are not suitable to determine the size.
keyValue() returns 0 in binary mode when there is no value. The reason is

```
key=keyNew(0);
keySetBinary(key, 0, 0);
keyValue(key); // you would expect 0 here
keySetBinary(key,"", 1);
keyValue(key); // you would expect "" (a pointer to '\0') here
int i=23;
keySetBinary(key, (void*)&i, 4);
(int*)keyValue(key); // you would expect a pointer to (int)23 here
keyDel(key);
```

**Note**

Note that the Key structure keeps its own size field that is calculated by library internal calls, so to avoid inconsistencies, you must never use the pointer returned by keyValue() method to set a new value. Use keySetString() or keySetBinary() instead.

**Warning**

Binary keys will return a NULL pointer when there is no data in contrast to keyName(), keyBaseName() and keyComment(). For string value the behaviour is the same.

**Example:**

```
KDB *handle = kdbOpen();
KeySet *ks=ksNew(0, KS_END);
kdbGetByName(handle,ks,"system:/sw/my",KDB_O_SORT|KDB_O_RECURSIVE);
 for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
 {
        Key * current = ksAtCursor (ks, it);
        size_t size=0;
        if (keyIsBinary(current)) {
                size=keyGetValueSize(current);
                printf("Key %s has a value of size %d bytes. Value: <BINARY>\nComment: %s",
                        keyName(current),
                        size,
                        keyComment(current));
        } else {
                size=elektraStrLen((char *)keyValue(current));
                printf("Key %s has a value of size %d bytes. Value: %s\nComment: %s",
                        keyName(current),
                        size,
                        (char *)keyValue(current),
                        keyComment(current));
        }
}
ksDel (ks);
kdbClose (handle);
```

**Precondition**

`key` is not NULL and has stored data

**Postcondition**

returned pointer points to the stored internal value

if the value is a string, the value is NULL terminated

**Parameters**

| | |
|---|---|
| *key* | the Key from which to get the value |

**Returns**

a pointer to the Key's internal value

**Return values**

| | |
|---|---|
| *""* | when there is no value and Key is not binary |
| *0* | where there is no value and Key is binary |
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

keyGetValueSize() to get the size of the Key's value

keyGetString() for getting the Key's value as string

keyGetBinary() for getting the Key's value as binary

# Chapter 571

# Namespace Documentation

## 571.1 kdb Namespace Reference

This is the main namespace for the C++ binding and libraries.

### Namespaces

- tools

  *This namespace is for the libtool library.*

### Classes

- class ElektraDiff

  *This class is a wrapper around the ElektraDiff C struct.*
- class KDB

  *Constructs a class KDB.*
- class Context

  *Provides a context for configuration.*
- class ThreadSubject

  *Subject from Observer pattern for ThreadContext.*
- struct PerContext

  *A data structure that is stored by context inside the Coordinator.*
- class Coordinator

  *Thread safe coordination of ThreadContext per Threads.*
- class none_t

  *This type is being used as bottom type that always fails.*
- class Layer

  *Base class for all layers.*
- class Wrapped

  *Everything implementing this interface can be used as layer.*
- class ValueObserver

  *Base class for values to be observed.*
- struct Command

  *Used by contexts for callbacks (to run code using a mutex).*
- class DefaultGetPolicy

  *Implements lookup with spec.*
- class DefaultSetPolicy

  *Implements creating user:/ key when key is not found.*
- class GetPolicyIs

  *Needed by the user to set one of the policies.*

- class SetPolicyIs

  *Needed by the user to set one of the policies.*
- class ContextPolicyIs

  *Needed by the user to set one of the policies.*
- class WritePolicyIs

  *Needed by the user to set one of the policies.*
- class ObserverPolicyIs

  *Needed by the user to set one of the policies.*
- class LockPolicyIs

  *Needed by the user to set one of the policies.*
- class Key

  *Key is an essential class that encapsulates key name , value and metainfo .*
- class NameIterator

  *For C++ forward Iteration over Names.*
- class NameReverseIterator

  *For C++ reverse Iteration over Names.*
- struct VaAlloc

  *Needed to avoid constructor ambiguity.*
- class KeySet

  *A keyset holds together a set of keys.*
- class KeySetIterator

  *For C++ forward Iteration over KeySets.*
- class KeySetReverseIterator

  *For C++ reverse Iteration over KeySets.*

## Typedefs

- typedef std::unordered_map< std::string, LayerAction > LayerMap

  *A vector of layers.*

## Functions

- int goptsContract (kdb::KeySet &contract, int argc, const char ∗const ∗argv, const char ∗const ∗envp, const kdb::Key &parentKey, kdb::KeySet &goptsConfig)
- int goptsContract (kdb::KeySet &contract, const std::string &argsString, const std::string &envString, const kdb::Key &parentKey, kdb::KeySet &goptsConfig)

  *Prefer to use goptsContract with argc, argv and envp if possible (especially when you are calling this in your main function)*
- int goptsContract (kdb::KeySet &contract, const std::vector< std::string > &args, const std::vector< std::string > &env, const kdb::Key &parentKey, kdb::KeySet &goptsConfig)

  *Prefer to use goptsContract with argc, argv and envp if possible (especially when you are calling this in your main function)*
- bool operator< (ValueObserver const &lhs, ValueObserver const &rhs)

  *Needed to put a ValueObserver in a map.*
- std::ostream & operator<< (std::ostream &os, kdb::Key const &k)

  *Stream the name of a key.*
- std::istream & operator>> (std::istream &is, kdb::Key &k)

  *Reads a line with a keys name.*
- std::ostream & operator<< (std::ostream &os, kdb::KeySet const &cks)

  *Outputs line per line the keynames.*
- std::istream & operator>> (std::istream &is, kdb::KeySet &ks)

  *Reads line per line key names and appends those keys to ks.*

### 571.1.1 Detailed Description

This is the main namespace for the C++ binding and libraries.
Classes or Functions directly below this namespace are header-only. Sub namespaces are intended for libraries and you need to link the library if you want to use them.

**See also**

- kdb::tools

### 571.1.2 Function Documentation

#### 571.1.2.1 goptsContract() [1/3]

```
int kdb::goptsContract (
            kdb::KeySet & contract,
            const std::string & argsString,
            const std::string & envString,
            const kdb::Key & parentKey,
            kdb::KeySet & goptsConfig )  [inline]
```

Prefer to use goptsContract with argc, argv and envp if possible (especially when you are calling this in your main function)
This function mainly exists for use from language bindings.

**See also**

elektraGOptsContractFromStrings

#### 571.1.2.2 goptsContract() [2/3]

```
int kdb::goptsContract (
            kdb::KeySet & contract,
            const std::vector< std::string > & args,
            const std::vector< std::string > & env,
            const kdb::Key & parentKey,
            kdb::KeySet & goptsConfig )  [inline]
```

Prefer to use goptsContract with argc, argv and envp if possible (especially when you are calling this in your main function)
This function mainly exists for use from language bindings.

**See also**

elektraGOptsContractFromStrings

#### 571.1.2.3 goptsContract() [3/3]

```
int kdb::goptsContract (
            kdb::KeySet & contract,
            int argc,
            const char *const * argv,
            const char *const * envp,
            const kdb::Key & parentKey,
            kdb::KeySet & goptsConfig )  [inline]
```

**See also**

elektraGOptsContract

#### 571.1.2.4 operator<()

```
bool kdb::operator< (
            ValueObserver const & lhs,
            ValueObserver const & rhs )  [inline]
```

Needed to put a ValueObserver in a map.

**Returns**

Comparison result

#### 571.1.2.5 operator<<() [1/2]

```
std::ostream& kdb::operator<< (
            std::ostream & os,
            kdb::Key const & k )  [inline]
```

Stream the name of a key.
Use setf(std::ios_base::showbase) on the stream if you want to also output all metakeys (warning, cannot be parsed back!)
If you also want to stream the value, use the plugin framework.

**Parameters**

| | |
|---|---|
| *os* | the stream to write to |
| *k* | the key which name should be streamed |

**Returns**

the stream

#### 571.1.2.6 operator<<() [2/2]

```
std::ostream& kdb::operator<< (
            std::ostream & os,
            kdb::KeySet const & cks )  [inline]
```

Outputs line per line the keynames.
To output values you should use the plugin framework.

**Parameters**

| | |
|---|---|
| *os* | the stream to write to |
| *cks* | the keyset which should be streamed |

Use unsetf(std::ios_base::skipws) or use noskipws iomanip on the stream if you want a null terminated sequence of key names.
Use setf(std::ios_base::unitbuf) on the stream if you want to flush the buffer after each key.

**Returns**

the stream

#### 571.1.2.7 operator>>() [1/2]

```
std::istream& kdb::operator>> (
            std::istream & is,
            kdb::Key & k )  [inline]
```

Reads a line with a keys name.

**Parameters**

| | |
|---|---|
| *is* | the stream to read from |
| *k* | the key whose name will be set |

Use unsetf(std::ios_base::skipws) on the stream if the keyname is terminated with an null character and not a newline.

**Returns**

the stream

#### 571.1.2.8 operator>>() [2/2]

```
std::istream& kdb::operator>> (
            std::istream & is,
            kdb::KeySet & ks )  [inline]
```

Reads line per line key names and appends those keys to ks.
To input values you need to use the plugin framework.

**Parameters**

| | |
|---|---|
| *is* | the stream to read from |
| *ks* | the keyset to append to |

**Returns**

the stream

## 571.2 kdb::tools Namespace Reference

This namespace is for the libtool library.

### Classes

- class BackendInterface

  *Minimal interface to add plugins.*
- class SerializeInterface

  *Interface to serialize a backend.*
- class MountBackendInterface

  *Interface to work with mountpoints (backends) for factory.*
- class Backend

  *A low-level representation of the backend (= set of plugins) that can be mounted.*
- class BackendFactory

  *Factory for MountBackendInterface.*
- class PluginAdder

  *Adds plugins in a generic map.*
- class GlobalPlugins

  *Low level representation of global plugins.*
- class ImportExportBackend

  *Backend for import/export functionality.*
- class BackendBuilderInit

*Used as argument of constructor of ∗BackendBuilder.*

- class BackendBuilder

  *Highlevel interface to build a backend.*
- class GlobalPluginsBuilder

  *Build global plugins.*
- class MountBackendBuilder

  *High-level functionality to build a mountpoint.*
- struct BackendInfo

  *Info about a backend.*
- class Backends

  *Allows one to list backends.*
- class Modules

  *Allows one to load plugins.*
- class Plugin

  *This is a C++ representation of a plugin.*
- class PluginDatabase

  *Loads all plugins and allows us to query them.*
- class ModulesPluginDatabase

  *A plugin database that works with installed modules.*
- class MockPluginDatabase

  *A plugin database that works with added fake data.*
- class Plugins

  *A collection of plugins (either get, set or error)*
- class GetPlugins

  *Plugins to get configuration.*
- class SetPlugins

  *Plugins to set configuration.*
- class ErrorPlugins

  *Plugins to handle errors during configuration access.*
- class CommitPlugins

  *Plugins to handle errors during configuration access.*
- class PluginSpec

  *Specifies a plugin by its name and configuration.*
- struct PluginSpecHash

  *Only to be used with PluginSpecName!*
- class SpecBackendBuilder

  *Build individual backend while reading specification.*
- class SpecReader

  *Highlevel interface to build a backend from specification.*
- struct ToolException

  *All exceptions from the elektratools library are derived from this exception.*

## Functions

- std::ostream & operator<< (std::ostream &os, Backend const &b)

  *Prints the current status.*
- kdb::KeySet parsePluginArguments (std::string const &pluginArguments, std::string const &basepath)

  *Parse a string containing information to create a KeySet.*
- PluginSpecVector parseArguments (std::string const &cmdline)

  *Parse a complete commandline.*

- template<typename Iterator >
  PluginSpecVector parseArguments (Iterator first, Iterator last)

  *Parse a complete commandline that is already tokenized in pluginname pluginconfig.*
- std::ostream & operator<< (std::ostream &os, PluginSpec const &spec)

  *Output the name, refname and size of config.*
- bool operator== (PluginSpec const &self, PluginSpec const &other)

  *Compare two pluginspec if their value is equal.*
- bool operator!= (PluginSpec const &self, PluginSpec const &other)

  *Compare two pluginspec if their value is not equal.*

## 571.2.1 Detailed Description

This namespace is for the libtool library.

**Note**

You have to link against libelektratools if you want to use functionality from it. Contrary to classes in namespace kdb it is not header-only.

**See also**

Backend for an entry point

## 571.2.2 Function Documentation

### 571.2.2.1 operator"!=()

```
bool kdb::tools::operator!= (
            PluginSpec const & self,
            PluginSpec const & other )
```

Compare two pluginspec if their value is not equal.

**Note**

the content of getConfig() will be only compared with keynames, not content!

### 571.2.2.2 operator<<()

```
std::ostream & kdb::tools::operator<< (
            std::ostream & os,
            Backend const & b )
```

Prints the current status.

**Parameters**

| | |
|---|---|
| *os* | stream to print to |
| *b* | backend to get status from |

**Returns**

ref to stream

#### 571.2.2.3 operator==()

```
bool kdb::tools::operator== (
            PluginSpec const & self,
            PluginSpec const & other )
```

Compare two pluginspec if their value is equal.

**Note**

the content of getConfig() will be only compared with keynames, not content!

#### 571.2.2.4 parseArguments() [1/2]

```
template<typename Iterator >
PluginSpecVector kdb::tools::parseArguments (
            Iterator first,
            Iterator last )
```

Parse a complete commandline that is already tokenized in pluginname pluginconfig.

**Template Parameters**

| | |
|---|---|
| *Iterator* | forward iterator type |

**Parameters**

| | |
|---|---|
| *cmdline* | contains space separated plugins with optional plugin configurations |

**Returns**

a parsed PluginSpecVector

#### 571.2.2.5 parseArguments() [2/2]

```
PluginSpecVector kdb::tools::parseArguments (
            std::string const & cmdline )
```

Parse a complete commandline.

**Parameters**

| | |
|---|---|
| *cmdline* | contains space separated plugins with optional plugin configurations |

**Note**

currently whitespaces are not allowed within pluginname or config, use iterator interface parseArguments() if you need it.

**See also**

parseArguments()

**Returns**

a parsed PluginSpecVector

#### 571.2.2.6 parsePluginArguments()

```
KeySet kdb::tools::parsePluginArguments (
            std::string const & pluginArguments,
            std::string const & basepath )
```

Parse a string containing information to create a KeySet.

**Parameters**

| | |
|---|---|
| *pluginArguments* | comma (,) to separate key=value, contains no whitespaces |

**Returns**

newly created keyset with the information found in the string

## 571.3 Package org.libelektra

JNA based proxy for native libelektra.

### Packages

- package exception

  *Exceptions and error mapping for JNA based proxy for native libelektra.*

### Classes

- class KDB

  *Represents a session with the Elektra key database.*
- class KDBException

  *Wraps Elektra errors into the corresponding Java exceptions.*
- class Key

  *Key represents a native Elektra key providing access to its name, value and meta information.*
- class KeySet

  *Java representation of a native Elektra key set, a container for keys.*
- class NativePlugin

  *This class can be used to load native Elektra plugins to be used by Java directly.*
- interface Plugin

  *Java representation of an Elektra plugin.*
- class PluginLoader

  *This class can be used to load plugins from Elektra.*
- class ReadableKey

  *Read only key representing a native Elektra key providing read access to its name and value.*

### 571.3.1 Detailed Description

JNA based proxy for native libelektra.

## 571.4 Package org.libelektra.exception

Exceptions and error mapping for JNA based proxy for native libelektra.

## Classes

- class KDBClosedException

  *Indicates that an already closed KDB session has been accessed.*
- class KeyBinaryValueException

  *Indicates a key's underlying native key value is not of type binary, and therefore is not compatible with the invoked functionality raising this exception.*
- class KeyException

  *Indicates a generic exception while calling one of Key's methods*

  *This exception is related to unrecoverable C API specific errors (primarily allocation problems).*
- class KeyMetaException

  *Indicates Key#copyMeta(Key, String), Key#copyAllMeta(Key), {} or Key#removeMeta(String)} failed because the key name is invalid, the r*
- class KeyNameException

  *Indicates Key#setName(String), Key#setBaseName(String) or {} failed because the key name is invalid, the key was inserted in a key set*
- class KeySetException

  *Indicates a generic exception while calling one of KeySet's methods*

  *This exception is related to unrecoverable C API specific errors (primarily allocation problems).*
- class KeyStringValueException

  *Indicates a key's underlying native key value is not of type string, and therefore is not compatible with the invoked functionality raising this exception.*

### 571.4.1 Detailed Description

Exceptions and error mapping for JNA based proxy for native libelektra.

# Chapter 572

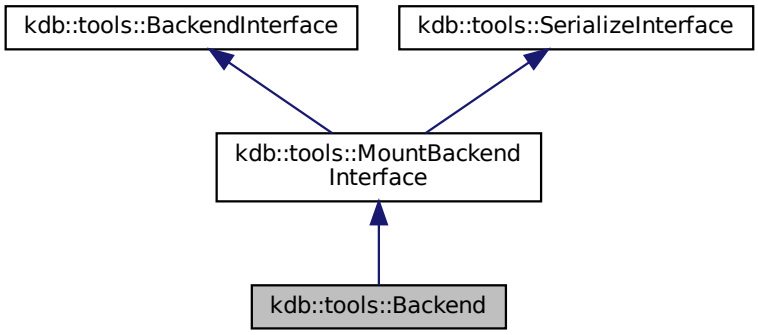
# Class Documentation

## 572.1 kdb::tools::Backend Class Reference

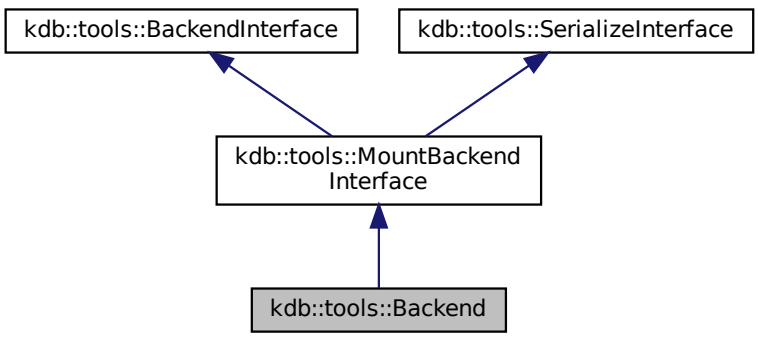
A low-level representation of the backend (= set of plugins) that can be mounted.

`#include <backend.hpp>`

Inheritance diagram for kdb::tools::Backend:




Collaboration diagram for kdb::tools::Backend:



## Public Member Functions

- Backend ()

  *Creates a new empty backend.*
- void setMountpoint (Key mountpoint, KeySet mountConf)

  *Sets the mountpoint for the backend.*
- void setBackendConfig (KeySet const &ks)

  *Backend Config to add to.*
- void addPlugin (PluginSpec const &spec)

  *Add a plugin that can be loaded, meets all constraints.*
- void useConfigFile (std::string file)
- bool validated () const
- void serialize (kdb::KeySet &ret)

### 572.1.1 Detailed Description

A low-level representation of the backend (= set of plugins) that can be mounted.
To build a backend, you should prefer BackendBuilder, which automatically fixes ordering and allows us to remove plugins.

### 572.1.2 Member Function Documentation

#### 572.1.2.1 addPlugin()

```
void kdb::tools::Backend::addPlugin (
            PluginSpec const & plugin )  [virtual]
```

Add a plugin that can be loaded, meets all constraints.

**Note**

that this does not mean that the backend validates after it is added. It only means that the situation is not getting worse.

**Exceptions**

| | |
|---|---|
| *PluginCheckException* | or its subclasses if it was not possible to load the plugin |

For validation

**See also**

validated().

Implements kdb::tools::BackendInterface.

#### 572.1.2.2 serialize()

```
void kdb::tools::Backend::serialize (
            kdb::KeySet & ret )  [virtual]
```

**Precondition**

name and mountpoint set Add plugin serialization into keyset ret.

Only can be done once! (see firstRef in Plugin)
Implements kdb::tools::SerializeInterface.

#### 572.1.2.3 setBackendConfig()

```
void kdb::tools::Backend::setBackendConfig (
            KeySet const & ks )  [virtual]
```

Backend Config to add to.

**Parameters**

| | |
|---|---|
| *ks* | the config to add, should be below system:/ |

Implements kdb::tools::MountBackendInterface.

#### 572.1.2.4 setMountpoint()

```
void kdb::tools::Backend::setMountpoint (
            Key mountpoint,
            KeySet mountConf )  [virtual]
```

Sets the mountpoint for the backend.

**Exceptions**

| | |
|---|---|
| *MountpointInvalidException* | |
| *MountpointAlreadyInUseException* | |

**Parameters**

| | |
|---|---|
| *mountpoint* | the key name will be used as mountpoint. It is allowed to pass a key with a cascading name. |
| *mountConf* | needs to include the keys below system:/elektra/mountpoints |

Implements kdb::tools::MountBackendInterface.

#### 572.1.2.5 useConfigFile()

```
void kdb::tools::Backend::useConfigFile (
            std::string file )  [virtual]
```

**Precondition**

: resolver needs to be loaded first Will check the filename and use it as configFile for this backend.

**Exceptions**

| | |
|---|---|
| *FileNotValidException* | if filename is not valid |
| *MissingSymbol* | if plugin does not implement 'checkfile' |

Implements kdb::tools::MountBackendInterface.

#### 572.1.2.6 validated()

```
bool kdb::tools::Backend::validated ( ) const  [virtual]
```

**Returns**

true if backend is validated

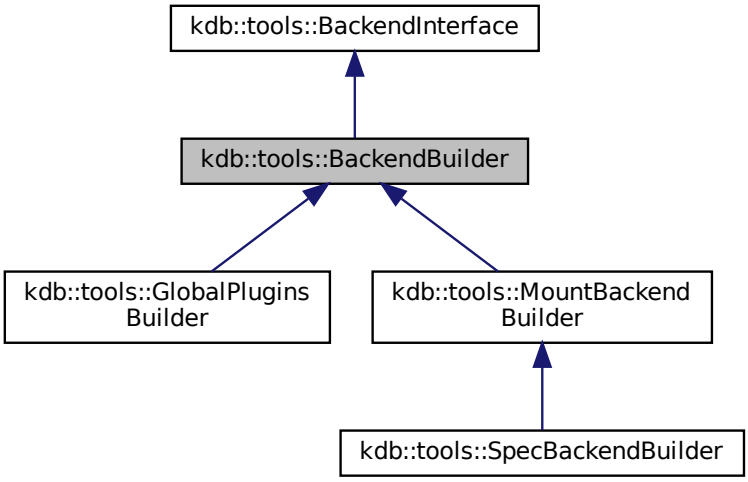
false if more plugins are needed to be valided

Implements kdb::tools::MountBackendInterface.
The documentation for this class was generated from the following files:

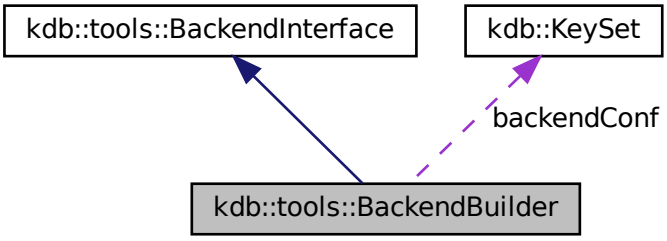
- backend.hpp
- src/backend.cpp

## 572.2 kdb::tools::BackendBuilder Class Reference

Highlevel interface to build a backend.
`#include <backendbuilder.hpp>`
Inheritance diagram for kdb::tools::BackendBuilder:




Collaboration diagram for kdb::tools::BackendBuilder:



## Public Member Functions

- void addPlugin (PluginSpec const &plugin)

  *Add a plugin.*
- std::vector< std::string > resolveNeeds (bool addRecommends=true)

  *resolve all needs that were not resolved by adding plugins.*

### 572.2.1 Detailed Description

Highlevel interface to build a backend.
Automatically reorders plugins and has different modes which Backend should be built.

### 572.2.2 Member Function Documentation

#### 572.2.2.1 addPlugin()

```
void kdb::tools::BackendBuilder::addPlugin (
            PluginSpec const & plugin )  [virtual]
```

Add a plugin.

**Precondition**

Needs to be a unique new name (use refname if you want to add the same module multiple times)

Will automatically resolve virtual plugins to actual plugins.
Also calls the checkconf function if provided by the plugin. The checkconf function has the following signature: int checkconf (Key * errorKey, KeySet * config) and allows a plugin to verify its configuration at mount time.

**See also**

resolveNeeds()

**Parameters**

| | |
|---|---|
| *plugin* | |

Implements kdb::tools::BackendInterface.
Reimplemented in kdb::tools::MountBackendBuilder.

#### 572.2.2.2 resolveNeeds()

```
std::vector< std::string > kdb::tools::BackendBuilder::resolveNeeds (
            bool addRecommends = true )
```

resolve all needs that were not resolved by adding plugins.

**Warning**

Must only be used once after all plugins/recommends are added.

**Returns**

the missing recommended plugins

**Return values**

| | |
|---|---|
| *empty* | if addRecommends was false |

**See also**

addPlugin()

The documentation for this class was generated from the following files:

- backendbuilder.hpp
- backendbuilder.cpp

## 572.3 kdb::tools::BackendBuilderInit Class Reference

Used as argument of constructor of ∗BackendBuilder.
`#include <backendbuilder.hpp>`

### 572.3.1 Detailed Description

Used as argument of constructor of ∗BackendBuilder.
Avoids the implementation of 5 Constructors for each of the ∗BackendBuilder.
The documentation for this class was generated from the following files:

- backendbuilder.hpp
- backendbuilder.cpp

## 572.4 kdb::tools::BackendFactory Class Reference

Factory for MountBackendInterface.
`#include <backend.hpp>`

### Public Member Functions

- MountBackendInterfacePtr create () const

  *Create classes that implement MountBackendInterface.*

### 572.4.1 Detailed Description

Factory for MountBackendInterface.
The documentation for this class was generated from the following file:

- backend.hpp

## 572.5 kdb::tools::BackendInfo Struct Reference

Info about a backend.
`#include <backends.hpp>`

### Public Attributes

- std::string mountpoint

  *where the backend is mounted*
- std::string path

  *the configuration file path to this backend*

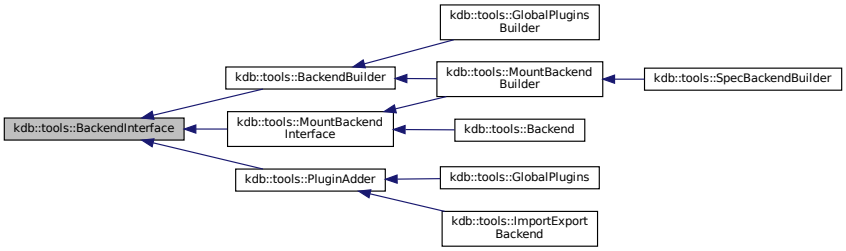
### 572.5.1 Detailed Description

Info about a backend.
The documentation for this struct was generated from the following file:

- backends.hpp

## 572.6 kdb::tools::BackendInterface Class Reference

Minimal interface to add plugins.
`#include <backend.hpp>`
Inheritance diagram for kdb::tools::BackendInterface:




### 572.6.1 Detailed Description

Minimal interface to add plugins.
The documentation for this class was generated from the following files:

- backend.hpp
- src/backend.cpp

## 572.7 kdb::tools::Backends Class Reference

Allows one to list backends.
`#include <backends.hpp>`

### Static Public Member Functions

- static BackendInfoVector getBackendInfo (KeySet mountConf)

  *give info about current mounted backends*
- static BackendInfo findBackend (std::string const &backend, KeySet mountConf, bool verbose=false)

  *Find a backend in the given name.*
- static bool umount (std::string const &backend, KeySet &mountConf)

  *Unmount a backend by given mountPath.*
- static std::string getBasePath (std::string name)

  *returns the base path of a mounted backend below system:/elektra/mountpoints*

### Static Public Attributes

- static const char ∗const mountpointsPath = "system:/elektra/mountpoints"

  *Below this path is the mountConf.*

### 572.7.1 Detailed Description

Allows one to list backends.

### 572.7.2 Member Function Documentation

#### 572.7.2.1 findBackend()

```
BackendInfo kdb::tools::Backends::findBackend (
            std::string const & mountPath,
            KeySet mountConf,
            bool verbose = false )  [static]
```

Find a backend in the given name.

**Parameters**

| | |
|---|---|
| *mountPath* | the given backend name to find |

For backwards compatibility old-style names containing _ instead of escaped / are accepted if no modern-style mountpoint is found.

**Parameters**

| | |
|---|---|
| *mountConf* | the configuration to search (should contain keys below mountpointsPath to find something) |

**Returns**

the found backend or an empty BackendInfo if nothing found (with empty strings)

#### 572.7.2.2 getBackendInfo()

```
Backends::BackendInfoVector kdb::tools::Backends::getBackendInfo (
            KeySet mountConf )  [static]
```

give info about current mounted backends

**Parameters**

| | |
|---|---|
| *mountConf* | a keyset that contains everything below Backends::mountpointsPath |

**Returns**

an vector of information about mounted backends

#### 572.7.2.3 getBasePath()

```
std::string kdb::tools::Backends::getBasePath (
            std::string mp )  [static]
```

returns the base path of a mounted backend below system:/elektra/mountpoints

**Parameters**

| | |
|---|---|
| *mp* | the mountpoint (name will be derived from it) |

**Returns**

the properly prefixed and escaped name

#### 572.7.2.4 umount()

```
bool kdb::tools::Backends::umount (
            std::string const & mountPath,
            KeySet & mountConf )  [static]
```

Unmount a backend by given mountPath.

**Parameters**

| | |
|---|---|
| *mountPath* | the given mountpoint |

Uses findBackend() to locate the backend.

**Return values**

| | |
|---|---|
| *true* | if something was done |
| *false* | if nothing was done (but also no error) |

The documentation for this class was generated from the following files:

- backends.hpp
- backends.cpp

## 572.8 kdb::Command Struct Reference

Used by contexts for callbacks (to run code using a mutex).
`#include <kdbvalue.hpp>`

### Public Types

- typedef std::function< Pair()> Func

  *Typedef for function that returns oldKey, newKey pair.*

### 572.8.1 Detailed Description

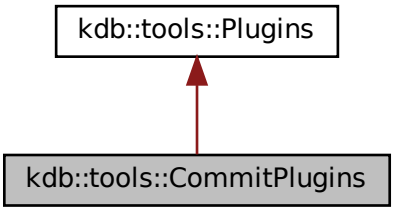
Used by contexts for callbacks (to run code using a mutex).
Following scenarios are possible: !oldName && !newName: execute code, do nothing else !oldName && newName: attach oldName && newName: reattach oldName == newName: assignment, attach for inter-thread updates oldName && !newName: detach
The documentation for this struct was generated from the following file:

- kdbvalue.hpp

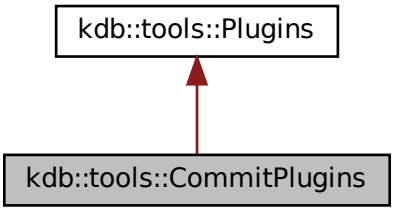
## 572.9 kdb::tools::CommitPlugins Class Reference

Plugins to handle errors during configuration access.
`#include <plugins.hpp>`

Inheritance diagram for kdb::tools::CommitPlugins:




Collaboration diagram for kdb::tools::CommitPlugins:




### 572.9.1 Detailed Description

Plugins to handle errors during configuration access.
The documentation for this class was generated from the following files:

- plugins.hpp
- plugins.cpp

## 572.10 kdb::Context Class Reference

Provides a context for configuration.
`#include <kdbcontext.hpp>`
Inherits kdb::Subject.
Inherited by kdb::ThreadContext.

### Public Member Functions

- std::string operator[] (std::string const &layer) const

  *Lookup value for a current active layer.*
- size_t size () const
- void attachByName (std::string const &key_name, ValueObserver &observer)

  *Attach observer using to all events given by its specification (name)*
- std::string evaluate (std::string const &key_name) const

*Evaluate a specification (name) and return a key name under current context.*

- std::string evaluate (std::string const &key_name, std::function< bool(std::string const &, std::string &, bool in_group)> const &on_layer) const

  *Evaluate specification with this context.*

- template<typename T , typename... Args>
  std::shared_ptr< Layer > activate (Args &&... args)

  *Globally activate the layer.*

## 572.10.1 Detailed Description

Provides a context for configuration.
Is a subject for observers.
Holds currently active layers and allows global/scoped activation of layers.

## 572.10.2 Member Function Documentation

### 572.10.2.1 activate()

```
template<typename T , typename...  Args>
std::shared_ptr<Layer> kdb::Context::activate (
            Args &&...  args )  [inline]
```

Globally activate the layer.

**Template Parameters**

| | |
|---|---|
| *T* | the layer to activate |
| *Args* | the types for the arguments to pass to layer construction |

**Parameters**

| | |
|---|---|
| *args* | the arguments to pass to layer construction |

### 572.10.2.2 attachByName()

```
void kdb::Context::attachByName (
            std::string const & key_name,
            ValueObserver & observer )  [inline]
```

Attach observer using to all events given by its specification (name)

**Parameters**

| | |
|---|---|
| *key_name* | the name with placeholders to be used for attaching |
| *observer* | the observer to attach to |

### 572.10.2.3 evaluate() [1/2]

```
std::string kdb::Context::evaluate (
            std::string const & key_name ) const  [inline]
```

Evaluate a specification (name) and return a key name under current context.

**Parameters**

| | |
|---|---|
| *key_name* | the name with placeholders to be evaluated |

#### 572.10.2.4 evaluate() [2/2]

```
std::string kdb::Context::evaluate (
            std::string const & key_name,
            std::function< bool(std::string const &, std::string &, bool in_group)> const &
on_layer ) const  [inline]
```

Evaluate specification with this context.

**Parameters**

| | |
|---|---|
| *key_name* | the keyname with placeholders to evaluate |
| *on_layer* | the function to be called for every placeholder found |

**on_layer is called for every layer in the**

specification.

**Returns**

the evaluated string

#### 572.10.2.5 operator[]()

```
std::string kdb::Context::operator[] (
            std::string const & layer ) const  [inline]
```

Lookup value for a current active layer.

**Parameters**

| | |
|---|---|
| *layer* | the name of the requested layer |

**Returns**

the layer

#### 572.10.2.6 size()

```
size_t kdb::Context::size ( ) const  [inline]
```

**Returns**

size of all current layers (to be used with operator[])

The documentation for this class was generated from the following file:

- kdbcontext.hpp

## 572.11 kdb::ContextPolicyIs< Policy > Class Template Reference

Needed by the user to set one of the policies.

`#include <kdbvalue.hpp>`
Inherits kdb::DefaultPolicies.

### 572.11.1 Detailed Description

**template<typename Policy>**
**class kdb::ContextPolicyIs< Policy >**

Needed by the user to set one of the policies.

**Template Parameters**

| | |
|---|---|
| *Policy* | |

The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.12 kdb::Coordinator Class Reference

Thread safe coordination of ThreadContext per Threads.
`#include <kdbthread.hpp>`

### 572.12.1 Detailed Description

Thread safe coordination of ThreadContext per Threads.
The documentation for this class was generated from the following file:

- kdbthread.hpp

## 572.13 org.libelektra.Key.CreateArgumentTag Enum Reference

Argument tags for use with create(String, Object...).

### Public Attributes

- KEY_END =(0)

  *Used as a parameter terminator.*
- KEY_VALUE =(1 << 1)

  *Flag for the key data.*
- KEY_BINARY =(1 << 4)

  *Flag if the key is binary.*
- KEY_SIZE =(1 << 11)

  *Flag for maximum size to limit value.*
- KEY_META =(1 << 15)

  *Flag for metadata.*

### 572.13.1 Detailed Description

Argument tags for use with create(String, Object...).
The documentation for this enum was generated from the following file:

- Key.java

## 572.14 kdb::DefaultGetPolicy Class Reference

Implements lookup with spec.
`#include <kdbvalue.hpp>`

### 572.14.1 Detailed Description

Implements lookup with spec.
The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.15 kdb::DefaultSetPolicy Class Reference

Implements creating user:/ key when key is not found.
`#include <kdbvalue.hpp>`

### 572.15.1 Detailed Description

Implements creating user:/ key when key is not found.
The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.16 DocBindingData Struct Reference

[kdbio operation data]

### Public Attributes

- char * foo

  *Example member.*

### 572.16.1 Detailed Description

[kdbio operation data]
[kdbio binding data] Container for additional information for an I/O binding.
The documentation for this struct was generated from the following file:

- io_doc.c

## 572.17 DocOperationData Struct Reference

[kdbio operation data]

### Public Attributes

- char * bar

  *Example member.*

### 572.17.1 Detailed Description

[kdbio operation data]
Container for additional information for I/O binding operations.
It is helpful to create a data structure for your binding to store additional data
The documentation for this struct was generated from the following file:

- io_doc.c

# 572.18 kdb::ElektraDiff Class Reference

This class is a wrapper around the ElektraDiff C struct.
`#include <elektradiff.hpp>`

## Public Member Functions

- ElektraDiff (ckdb::ElektraDiff *cdiff)

  *Constructs a diff out of a C diff.*
- ElektraDiff (ElektraDiff &other)

  *Takes a reference of another diff.*
- ElektraDiff (ElektraDiff const &other)

  *Takes a reference of another diff.*
- void undo (KeySet &ks)

  *Undo the changes represented in this diff.*
- void operator++ (int) const
- void operator++ () const
- void operator-- (int) const
- void operator-- () const
- ckdb::ElektraDiff * getDiff () const

  *Passes out the raw diff pointer.*
- ckdb::ElektraDiff * operator* () const

  *Passes out the raw diff pointer.*
- ElektraDiff & operator= (const ElektraDiff &other)

  *Assign a diff.*
- uint16_t getReferenceCounter () const
- void removeOther (std::string const &parentKeyName)
- void removeOther (const Key &parentKey)
- void removeSameOrBelow (std::string const &cutpointName)
- void removeSameOrBelow (const Key &cutpoint)
- void removeKey (std::string const &keyName)
- void removeKey (const Key &key)
- ElektraDiff cut (std::string const &cutpointName)
- ElektraDiff cut (const Key &cutpoint)
- ElektraDiff dup () const
- bool isEmpty () const
- KeySet getAddedKeys () const
- KeySet getModifiedKeys () const
- KeySet getRemovedKeys () const
- KeySet getAddedMetaKeys (std::string const &keyName) const
- KeySet getAddedMetaKeys (const Key &key) const
- KeySet getModifiedMetaKeys (std::string const &keyName) const
- KeySet getModifiedMetaKeys (const Key &key) const
- KeySet getRemovedMetaKeys (std::string const &keyName) const
- KeySet getRemovedMetaKeys (const Key &key) const

## Static Public Member Functions

- static ElektraDiff calculateDiff (const KeySet &newKeys, const KeySet &oldKeys, const std::string &parent↩ KeyName)

  *Calculates the difference between the given keysets The diff will contain the keys that were added, modified and removed in `newKeys`.*
- static ElektraDiff calculateDiff (const KeySet &newKeys, const KeySet &oldKeys, const Key &parentKey)

  *Calculates the difference between the given keysets The diff will contain the keys that were added, modified and removed in `newKeys`.*

### 572.18.1 Detailed Description

This class is a wrapper around the ElektraDiff C struct.

**Invariant**

always holds an underlying ElektraDiff C object.

### 572.18.2 Constructor & Destructor Documentation

#### 572.18.2.1 ElektraDiff() [1/3]

```
kdb::ElektraDiff::ElektraDiff (
            ckdb::ElektraDiff * cdiff )  [inline], [explicit]
```

Constructs a diff out of a C diff.

**Parameters**

| | |
|---|---|
| *cdiff* | the diff to work with |

#### 572.18.2.2 ElektraDiff() [2/3]

```
kdb::ElektraDiff::ElektraDiff (
            ElektraDiff & other )  [inline]
```

Takes a reference of another diff.
The diff will not be copied, but the reference counter will be increased.

**Parameters**

| | |
|---|---|
| *other* | the diff to work with |

#### 572.18.2.3 ElektraDiff() [3/3]

```
kdb::ElektraDiff::ElektraDiff (
            ElektraDiff const & other )  [inline]
```

Takes a reference of another diff.
The diff will not be copied, but the reference counter will be increased.

**Parameters**

| | |
|---|---|
| *other* | the diff to work with |

### 572.18.3 Member Function Documentation

#### 572.18.3.1 calculateDiff() [1/2]

```
ElektraDiff kdb::ElektraDiff::calculateDiff (
            const KeySet & newKeys,
            const KeySet & oldKeys,
            const Key & parentKey )  [inline], [static]
```

Calculates the difference between the given keysets The diff will contain the keys that were added, modified and removed in `newKeys`.

**Parameters**

| | |
|---|---|
| *newKeys* | the new keyset |
| *oldKeys* | the old keyset |
| *parentKey* | only changes same or below this key will be calculated |

**Returns**

diff of the two given keysets

#### 572.18.3.2 calculateDiff() [2/2]

```
ElektraDiff kdb::ElektraDiff::calculateDiff (
            const KeySet & newKeys,
            const KeySet & oldKeys,
            const std::string & parentKeyName )  [inline], [static]
```

Calculates the difference between the given keysets The diff will contain the keys that were added, modified and removed in `newKeys`.

**Parameters**

| | |
|---|---|
| *newKeys* | the new keyset |
| *oldKeys* | the old keyset |
| *parentKeyName* | only changes same or below this key will be calculated |

**Returns**

diff of the two given keysets

#### 572.18.3.3 cut() [1/2]

```
ElektraDiff kdb::ElektraDiff::cut (
            const Key & cutpoint )  [inline]
```

#### 572.18.3.4 cut() [2/2]

```
ElektraDiff kdb::ElektraDiff::cut (
            std::string const & cutpointName )  [inline]
```

#### 572.18.3.5 dup()

```
ElektraDiff kdb::ElektraDiff::dup ( ) const  [inline]
```

#### 572.18.3.6 getAddedKeys()

```
KeySet kdb::ElektraDiff::getAddedKeys ( ) const  [inline]
```

#### 572.18.3.7 getAddedMetaKeys() [1/2]

```
KeySet kdb::ElektraDiff::getAddedMetaKeys (
            const Key & key ) const  [inline]
```

#### 572.18.3.8 getAddedMetaKeys() [2/2]

```
KeySet kdb::ElektraDiff::getAddedMetaKeys (
            std::string const & keyName ) const  [inline]
```

#### 572.18.3.9 getDiff()

```
ckdb::ElektraDiff * kdb::ElektraDiff::getDiff ( ) const  [inline]
```

Passes out the raw diff pointer.
This pointer can be used to directly change the underlying diff object.

**Note**

that the ownership remains in the object

#### 572.18.3.10 getModifiedKeys()

```
KeySet kdb::ElektraDiff::getModifiedKeys ( ) const  [inline]
```

#### 572.18.3.11 getModifiedMetaKeys() [1/2]

```
KeySet kdb::ElektraDiff::getModifiedMetaKeys (
            const Key & key ) const  [inline]
```

#### 572.18.3.12 getModifiedMetaKeys() [2/2]

```
KeySet kdb::ElektraDiff::getModifiedMetaKeys (
            std::string const & keyName ) const  [inline]
```

#### 572.18.3.13 getReferenceCounter()

```
uint16_t kdb::ElektraDiff::getReferenceCounter ( ) const  [inline]
```

#### 572.18.3.14 getRemovedKeys()

```
KeySet kdb::ElektraDiff::getRemovedKeys ( ) const  [inline]
```

#### 572.18.3.15 getRemovedMetaKeys() [1/2]

```
KeySet kdb::ElektraDiff::getRemovedMetaKeys (
            const Key & key ) const  [inline]
```

#### 572.18.3.16 getRemovedMetaKeys() [2/2]

```
KeySet kdb::ElektraDiff::getRemovedMetaKeys (
            std::string const & keyName ) const  [inline]
```

#### 572.18.3.17 isEmpty()

```
bool kdb::ElektraDiff::isEmpty ( ) const  [inline]
```

#### 572.18.3.18 operator*()

```
ckdb::ElektraDiff * kdb::ElektraDiff::operator* ( ) const  [inline]
```

Passes out the raw diff pointer.
This pointer can be used to directly change the underlying diff object.

**Note**

that the ownership remains in the object

#### 572.18.3.19 operator++() [1/2]

```
void kdb::ElektraDiff::operator++ ( ) const  [inline]
```

#### 572.18.3.20 operator++() [2/2]

```
void kdb::ElektraDiff::operator++ (
            int  ) const  [inline]
```

#### 572.18.3.21 operator--() [1/2]

```
void kdb::ElektraDiff::operator-- ( ) const  [inline]
```

#### 572.18.3.22 operator--() [2/2]

```
void kdb::ElektraDiff::operator-- (
            int  ) const  [inline]
```

#### 572.18.3.23 operator=()

```
ElektraDiff & kdb::ElektraDiff::operator= (
            const ElektraDiff & other )  [inline]
```

Assign a diff.

**Parameters**

| | |
|---|---|
| *other* | the diff to assign |

**Returns**

reference to this

#### 572.18.3.24 removeKey() [1/2]

```
void kdb::ElektraDiff::removeKey (
            const Key & key )  [inline]
```

#### 572.18.3.25 removeKey() [2/2]

```
void kdb::ElektraDiff::removeKey (
            std::string const & keyName )  [inline]
```

#### 572.18.3.26 removeOther() [1/2]

```
void kdb::ElektraDiff::removeOther (
            const Key & parentKey )  [inline]
```

#### 572.18.3.27 removeOther() [2/2]

```
void kdb::ElektraDiff::removeOther (
            std::string const & parentKeyName )  [inline]
```

#### 572.18.3.28 removeSameOrBelow() [1/2]

```
void kdb::ElektraDiff::removeSameOrBelow (
            const Key & cutpoint )  [inline]
```

#### 572.18.3.29 removeSameOrBelow() [2/2]

```
void kdb::ElektraDiff::removeSameOrBelow (
            std::string const & cutpointName )  [inline]
```

#### 572.18.3.30 undo()

```
void kdb::ElektraDiff::undo (
            KeySet & ks )  [inline]
```

Undo the changes represented in this diff.

**Parameters**

| | |
|---|---|
| *ks* | the keyset where the changs should be undone |

The documentation for this class was generated from the following file:

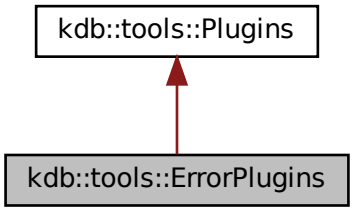
- elektradiff.hpp

## 572.19 kdb::tools::ErrorPlugins Class Reference

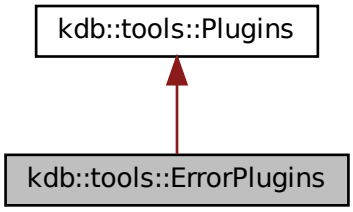
Plugins to handle errors during configuration access.

`#include <plugins.hpp>`

Inheritance diagram for kdb::tools::ErrorPlugins:




Collaboration diagram for kdb::tools::ErrorPlugins:




### 572.19.1 Detailed Description

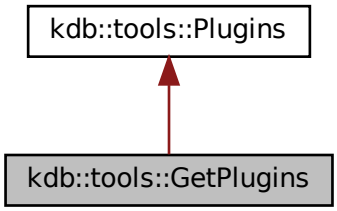
Plugins to handle errors during configuration access.
The documentation for this class was generated from the following files:

- plugins.hpp
- plugins.cpp

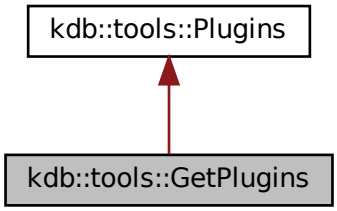
## 572.20 kdb::tools::GetPlugins Class Reference

Plugins to get configuration.
`#include <plugins.hpp>`

Inheritance diagram for kdb::tools::GetPlugins:




Collaboration diagram for kdb::tools::GetPlugins:




## Public Member Functions

- void tryPlugin (Plugin &plugin)

  *Returns true if GetPlugins are valid afterwards.*

## 572.20.1 Detailed Description

Plugins to get configuration.

## 572.20.2 Member Function Documentation

### 572.20.2.1 tryPlugin()

```
void kdb::tools::GetPlugins::tryPlugin (
            Plugin & plugin )
```

Returns true if GetPlugins are valid afterwards.
Will throw an exception if plugin could not be added.
The documentation for this class was generated from the following files:

- plugins.hpp
- plugins.cpp

# 572.21 kdb::GetPolicyIs< Policy > Class Template Reference

Needed by the user to set one of the policies.
`#include <kdbvalue.hpp>`
Inherits kdb::DefaultPolicies.

## 572.21.1 Detailed Description

**template<typename Policy>**
**class kdb::GetPolicyIs< Policy >**

Needed by the user to set one of the policies.

**Template Parameters**

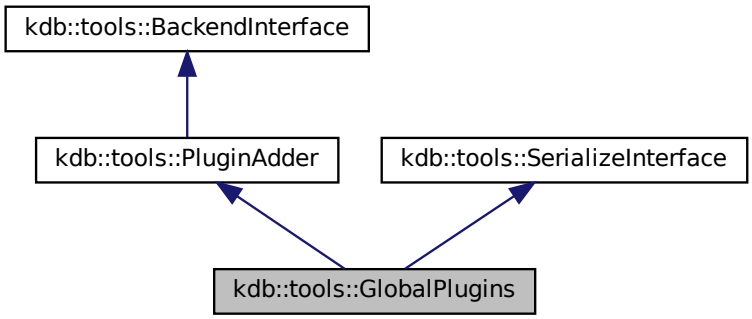
| | |
|---|---|
| *Policy* | |

The documentation for this class was generated from the following file:

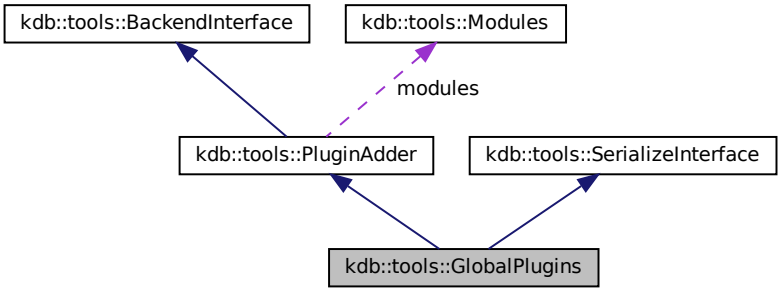
- kdbvalue.hpp

# 572.22 kdb::tools::GlobalPlugins Class Reference

Low level representation of global plugins.
`#include <backend.hpp>`
Inheritance diagram for kdb::tools::GlobalPlugins:



Collaboration diagram for kdb::tools::GlobalPlugins:



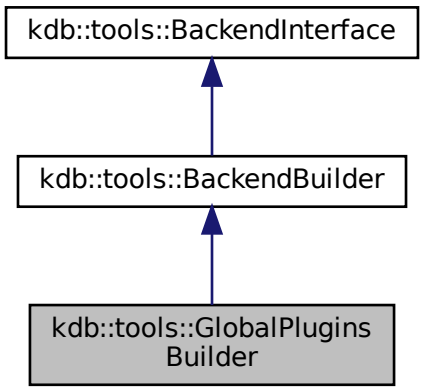

### 572.22.1 Detailed Description

Low level representation of global plugins.
The documentation for this class was generated from the following files:

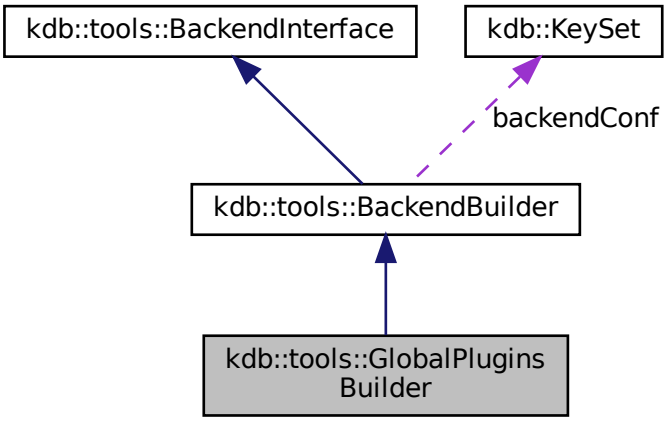
- backend.hpp
- src/backend.cpp

## 572.23 kdb::tools::GlobalPluginsBuilder Class Reference

Build global plugins.
`#include <backendbuilder.hpp>`
Inheritance diagram for kdb::tools::GlobalPluginsBuilder:



Collaboration diagram for kdb::tools::GlobalPluginsBuilder:




### Static Public Attributes

- static const char ∗const globalPluginsPath = "system:/elektra/globalplugins"

  *Below this path is the configuration for global plugins.*

### Additional Inherited Members

#### 572.23.1 Detailed Description

Build global plugins.
The documentation for this class was generated from the following files:

- backendbuilder.hpp
- backendbuilder.cpp

## 572.24 std::hash< kdb::Key > Struct Reference

Support for putting Key in a hash.
`#include <key.hpp>`

#### 572.24.1 Detailed Description

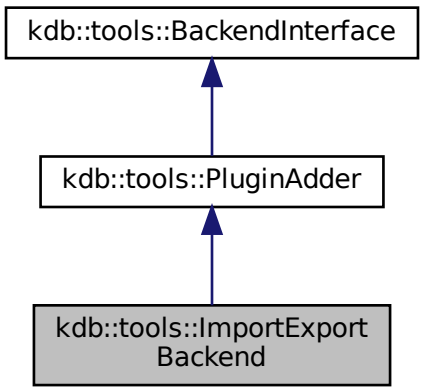
Support for putting Key in a hash.
The documentation for this struct was generated from the following file:

- key.hpp

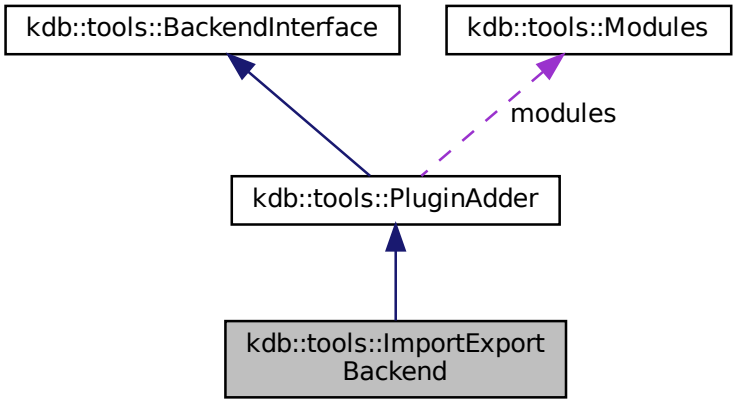
## 572.25 kdb::tools::ImportExportBackend Class Reference

Backend for import/export functionality.
`#include <backend.hpp>`

Inheritance diagram for kdb::tools::ImportExportBackend:




Collaboration diagram for kdb::tools::ImportExportBackend:




### 572.25.1 Detailed Description

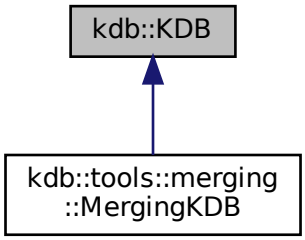
Backend for import/export functionality.
(only partly implemented)
The documentation for this class was generated from the following files:

- backend.hpp
- src/backend.cpp

## 572.26 kdb::KDB Class Reference

Constructs a class KDB.
`#include <kdb.hpp>`

Inheritance diagram for kdb::KDB:




## Public Member Functions

- KDB ()

  *Constructs a class KDB.*
- KDB (Key &errorKey)

  *Constructs a class KDB.*
- KDB (KeySet &contract)

  *Constructs a class KDB.*
- KDB (KeySet &contract, Key &errorKey)

  *Constructs a class KDB.*
- virtual void open (Key &errorKey)

  *Open the database.*
- virtual void open (KeySet &contract, Key &errorKey)

  *Open the database.*
- virtual void close () throw ()

  *Close the database.*
- virtual void close (Key &errorKey) throw ()

  *Close the database.*
- virtual int get (KeySet &returned, std::string const &keyname)

  *Get all keys below keyname inside returned.*
- virtual int get (KeySet &returned, Key &parentKey)

  *Get all keys below parentKey inside returned.*
- virtual int set (KeySet &returned, std::string const &keyname)

  *Set all keys below keyname.*
- virtual int set (KeySet &returned, Key &parentKey)

  *Set all keys below parentKey.*
- virtual ElektraDiff calculateChanges (KeySet &changedKeySet, std::string const &parentKeyName)

  *Calculates the changes between the provided KeySet and the current state of the KDB.*
- virtual ElektraDiff calculateChanges (KeySet &changedKeySet, Key &parentKey)

  *Calculates the changes between the provided KeySet and the current state of the KDB.*
- ckdb::KDB * getKdb () const

  *Passes out the raw kdb pointer.*
- ckdb::KDB * operator* () const

  *Is an abbreviation for getKdb.*

### 572.26.1 Detailed Description

Constructs a class KDB.

#### Exceptions

| | |
|---|---|
| *KDBException* | if database could not be opened |

Opens the session with the Key database.

#### Precondition

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the KDB data structure will be initialized.
The pointer to the KDB structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a KDB handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

#### Precondition

errorKey must be a valid key, e.g. created with keyNew()

#### Parameters

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

#### Returns

handle to the newly created KDB on success

#### Return values

| | |
|---|---|
| *NULL* | on failure |

#### Since

1.0.0

#### See also

kdbClose() to close the session of a Key database opened by kdbOpen()

Access to the key database.

**Invariant**

the object holds a valid connection to the key database or is empty

## 572.26.2 Constructor & Destructor Documentation

### 572.26.2.1 KDB() [1/4]

`kdb::KDB::KDB ( ) [inline]`

Constructs a class KDB.

**Exceptions**

| | |
|---|---|
| *KDBException* | if database could not be opened |

Opens the session with the Key database.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the `KDB` data structure will be initialized.
The pointer to the `KDB` structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a `KDB` handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

handle to the newly created KDB on success

**Return values**

| | |
|---|---|
| *NULL* | on failure |

**Since**

1.0.0

**See also**

kdbClose() to close the session of a Key database opened by kdbOpen()

### 572.26.2.2 KDB() [2/4]

```
kdb::KDB::KDB (
            Key & errorKey )  [inline], [explicit]
```

Constructs a class KDB.

**Parameters**

| | |
|---|---|
| *errorKey* | is useful if you want to get the warnings in the successful case, when no exception is thrown. |

**Exceptions**

| | |
|---|---|
| *KDBException* | if database could not be opened |

Opens the session with the Key database.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the KDB data structure will be initialized.
The pointer to the KDB structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a KDB handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

handle to the newly created KDB on success

**Return values**

| | |
|---|---|
| *NULL* | on failure |

**Since**

1.0.0

**See also**

kdbClose() to close the session of a Key database opened by kdbOpen()

### 572.26.2.3 KDB() [3/4]

```
kdb::KDB::KDB (
            KeySet & contract )  [inline], [explicit]
```

Constructs a class KDB.

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured |
| *errorKey* | is useful if you want to get the warnings in the successful case, when no exception is thrown. |

**Exceptions**

| | |
|---|---|
| *KDBException* | if database could not be opened |

Opens the session with the Key database.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the KDB data structure will be initialized.
The pointer to the KDB structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a KDB handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
```

```
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

handle to the newly created KDB on success

**Return values**

| | |
|---|---|
| *NULL* | on failure |

**Since**

1.0.0

**See also**

kdbClose() to close the session of a Key database opened by kdbOpen()

### 572.26.2.4 KDB() [4/4]

```
kdb::KDB::KDB (
            KeySet & contract,
            Key & errorKey )  [inline]
```

Constructs a class KDB.

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured |
| *errorKey* | is useful if you want to get the warnings in the successful case, when no exception is thrown. |

**Exceptions**

| | |
|---|---|
| *KDBException* | if database could not be opened |

Opens the session with the Key database.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the KDB data structure will be initialized.
The pointer to the KDB structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a KDB handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

handle to the newly created KDB on success

**Return values**

| | |
|---|---|
| *NULL* | on failure |

**Since**

1.0.0

**See also**

kdbClose() to close the session of a Key database opened by kdbOpen()

## 572.26.3 Member Function Documentation

### 572.26.3.1 calculateChanges() [1/2]

```
ElektraDiff kdb::KDB::calculateChanges (
            KeySet & changedKeySet,
```

```
            Key & parentKey )  [inline], [virtual]
```

Calculates the changes between the provided KeySet and the current state of the KDB.

**Parameters**

| | |
|---|---|
| *changedKeySet* | the keyset that should be used to diff |
| *parentKey* | only changes same or below this keys are calculated |

**Returns**

a diff with all the changes

#### 572.26.3.2 calculateChanges() [2/2]

```
ElektraDiff kdb::KDB::calculateChanges (
            KeySet & changedKeySet,
            std::string const & parentKeyName )  [inline], [virtual]
```

Calculates the changes between the provided KeySet and the current state of the KDB.

**Parameters**

| | |
|---|---|
| *changedKeySet* | the keyset that should be used to diff |
| *parentKeyName* | only changes same or below this keys are calculated |

**Returns**

a diff with all the changes

#### 572.26.3.3 close() [1/2]

```
void kdb::KDB::close ( ) throw ( )   [inline], [virtual]
```

Close the database.
The return value does not matter because its only a null pointer check.
Closes the session with the Key database.

**Precondition**

The handle must be a valid handle as returned from kdbOpen()

errorKey must be a valid key, e.g. created with keyNew()

This is the counterpart of kdbOpen().
You must call this method when you are finished working with the Key database. You can manipulate Key and KeySet objects also after kdbClose(), but you must not use any kdb∗() call afterwards.
The `handle` parameter will be finalized and all resources associated to it will be freed. After a kdbClose(), the `handle` cannot be used anymore.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *errorKey* | the key which holds error/warning information |

**Return values**

| | |
|---|---|
| *0* | on success |

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

kdbOpen() for opening a session with a Key database

### 572.26.3.4 close() [2/2]

```
void kdb::KDB::close (
            Key & errorKey ) throw ( )   [inline], [virtual]
```

Close the database.
The return value does not matter because its only a null pointer check.

**Parameters**

| | |
|---|---|
| *errorKey* | is useful if you want to get the warnings |

Closes the session with the Key database.

**Precondition**

The handle must be a valid handle as returned from kdbOpen()

errorKey must be a valid key, e.g. created with keyNew()

This is the counterpart of kdbOpen().
You must call this method when you are finished working with the Key database. You can manipulate Key and KeySet objects also after kdbClose(), but you must not use any kdb∗() call afterwards.
The `handle` parameter will be finalized and all resources associated to it will be freed. After a kdbClose(), the `handle` cannot be used anymore.

**Parameters**

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *errorKey* | the key which holds error/warning information |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

kdbOpen() for opening a session with a Key database

#### 572.26.3.5 get() [1/2]

```
int kdb::KDB::get (
            KeySet & returned,
            Key & parentKey )  [inline], [virtual]
```

Get all keys below parentKey inside returned.
Retrieve Keys from the Key database in an atomic and universal way.

**Precondition**

The `handle` must be a valid KDB handle as returned from kdbOpen().

The KeySet `returned` must be a valid KeySet, i.e., constructed with ksNew().

The KeySet `returned` must contain keys only from the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/` or `proc:/` namespaces.

The Key `parentKey` must be a valid Key, i.e., constructed with keyNew().

The Key `parentKey` must not have read-only name, value or metadata.

The Key `parentKey` must use the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/`, `proc:/` or cascading namespace.

If you pass `NULL` or a key with read-only metadata as `parentKey`, kdbGet() will fail immediately without doing anything. If you pass another invalid `parentKey`, or `NULL` as `ks` or `handle`, kdbGet() will set an error on `parentKey` and then return immediately.

**Note**

If you pass a non-NULL `parentKey` with writable metadata, kdbGet() will **always** remove any existing errors and warnings from `parentKey`.

**Warning**

If you later call kdbSet() with the same `handle` you must make sure to pass all keys from `ks`, which you do not want to remove.

**Loadable Namespaces**

Not all namespace can be loaded.

- `spec:/`, `dir:/`, `user:/` and `system:/` can be loaded via kdbGet().
- `proc:/` keys can be loaded via kdbGet(), but are not persisted or cached.
- `default:/` keys can be inserted by kdbGet() but they will always stem from a specification in `spec:/` keys.
- If `ks` contains a key with any other namespace, an error will be returned.

**Parent Key**

The `parentKey` defines which parts of `ks` will be loaded. Everything that is at or below `parentKey` wil be loaded together with any key that shares a backend with such a key. Backends are always loaded as an atomic unit.

**Note**

If `parentKey` is in the cascading namespace, keys of all loadable namespaces (see above) will be loaded. This is generally the recommended approach.

Upon sucessfully returning kdbGet() also sets the value of `parentKey` to the storage identifier used by the backend that contains (or would contain) `parentKey`. For file-based backends this is the absolute path of the underlying file. Other backends may use different identifiers, but it always uniquely identifies the underlying storage unit.

#### Note

If `parentKey` is in the cascading, `default:/` or `proc:/ namespace, the value of `parentKey` will be set to an empty string. This is done, because those namespaces are not persistable (see kdbSet()) and therfore have no storage identifier.

#### KeySet Modifications

Below or at `parentKey`, the KeySet `ks` will mostly contain keys loaded from backends. The only exception are `proc:/` and `spec:/` keys that were already present, before kdbGet() was called and do not overlap with an existing backend (for those namespaces). This can be used to provide a hard-coded fallback specifications and/or process-specific data.
Keys not below (or at) `parentKey` that were present when kdbGet() was called, may still be removed. For example, this could be because they overlap with a backend that also has keys below `parentKey` (backends are atomic units).

#### Example:

This example demonstrates the typical usecase within an application (without error handling).

```
#include <kdb.h>
#include <stdio.h>
int main (void)
{
        KeySet * myConfig = ksNew (0, KS_END);
        // for error handling see kdbget_error.c
        // clang-format off
Key * key = keyNew ("/sw/tests/myapp/#0/current/",  KEY_END);
KDB * handle = kdbOpen (NULL, key);
kdbGet (handle, myConfig, key);
Key * result = ksLookupByName (myConfig, "/sw/tests/myapp/#0/current/testkey1", 0);
        // clang-format on
        keyDel (key);
        const char * key_name = keyName (result);
        const char * key_value = keyString (result);
        const char * key_comment = keyString (keyGetMeta (result, "comment/#0"));
        printf ("key: %s value: %s comment: %s\n", key_name, key_value, key_comment);
        ksDel (myConfig); // delete the in-memory configuration
        // maybe you want kdbSet() myConfig here
        kdbClose (handle, 0); // no more affairs with the key database.
}
```

When a backend fails kdbGet() will return -1 with all error and warning information in the `parentKey`. The parameter `returned` will not be changed.

#### Optimization:

In the first run of kdbGet all requested (or more) Keys are retrieved. On subsequent calls only the Keys are retrieved where something was changed inside the Key database. The other Keys stay in the KeySet returned as passed.

It is your responsibility to save the original KeySet if you need it afterwards.
If you want to be sure to get a fresh KeySet again, you need to open a second handle to the Key database using kdbOpen().

#### Parameters

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *parentKey* | Keys below `parentKey` will be retrieved from `handle`. It is also used to add warnings and set error information. |
| *ks* | the (pre-initialized) KeySet returned with all keys found will not be changed on error or if no update is required |

**Return values**

| | |
|---|---|
| *2* | if only `proc:/` backends were executed. This means no data was loaded from storage. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as errors! |
| *1* | if the Keys were retrieved successfully. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as errors! |
| *0* | if there was no update at all - no changes are made to the KeySet then. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as erorrs! |
| *-1* | on failure - no changes are made to the KeySet then |

**Since**

1.0.0

**See also**

ksLookup(), ksLookupByName() for powerful lookups after the KeySet was retrieved

kdbOpen() which needs to be called before

kdbSet() to save the configuration afterwards

kdbClose() to finish affairs with the Key database.

**Parameters**

| | |
|---|---|
| *returned* | the keyset where the keys will be in |
| *parentKey* | the parentKey of returned |

**Return values**

| | |
|---|---|
| *0* | if no key was updated |
| *1* | if user or system keys were updated |
| *2* | if user and system keys were updated |

**Exceptions**

| | |
|---|---|
| *KDBException* | if there were problems with the database |

Reimplemented in kdb::tools::merging::MergingKDB.

### 572.26.3.6 get() [2/2]

```
int kdb::KDB::get (
            KeySet & returned,
            std::string const & keyname )  [inline], [virtual]
```

Get all keys below keyname inside returned.
Retrieve Keys from the Key database in an atomic and universal way.

**Precondition**

The `handle` must be a valid KDB handle as returned from kdbOpen().

The KeySet `returned` must be a valid KeySet, i.e., constructed with ksNew().

The KeySet `returned` must contain keys only from the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/` or `proc:/` namespaces.

The Key parentKey must be a valid Key, i.e., constructed with keyNew().

The Key parentKey must not have read-only name, value or metadata.

The Key parentKey must use the spec:/, dir:/, user:/, system:/, default:/, proc:/ or cascading namespace.

If you pass NULL or a key with read-only metadata as parentKey, kdbGet() will fail immediately without doing anything. If you pass another invalid parentKey, or NULL as ks or handle, kdbGet() will set an error on parentKey and then return immediately.

**Note**

If you pass a non-NULL parentKey with writable metadata, kdbGet() will **always** remove any existing errors and warnings from parentKey.

**Warning**

If you later call kdbSet() with the same handle you must make sure to pass all keys from ks, which you do not want to remove.

**Loadable Namespaces**

Not all namespace can be loaded.

- spec:/, dir:/, user:/ and system:/ can be loaded via kdbGet().
- proc:/ keys can be loaded via kdbGet(), but are not persisted or cached.
- default:/ keys can be inserted by kdbGet() but they will always stem from a specification in spec:/ keys.
- If ks contains a key with any other namespace, an error will be returned.

**Parent Key**

The parentKey defines which parts of ks will be loaded. Everything that is at or below parentKey wil be loaded together with any key that shares a backend with such a key. Backends are always loaded as an atomic unit.

**Note**

If parentKey is in the cascading namespace, keys of all loadable namespaces (see above) will be loaded. This is generally the recommended approach.

Upon sucessfully returning kdbGet() also sets the value of parentKey to the storage identifier used by the backend that contains (or would contain) parentKey. For file-based backends this is the absolute path of the underlying file. Other backends may use different identifiers, but it always uniquely identifies the underlying storage unit.

**Note**

If parentKey is in the cascading, default:/ or `proc:/ namespace, the value of parentKey will be set to an empty string. This is done, because those namespaces are not persistable (see kdbSet()) and therfore have no storage identifier.

#### KeySet Modifications

Below or at `parentKey`, the KeySet `ks` will mostly contain keys loaded from backends. The only exception are `proc:/` and `spec:/` keys that were already present, before kdbGet() was called and do not overlap with an existing backend (for those namespaces). This can be used to provide a hard-coded fallback specifications and/or process-specific data.
Keys not below (or at) `parentKey` that were present when kdbGet() was called, may still be removed. For example, this could be because they overlap with a backend that also has keys below `parentKey` (backends are atomic units).

**Example:**

This example demonstrates the typical usecase within an application (without error handling).

```
#include <kdb.h>
#include <stdio.h>
int main (void)
{
        KeySet * myConfig = ksNew (0, KS_END);
        // for error handling see kdbget_error.c
        // clang-format off
Key * key = keyNew ("/sw/tests/myapp/#0/current/",  KEY_END);
KDB * handle = kdbOpen (NULL, key);
kdbGet (handle, myConfig, key);
Key * result = ksLookupByName (myConfig, "/sw/tests/myapp/#0/current/testkey1", 0);
        // clang-format on
        keyDel (key);
        const char * key_name = keyName (result);
        const char * key_value = keyString (result);
        const char * key_comment = keyString (keyGetMeta (result, "comment/#0"));
        printf ("key: %s value: %s comment: %s\n", key_name, key_value, key_comment);
        ksDel (myConfig); // delete the in-memory configuration
        // maybe you want kdbSet() myConfig here
        kdbClose (handle, 0); // no more affairs with the key database.
}
```

When a backend fails kdbGet() will return -1 with all error and warning information in the `parentKey`. The parameter `returned` will not be changed.

**Optimization:**

In the first run of kdbGet all requested (or more) Keys are retrieved. On subsequent calls only the Keys are retrieved where something was changed inside the Key database. The other Keys stay in the KeySet returned as passed.

It is your responsibility to save the original KeySet if you need it afterwards.
If you want to be sure to get a fresh KeySet again, you need to open a second handle to the Key database using kdbOpen().

#### Parameters

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *parentKey* | Keys below `parentKey` will be retrieved from `handle`. It is also used to add warnings and set error information. |
| *ks* | the (pre-initialized) KeySet returned with all keys found will not be changed on error or if no update is required |

#### Return values

| | |
|---|---|
| *2* | if only `proc:/` backends were executed. This means no data was loaded from storage. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as errors! |
| *1* | if the Keys were retrieved successfully. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as errors! |
| *0* | if there was no update at all - no changes are made to the KeySet then. There might be warnings attached to the parentKey! Depending on your use case, you might need to treat them as erorrs! |

**Return values**

| | |
|---|---|
| *-1* | on failure - no changes are made to the KeySet then |

**Since**

1.0.0

**See also**

ksLookup(), ksLookupByName() for powerful lookups after the KeySet was retrieved

kdbOpen() which needs to be called before

kdbSet() to save the configuration afterwards

kdbClose() to finish affairs with the Key database.

```
#include <kdb.hpp>
#include <keyio.hpp>
using namespace kdb;
int main ()
{
        KeySet config;
        KDB kdb;
        kdb.get (config, "/sw/MyApp");
        Key k = config.lookup ("/sw/MyApp/mykey");
        if (k)
        {
                std::cout « k « " is " « k.get<int> () « std::endl;
        }
        else
        {
                std::cerr « "No key found" « std::endl;
                return 1;
        }
}
```

**Parameters**

| | |
|---|---|
| *returned* | the keyset where the keys will be in |
| *keyname* | the root keyname which should be used to get keys below it |

**Return values**

| | |
|---|---|
| *0* | if no key was updated |
| *1* | if user or system keys were updated |
| *2* | if user and system keys were updated |

**Exceptions**

| | |
|---|---|
| *KDBException* | if there were problems with the database |

**See also**

KDB::get (KeySet & returned, Key & parentKey)

Reimplemented in kdb::tools::merging::MergingKDB.

### 572.26.3.7 getKdb()

```
ckdb::KDB * kdb::KDB::getKdb ( ) const  [inline]
```

Passes out the raw kdb pointer.
This pointer can be used to directly interact with the underlying kdb instance

**Note**

that the ownership remains in the object

#### 572.26.3.8 open() [1/2]

```
void kdb::KDB::open (
            Key & errorKey )  [inline], [virtual]
```

Open the database.

**Parameters**

| | |
|---|---|
| *errorKey* | is useful if you want to get the warnings in the successful case, when no exception is thrown. |

Opens the session with the Key database.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the KDB data structure will be initialized.
The pointer to the KDB structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a KDB handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

handle to the newly created KDB on success

**Return values**

| | |
|---|---|
| *NULL* | on failure |

**Since**

1.0.0

**See also**

kdbClose() to close the session of a Key database opened by kdbOpen()

### 572.26.3.9 open() [2/2]

```
void kdb::KDB::open (
            KeySet & contract,
            Key & errorKey )  [inline], [virtual]
```

Open the database.

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured |
| *errorKey* | is useful if you want to get the warnings in the successful case, when no exception is thrown. |

Opens the session with the Key database.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

You must always call this method before retrieving or committing any keys to the database. At the end of a program, after using the Key database (KDB), you must not forget to call kdbClose() to free resources.
The method will bootstrap itself in the following way. The first step is to open the default backend. With it `system:/elektra/mountpoints` will be loaded and all needed libraries and mountpoints will be determined. Then the global plugins and global keyset data from the `contract` is processed. Finally, the libraries for backends will be loaded and with it the KDB data structure will be initialized.
The pointer to the KDB structure returned will be initialized like described above, and it must be passed along on any kdb*() method your application calls.
Get a KDB handle for every thread using elektra. Don't share the handle across threads, and also not the pointer accessing it:

```
void thread1 (void)
{
        Key * parent = keyNew ("/app/part1", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
void thread2 (void)
{
        Key * parent = keyNew ("/app/part2", KEY_END);
        KDB * h = kdbOpen (NULL, parent);
        // fetch keys and work with them
        kdbClose (h, parent);
}
```

You don't need kdbOpen() if you only want to manipulate plain in-memory Key or KeySet objects.

**Precondition**

errorKey must be a valid key, e.g. created with keyNew()

**Parameters**

| | |
|---|---|
| *contract* | the contract that should be ensured before opening the KDB all data is copied and the KeySet can safely be used for e.g. kdbGet() later |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

handle to the newly created KDB on success

**Return values**

| | |
|---|---|
| *NULL* | on failure |

**Since**

1.0.0

**See also**

kdbClose() to close the session of a Key database opened by kdbOpen()

### 572.26.3.10 operator∗()

```
ckdb::KDB * kdb::KDB::operator* ( ) const  [inline]
```

Is an abbreviation for getKdb.
Passes out the raw kdb pointer. This pointer can be used to directly interact with the underlying kdb instance

**Note**

that the ownership remains in the object

**See also**

getKdb()

### 572.26.3.11 set() [1/2]

```
int kdb::KDB::set (
            KeySet & returned,
            Key & parentKey )  [inline], [virtual]
```

Set all keys below parentKey.
If the keyname of the parentKey is invalid (e.g. empty) all keys will be set.
Set Keys to the Key database in an atomic and universal way.

**Precondition**

kdbGet() must be called before kdbSet():

- initially (after kdbOpen())
- after conflict errors in kdbSet().

The KeySet `ks` must be a valid KeySet, i.e., constructed with ksNew().

The KeySet `ks` must only contain only keys in the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/` or `proc:/` namespaces.

The Key `parentKey` must be a valid Key, e.g. constructed with keyNew().

The Key parentKey must not have read-only name, value or metadata.

The Key parentKey must use the spec:/, dir:/, user:/, system:/, default:/, proc:/ or cascading namespace.

If you pass NULL or a key with read-only metadata as parentKey, kdbSet() will fail immediately without doing anything. If you pass another invalid parentKey, or NULL as ks or handle, kdbSet() will set an error on parentKey and then return immediately.

**Note**

If you pass a non-NULL parentKey with writable metadata, kdbSet() will **always** remove any existing errors and warnings from parentKey.

#### Persistable Namespaces

Not all namespace can be persisted.

- spec:/, dir:/, user:/ and system:/ will be persisted by kdbSet().
- default:/ and proc:/ keys are ignored by kdbSet().
- If ks contains a key with any other namespace, an error will be returned.

In general it is recommended to use a parentKey in the cascading namespace to cover all namespaces at once.

#### Parent Key

The parentKey defines which parts of ks will be stored. Everything that is at or below parentKey will be persisted together with any key that shares a backend with such a key. Backends are always stored as an atomic unit.

**Note**

If parentKey is in the cascading namespace, keys of all persistable namespaces (see above) will be stored. This is generally the recommended approach.

#### KeySet modifications

The contents of ks will mostly not be modified by kdbSet(). The only modifications made are those caused by applying the specification in spec:/ to dir:/, user:/ and system:/.

**Errors**

If parentKey == NULL or parentKey has read-only metadata, kdbSet() will immediately return the error code -1. In all other error cases the following happens:

- Error information will be written into the metadata of the parent key, if possible.
- None of the keys are actually committed in this situation, i.e. no configuration file will be modified.

In case of errors you should present the error message to the user and let the user decide what to do. Possible solutions are:

- remove the problematic key and use kdbSet() again (for validation or type errors)
- change the value of the problematic key and use kdbSet() again (for validation errors)
- do a kdbGet() (for conflicts, i.e. error C02000) and then
  - set the same keyset again (in favour of what was set by this user)

  - drop the old keyset (in favour of what was set from another application)
  - merge the original, your own and the other keyset
- export the configuration into a file (for unresolvable errors)
- repeating the same kdbSet() might be of limited use, if the user does not explicitly request it, because temporary errors are rare and its unlikely that they fix themselves (e.g. disc full, permission problems)

#### Optimization

Only backends that

- contain at least one changed key according to elektraDiffCalculate(),
- contain fewer keys than at the end of kdbGet() will be called. There won't be an unnecessary write for unchanged keys.

If none of the backends needs an update, kdbSet() returns 0 and does nothing.

```
KeySet * myConfig = ksNew (0, KS_END);
Key * parentKey = keyNew ("system:/sw/MyApp", KEY_END);
KDB * handle = kdbOpen (NULL, parentKey);
kdbGet (handle, myConfig, parentKey); // kdbGet needs to be called first!
KeySet * base = ksDup (myConfig);     // save a copy of original keyset
// change the keys within myConfig
ksAppendKey (myConfig, keyNew ("system:/sw/MyApp/Test", KEY_VALUE, "5", KEY_END));
KeySet * ours = ksDup (myConfig); // save a copy of our keyset
KeySet * theirs;                  // needed for 3-way merging
int ret = kdbSet (handle, myConfig, parentKey);
while (ret == -1) // as long as we have an error
{
        int strategy = showElektraErrorDialog (parentKey);
        theirs = ksDup (ours);
        kdbGet (handle, theirs, parentKey); // refresh key database
        KeySet * result = elektraMerge(
                ksCut(ours, parentKey), parentKey,
                ksCut(theirs, parentKey), parentKey,
                ksCut(base, parentKey), parentKey,
                parentKey, strategy, parentKey);
        int numberOfConflicts = elektraMergeGetConflicts (parentKey);
        ksDel (theirs);
        if (result != NULL) {
                ret = kdbSet (handle, result, parentKey);
        } else {
                // an error happened while merging
                if (numberOfConflicts > 0 && strategy == MERGE_STRATEGY_ABORT)
                {
                        // Error due to merge conflicts
                        ret = -1;
                }
                else
                {
                        // Internal errors, out of memory etc.
                        ret = -1;
                }
        }
}
ksDel (ours);
ksDel (base);
ksDel (myConfig); // delete the in-memory configuration
kdbClose (handle, parentKey); // no more affairs with the key database.
keyDel (parentKey);
```

showElektraErrorDialog() and doElektraMerge() need to be implemented by the user of Elektra. For doElektraMerge a 3-way merge algorithm exists in libelektra-tools.

#### Parameters

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *ks* | a KeySet which should contain changed keys, otherwise nothing is done |
| *parentKey* | Keys below `parentKey` will be set to `handle`. It is also used to add warnings and set error information. |

#### Return values

| | |
|---|---|
| *1* | on success |

**Return values**

| | |
|---|---|
| *0* | if nothing had to be done, no changes in KDB |
| *-1* | on failure, no changes in KDB, an error will be set on `parentKey` if possible (see "Errors" above) |

**Since**

1.0.0

**See also**

kdbOpen() for getting `handle`

kdbClose() that must be called afterwards

ksCurrent() contains the error Key

**Return values**

| | |
|---|---|
| *0* | if no key was updated |
| *1* | if user or system keys were updated |
| *2* | if user and system keys were updated |

**Parameters**

| | |
|---|---|
| *returned* | the keyset where the keys are passed to the user |
| *parentKey* | the parentKey of returned |

**Exceptions**

| | |
|---|---|
| *KDBException* | if there were problems with the database |

### 572.26.3.12 set() [2/2]

```
int kdb::KDB::set (
            KeySet & returned,
            std::string const & keyname )  [inline], [virtual]
```

Set all keys below keyname.
If the keyname of the parentKey is invalid (e.g. empty) all keys will be set.
Set Keys to the Key database in an atomic and universal way.

**Precondition**

kdbGet() must be called before kdbSet():

- initially (after kdbOpen())
- after conflict errors in kdbSet().

The KeySet `ks` must be a valid KeySet, i.e., constructed with ksNew().

The KeySet `ks` must only contain only keys in the `spec:/`, `dir:/`, `user:/`, `system:/`, `default:/` or `proc:/` namespaces.

The Key `parentKey` must be a valid Key, e.g. constructed with keyNew().

The Key `parentKey` must not have read-only name, value or metadata.

The Key parentKey must use the spec:/, dir:/, user:/, system:/, default:/, proc:/ or cascading namespace.

If you pass NULL or a key with read-only metadata as parentKey, kdbSet() will fail immediately without doing anything. If you pass another invalid parentKey, or NULL as ks or handle, kdbSet() will set an error on parentKey and then return immediately.

**Note**

If you pass a non-NULL parentKey with writable metadata, kdbSet() will **always** remove any existing errors and warnings from parentKey.

**Persistable Namespaces**

Not all namespace can be persisted.

- spec:/, dir:/, user:/ and system:/ will be persisted by kdbSet().
- default:/ and proc:/ keys are ignored by kdbSet().
- If ks contains a key with any other namespace, an error will be returned.

In general it is recommended to use a parentKey in the cascading namespace to cover all namespaces at once.

**Parent Key**

The parentKey defines which parts of ks will be stored. Everything that is at or below parentKey will be persisted together with any key that shares a backend with such a key. Backends are always stored as an atomic unit.

**Note**

If parentKey is in the cascading namespace, keys of all persistable namespaces (see above) will be stored. This is generally the recommended approach.

**KeySet modifications**

The contents of ks will mostly not be modified by kdbSet(). The only modifications made are those caused by applying the specification in spec:/ to dir:/, user:/ and system:/.

**Errors**

If parentKey == NULL or parentKey has read-only metadata, kdbSet() will immediately return the error code -1. In all other error cases the following happens:

- Error information will be written into the metadata of the parent key, if possible.
- None of the keys are actually committed in this situation, i.e. no configuration file will be modified.

In case of errors you should present the error message to the user and let the user decide what to do. Possible solutions are:

- remove the problematic key and use kdbSet() again (for validation or type errors)
- change the value of the problematic key and use kdbSet() again (for validation errors)
- do a kdbGet() (for conflicts, i.e. error C02000) and then
  - set the same keyset again (in favour of what was set by this user)
  - drop the old keyset (in favour of what was set from another application)
  - merge the original, your own and the other keyset
- export the configuration into a file (for unresolvable errors)
- repeating the same kdbSet() might be of limited use, if the user does not explicitly request it, because temporary errors are rare and its unlikely that they fix themselves (e.g. disc full, permission problems)

### Optimization

Only backends that

- contain at least one changed key according to elektraDiffCalculate(),
- contain fewer keys than at the end of kdbGet() will be called. There won't be an unnecessary write for unchanged keys.

If none of the backends needs an update, kdbSet() returns 0 and does nothing.

```
KeySet * myConfig = ksNew (0, KS_END);
Key * parentKey = keyNew ("system:/sw/MyApp", KEY_END);
KDB * handle = kdbOpen (NULL, parentKey);
kdbGet (handle, myConfig, parentKey); // kdbGet needs to be called first!
KeySet * base = ksDup (myConfig);     // save a copy of original keyset
// change the keys within myConfig
ksAppendKey (myConfig, keyNew ("system:/sw/MyApp/Test", KEY_VALUE, "5", KEY_END));
KeySet * ours = ksDup (myConfig); // save a copy of our keyset
KeySet * theirs;                  // needed for 3-way merging
int ret = kdbSet (handle, myConfig, parentKey);
while (ret == -1) // as long as we have an error
{
        int strategy = showElektraErrorDialog (parentKey);
        theirs = ksDup (ours);
        kdbGet (handle, theirs, parentKey); // refresh key database
        KeySet * result = elektraMerge(
                ksCut(ours, parentKey), parentKey,
                ksCut(theirs, parentKey), parentKey,
                ksCut(base, parentKey), parentKey,
                parentKey, strategy, parentKey);
        int numberOfConflicts = elektraMergeGetConflicts (parentKey);
        ksDel (theirs);
        if (result != NULL) {
                ret = kdbSet (handle, result, parentKey);
        } else {
                // an error happened while merging
                if (numberOfConflicts > 0 && strategy == MERGE_STRATEGY_ABORT)
                {
                        // Error due to merge conflicts
                        ret = -1;
                }
                else
                {
                        // Internal errors, out of memory etc.
                        ret = -1;
                }
        }
}
ksDel (ours);
ksDel (base);
ksDel (myConfig); // delete the in-memory configuration
kdbClose (handle, parentKey); // no more affairs with the key database.
keyDel (parentKey);
```

showElektraErrorDialog() and doElektraMerge() need to be implemented by the user of Elektra. For doElektraMerge a 3-way merge algorithm exists in libelektra-tools.

### Parameters

| | |
|---|---|
| *handle* | contains internal information of opened key database |
| *ks* | a KeySet which should contain changed keys, otherwise nothing is done |
| *parentKey* | Keys below `parentKey` will be set to `handle`. It is also used to add warnings and set error information. |

### Return values

| | |
|---|---|
| *1* | on success |
| *0* | if nothing had to be done, no changes in KDB |
| *-1* | on failure, no changes in KDB, an error will be set on `parentKey` if possible (see "Errors" above) |

**Since**

1.0.0

**See also**

kdbOpen() for getting `handle`

kdbClose() that must be called afterwards

ksCurrent() contains the error Key

**Return values**

| | |
|---|---|
| *0* | if no key was updated |
| *1* | if user or system keys were updated |
| *2* | if user and system keys were updated |

**Parameters**

| | |
|---|---|
| *returned* | the keyset where the keys will be in |
| *keyname* | the keyname below the names should be set |

**Exceptions**

| | |
|---|---|
| *KDBException* | if there were problems with the database |

The documentation for this class was generated from the following file:

- kdb.hpp

## 572.27 org.libelektra.KDB Class Reference

Represents a session with the Elektra key database.
Inherits AutoCloseable.

### Public Member Functions

- void close () throws KDBException

  *Closes the KDB session and frees native resources associated with it.*
- void close (Key warningsKey) throws KDBException

  *Closes the KDB session and frees native resources associated with it.*
- KeySet get (Key parentKey) throws KDBException

  *Fetches at least all keys that are sub-keys or children of sub-keys of the supplied parent key.*
- KDB get (KeySet keySet, Key parentKey) throws KDBException

  *Fetches at least all keys that are sub-keys or children of sub-keys of the supplied parent key.*
- KDB set (KeySet keySet, Key parentKey) throws KDBException

  *Will update changed keys of the given.*

### Static Public Member Functions

- static KDB open () throws KDBException

  *Opens a new KDB session.*
- static KDB open (KeySet contract) throws KDBException

*Opens KDB session using the specified.*

- static KDB open (Key warningsKey) throws KDBException

  *Opens a new KDB session.*
- static KDB open (KeySet contract, Key warningsKey) throws KDBException

  *Opens KDB session using the specified.*
- static KeySet goptsContract (String[ ] args, String[ ] env, Key parentKey, KeySet goptsConfig)

  *Creates a contract KeySet for use with KDB#open(KeySet) that mounts and configures the.*
- static void goptsContract (KeySet contract, String[ ] args, String[ ] env, Key parentKey, KeySet goptsConfig)

  *Writes a contract into a specified KeySet for use with KDB#open(KeySet) that mounts and configures the.*

## Protected Member Functions

- Pointer getPointer ()

## 572.27.1 Detailed Description

Represents a session with the Elektra key database.
@apiNote Close after usage, or simply use a try-with-resources statement

## 572.27.2 Member Function Documentation

### 572.27.2.1 close() [1/2]

```
void org.libelektra.KDB.close ( ) throws KDBException  [inline]
```

Closes the KDB session and frees native resources associated with it.

**Exceptions**

| | |
|---|---|
| *KDBException* | if opening the session fails - see specialization of KDBException |
| *KDBClosedException* | if this session has already been closed |

### 572.27.2.2 close() [2/2]

```
void org.libelektra.KDB.close (
            Key warningsKey ) throws KDBException  [inline]
```

Closes the KDB session and frees native resources associated with it.

**Parameters**

| | |
|---|---|
| *warningsKey* | Used to store warnings, which may occur during closing the session, in this key's meta data |

**Exceptions**

| | |
|---|---|
| *KDBException* | if opening the session fails - see specialization of KDBException |
| *KDBClosedException* | if this session has already been closed |
| *IllegalStateException* | if `parentKey` has already been released |
| *IllegalArgumentException* | if `warningsKey` is `null` |

**See also**

Key::create()

### 572.27.2.3 get() [1/2]

```
KeySet org.libelektra.KDB.get (
            Key parentKey ) throws KDBException  [inline]
```

Fetches at least all keys that are sub-keys or children of sub-keys of the supplied parent key.
Note: Resulting key set may contain more keys than requested

**Parameters**

| | |
|---|---|
| *parentKey* | Root key which name is used to fetch keys below. This key is also used to store warnings, which may occur during the operation, in this key's meta data. |

**Returns**

New KeySet containing the fetched keys

**Exceptions**

| | |
|---|---|
| *KDBException* | if loading keys fails - see specialization of KDBException |
| *KDBClosedException* | if this session has already been closed |
| *IllegalStateException* | if `parentKey` has already been released |
| *IllegalArgumentException* | |

### 572.27.2.4 get() [2/2]

```
KDB org.libelektra.KDB.get (
            KeySet keySet,
            Key parentKey ) throws KDBException  [inline]
```

Fetches at least all keys that are sub-keys or children of sub-keys of the supplied parent key.
Note: Resulting key set may contain more keys than requested

**Parameters**

| | |
|---|---|
| *keySet* | KeySet used to store the fetched keys |
| *parentKey* | Root key which name is used to fetch keys below it. This key is also used to store warnings, which may occur during the operation, in this key's meta data. It is recommended to use the most specific `parentKey` possible. (e.g. using `system:/` is rarely the most specific) |

**Returns**

This KDB session, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *KDBException* | if loading keys fails - see specialization of KDBException |
| *KDBClosedException* | if this session has already been closed |
| *IllegalStateException* | if `keySet` or `parentKey` has already been released |
| *IllegalArgumentException* | if `keySet` or `parentKey` is `null` |

**See also**

get(Key)

### 572.27.2.5 getPointer()

```
Pointer org.libelektra.KDB.getPointer ( )  [inline], [protected]
```

**Returns**

JNA pointer to the native pointer for this key set

**Exceptions**

| | |
|---|---|
| *KDBClosedException* | if this KDB session has already been closed |

### 572.27.2.6 goptsContract() [1/2]

```
static void org.libelektra.KDB.goptsContract (
            KeySet contract,
            String[] args,
            String[] env,
            Key parentKey,
            KeySet goptsConfig )  [inline], [static]
```

Writes a contract into a specified KeySet for use with KDB#open(KeySet) that mounts and configures the. `gopts` plugin

**Parameters**

| | |
|---|---|
| *contract* | Key set to write the contract to |
| *args* | Arguments that will be converted into `argc` and `argv` for `gopts` |
| *env* | Environment variables that `gopts` will use |

**Parameters**

| | |
|---|---|
| *parentKey* | Parent key that should be used by `gopts`. Only the key name is copied. The key can be deleted immediately after calling this function. |
| *goptsConfig* | Config used for mounting the `gopts` plugin |

**Exceptions**

| | |
|---|---|
| *IllegalArgumentException* | if any of the arguments are `null` |
| *IllegalStateException* | if `contract`, `goptsConfig` or `parentKey` has already been released |
| *IllegalArgumentException* | if any of the specified parameters is `null` |

### 572.27.2.7 goptsContract() [2/2]

```
static KeySet org.libelektra.KDB.goptsContract (
            String[] args,
            String[] env,
            Key parentKey,
            KeySet goptsConfig )  [inline], [static]
```

Creates a contract KeySet for use with KDB#open(KeySet) that mounts and configures the. `gopts` plugin

**Parameters**

| | |
|---|---|
| *args* | Arguments that will be converted into `argc` and `argv` for `gopts` |
| *env* | Environment variables that `gopts` will use |
| *parentKey* | Parent key that should be used by `gopts`. Only the key name is copied. The key can be deleted immediately after calling this function. |
| *goptsConfig* | Config used for mounting the `gopts` plugin |

**Returns**

New KeySet containing the contract

**Exceptions**

| | |
|---|---|
| *IllegalArgumentException* | if any of the arguments are `null` |
| *IllegalStateException* | if `goptsConfig` or `parentKey` has already been released |
| *IllegalArgumentException* | if any of the specified parameters is `null` |

#### 572.27.2.8 open() [1/4]

```
static KDB org.libelektra.KDB.open ( ) throws KDBException  [inline], [static]
```

Opens a new KDB session.

**Returns**

New KDB session

**Exceptions**

| | |
|---|---|
| *KDBException* | if opening the session fails - see specialization of KDBException |

#### 572.27.2.9 open() [2/4]

```
static KDB org.libelektra.KDB.open (
            Key warningsKey ) throws KDBException  [inline], [static]
```

Opens a new KDB session.

**Parameters**

| | |
|---|---|
| *warningsKey* | Used to store warnings, which may occur during opening the session, in this key's meta data |

**Returns**

New KDB session

**Exceptions**

| | |
|---|---|
| *KDBException* | if opening the session fails - see specialization of KDBException |
| *IllegalStateException* | if `warningsKey` has already been released |
| *IllegalArgumentException* | if `warningsKey` is `null` |

**See also**

Key::create()

### 572.27.2.10 open() [3/4]

```
static KDB org.libelektra.KDB.open (
            KeySet contract ) throws KDBException  [inline], [static]
```

Opens KDB session using the specified.
`contract`

**Parameters**

| | |
|---|---|
| *contract* | Contract configuring the `gopts` plugin |

**Returns**

New KDB session

**Exceptions**

| | |
|---|---|
| *KDBException* | if opening the session fails - see specialization of KDBException |
| *IllegalStateException* | if `contract` has already been released |
| *IllegalArgumentException* | if `contract` is `null` |

**See also**

goptsContract(String[ ], String[ ], Key, KeySet)

goptsContract(KeySet, String[ ], String[ ], Key, KeySet)

### 572.27.2.11 open() [4/4]

```
static KDB org.libelektra.KDB.open (
            KeySet contract,
            Key warningsKey ) throws KDBException  [inline], [static]
```

Opens KDB session using the specified.
`contract`

**Parameters**

| | |
|---|---|
| *contract* | Contract configuring the `gopts` plugin |
| *warningsKey* | Used to store warnings, which may occur during opening the session, in this key's meta data |

**Returns**

New KDB session

**Exceptions**

| | |
|---|---|
| *KDBException* | if opening the session fails - see specialization of KDBException |
| *IllegalStateException* | if `contract` or `warningsKey` has already been released |
| *IllegalArgumentException* | if `contract` or `warningsKey` is `null` |

**See also**

goptsContract(String[ ], String[ ], Key, KeySet)

goptsContract(KeySet, String[ ], String[ ], Key, KeySet)

### 572.27.2.12 set()

```
KDB org.libelektra.KDB.set (
            KeySet keySet,
            Key parentKey ) throws KDBException  [inline]
```

Will update changed keys of the given.
`keySet`
in the backend. get(Key) or get(KeySet, Key) has to be called before this function may be executed.

**Parameters**

| | |
|---|---|
| *keySet* | KeySet which contains keys to be updated in the backend |
| *parentKey* | Specify which part of the given `keySet` is of interest for you. This key is also used to store warnings, which may occur during the operation, in this key's meta data. In general it is highly recommended, that you use the same `parentKey` used to fetch the `keySet` with get(Key) or get(KeySet, Key). You promise to only modify or remove keys below this key. All others would be passed back as they were retrieved by `keySet` with get(Key). Cascading keys (starting with `/`) will set the path in all namespaces. A nameless key as created by {} will commit all changes in the keySet . This parameter is an optimization to only save keys of mountpoints affected b |

The documentation for this class was generated from the following file:

- KDB.java

## 572.28 org.libelektra.exception.KDBClosedException Class Reference

Indicates that an already closed KDB session has been accessed.

Inherits IllegalStateException.

### 572.28.1 Detailed Description

Indicates that an already closed KDB session has been accessed.
The documentation for this class was generated from the following file:

- KDBClosedException.java

## 572.29 org.libelektra.KDBException Class Reference

Wraps Elektra errors into the corresponding Java exceptions.
Inherits Exception.
Inherited by org.libelektra.exception.ConflictingStateException, org.libelektra.exception.PermanentException, and org.libelektra.exception.ValidationException.

### Public Member Functions

- String getErrorNumber ()
- String getConfigFile ()
- String getMountpoint ()
- String getDebugInformation ()
- String getModule ()
- String getReason ()
- String getMessage ()
- boolean hasWarnings ()

  *If an error occurred it may also has important warnings which caused the error.*
- List< WarningEntry > getWarnings ()

### Static Public Member Functions

- static KDBException getMappedException (Key errorKey)

  *Extracts warnings and error information and maps it to an appropriate exception.*

### Protected Member Functions

- KDBException (Key errorKey)

### 572.29.1 Detailed Description

Wraps Elektra errors into the corresponding Java exceptions.

### 572.29.2 Constructor & Destructor Documentation

#### 572.29.2.1 KDBException()

```
org.libelektra.KDBException.KDBException (
            Key errorKey )  [inline], [protected]
```

**Parameters**

| | |
|---|---|
| *errorKey* | Key containing `error/*` and `warnings/*` meta keys |

## 572.29.3 Member Function Documentation

### 572.29.3.1 getConfigFile()

```
String org.libelektra.KDBException.getConfigFile ( )  [inline]
```

**Returns**

The affected configuration file of the error or if not available returns the error key name

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

### 572.29.3.2 getDebugInformation()

```
String org.libelektra.KDBException.getDebugInformation ( )  [inline]
```

**Returns**

Elektra specific debug information in the form of "At: file:line"

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

### 572.29.3.3 getErrorNumber()

```
String org.libelektra.KDBException.getErrorNumber ( )  [inline]
```

**Returns**

Elektra error number read from the error key backing this exception

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

### 572.29.3.4 getMappedException()

```
static KDBException org.libelektra.KDBException.getMappedException (
            Key errorKey )  [inline], [static]
```

Extracts warnings and error information and maps it to an appropriate exception.

**Parameters**

| | |
|---|---|
| *errorKey* | Key containing `error/*` and `warnings/*` meta keys |

**Returns**

KDBException corresponding to the error information

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this `errorKey` has already been released |
| *IllegalArgumentException* | if `errorKey` is `null` |

#### 572.29.3.5 getMessage()

```
String org.libelektra.KDBException.getMessage ( )  [inline]
```

**Returns**

The complete error information in a String with config file, mount point and debug information as it would be printed in the terminal

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

#### 572.29.3.6 getModule()

```
String org.libelektra.KDBException.getModule ( )  [inline]
```

**Returns**

Module which issued the error

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

#### 572.29.3.7 getMountpoint()

```
String org.libelektra.KDBException.getMountpoint ( )  [inline]
```

**Returns**

Mountpoint of the configuration

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

#### 572.29.3.8 getReason()

```
String org.libelektra.KDBException.getReason ( )  [inline]
```

**Returns**

Error reason read from the error key backing this exception

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this error key backing this KDBException has already been released |

#### 572.29.3.9 getWarnings()

```
List<WarningEntry> org.libelektra.KDBException.getWarnings ( )  [inline]
```

**Returns**

Additional warnings emitted with the error

#### 572.29.3.10 hasWarnings()

```
boolean org.libelektra.KDBException.hasWarnings ( )  [inline]
```

If an error occurred it may also has important warnings which caused the error.

**Returns**

True if additional warnings were emitted, false otherwise

The documentation for this class was generated from the following file:

- KDBException.java

## 572.30 kdb::Key Class Reference

Key is an essential class that encapsulates key name , value and metainfo .

`#include <key.hpp>`

### Public Member Functions

- Key ()
  *Constructs a key with the name /.*
- Key (ckdb::Key *k)
  *Constructs a key out of a C key.*
- Key (Key &k)
  *Takes a reference of another key.*
- Key (Key const &k)
  *Takes a reference of another key.*
- Key (const char *keyName,...)
  *A practical way to fully create a Key object in one step.*
- Key (const std::string keyName,...)
  *A practical way to fully create a Key object in one step.*
- Key (const char *keyName, va_list ap)
  *A practical way to fully create a Key object in one step.*
- void operator++ (int) const

*Increment the reference counter of a Key object.*

- void operator++ () const

  *Increment the reference counter of a Key object.*
- void operator-- (int) const

  *Decrement the reference counter of a Key object.*
- void operator-- () const

  *Decrement the reference counter of a Key object.*
- uint16_t getReferenceCounter () const

  *Return the current reference counter value of a Key object.*
- Key & operator= (ckdb::Key *k)

  *Assign a C key.*
- Key & operator= (const Key &k)

  *Assign a key.*
- void copy (const Key &other, elektraCopyFlags flags=KEY_CP_ALL)

  *Copy or clear a key.*
- void clear ()

  *Clears/Invalidates a key.*
- Key * operator-> ()
- ckdb::Key * getKey () const

  *Passes out the raw key pointer.*
- ckdb::Key * operator* () const

  *Is an abbreviation for getKey.*
- ckdb::Key * release ()

  *Passes out the raw key pointer and resets internal key handle.*
- Key dup (elektraCopyFlags flags=KEY_CP_ALL) const
- ~Key ()

  *Destructs the key.*
- std::string getName () const

  *Returns a pointer to the abbreviated real internal `key` name.*
- ssize_t getNameSize () const

  *Bytes needed to store the Key's name (excluding owner).*
- std::string getBaseName () const

  *Returns a pointer to the unescaped Key's name where the basename starts.*
- ssize_t getBaseNameSize () const

  *Calculates number of bytes needed to store basename of `key` (including NULL terminator).*
- void setName (const std::string &newName)

  *Set a new name to a Key.*
- void setBaseName (const std::string &baseName)

  *Sets a base name for a key.*
- void addBaseName (const std::string &baseName)

  *Adds a base name for a key.*
- void delBaseName ()

  *Delete the baseName of a key.*
- bool operator== (const Key &k) const

  *Compare the name of two Keys.*
- bool operator!= (const Key &k) const

  *Compare the name of two Keys.*
- bool operator< (const Key &other) const

  *Compare the name of two Keys.*
- bool operator<= (const Key &other) const

  *Compare the name of two Keys.*

- bool operator> (const Key &other) const

  *Compare the name of two Keys.*
- bool operator>= (const Key &other) const

  *Compare the name of two Keys.*
- bool isNull () const

  *Checks if C++ wrapper has an underlying key.*
- operator bool () const

  *This is for loops and lookups only.*
- template<class T >
  T get () const

  *Get a key value.*
- template<class T >
  void set (T x)

  *Set a key value.*
- std::string getString () const
- void setString (const char *newString)

  *Set the value for `key` as `newStringValue`.*
- ssize_t getStringSize () const

  *Returns the number of bytes needed to store the key value, including the NULL terminator.*
- func_t getFunc () const

  *Elektra can store function pointers as binary.*
- const void * getValue () const

  *Return a pointer to the real internal `key` value.*
- std::string getBinary () const

  *Copy the binary value of a Key into `returnedBinary`.*
- ssize_t getBinarySize () const

  *Returns the number of bytes needed to store the key value, including the NULL terminator.*
- ssize_t setBinary (const void *newBinary, size_t dataSize)

  *Set the value of a Key to the binary value `newBinary`.*
- bool hasMeta (const std::string &metaName) const
- template<class T >
  T getMeta (const std::string &metaName) const

  *Returns the Key for a metadata entry with name `metaName`.*
- template<class T >
  void setMeta (const std::string &metaName, T x)

  *Set metadata for key.*
- void delMeta (const std::string &metaName)

  *Delete metadata for key.*
- void copyMeta (const Key &other, const std::string &metaName)

  *Do a shallow copy of metadata with name `metaName` from source to dest.*
- void copyAllMeta (const Key &other)

  *Do a shallow copy of all metadata from source to dest.*
- bool isValid () const
- ElektraNamespace getNamespace () const
- ssize_t setNamespace (ElektraNamespace ns) const

  *Set the namespace of the key.*
- bool isCascading () const

  *Determines if the key is in cascading namespace.*
- bool isSpec () const

  *Determines if the key is in spec namespace.*
- bool isProc () const

*Determines if the key is in proc namespace.*

- bool isDir () const

  *Determines if the key is in dir namespace.*

- bool isUser () const

  *Determines if the key is in user namespace.*

- bool isSystem () const

  *Determines if the key is in system namespace.*

- bool isString () const

  *Check if the value of `key` is of string type.*

- bool isBinary () const

  *Check if the value of a `key` is of binary type.*

- bool isBelow (std::string const &name) const

  *Check if the Key `check` is below the Key `key` or not.*

- bool isBelow (const Key &k) const

  *Check if the Key `check` is below the Key `key` or not.*

- bool isBelowOrSame (std::string const &name) const

  *Check if a key is below or same.*

- bool isBelowOrSame (const Key &k) const

  *Check if a key is below or same.*

- bool isDirectBelow (std::string const &name) const

  *Check whether the Key `check` is directly below the Key `key`.*

- bool isDirectBelow (const Key &k) const

  *Check whether the Key `check` is directly below the Key `key`.*

- bool isNameLocked () const
- bool isValueLocked () const
- bool isMetaLocked () const

### 572.30.1 Detailed Description

Key is an essential class that encapsulates key name , value and metainfo .
To use it include:

```
#include <kdb.h>
```

Key properties are:

- Key name
- Key value
- Key metadata , including but not limited to:
  - Key comment
  - Key owner
  - UID, GID and filesystem-like mode permissions
  - Mode, change and modification times

#### ABI

Due to ABI compatibility, the `Key` structure is not defined in kdb.h, only declared. So you can only declare `pointers` to `Keys` in your program, and allocate and free memory for them with keyNew() and keyDel() respectively.

#### Reference Counting

Every key has its reference counter (see keyGetRef() for longer explanation) that will be initialized with 0, that means a subsequent call of keyDel() will delete the key. If you append the key to a keyset the reference counter will be incremented by one (see keyIncRef()) and the key can't be deleted by a keyDel().

As you can imagine this refcounting allows you to put the Key in your own data structures. It can be a very powerful feature, e.g. if you need your own-defined ordering or different Models of your configuration.

**Copy-On-Write**

Keys employ copy-on-write techniques to minimize memory footprint. If keys are copied or duplicated, they will point at the same name and value as the source key. Only if this data is changed, additional memory is allocated.

This class is an wrapper for an optional, refcounted ckdb::Key. It is like an shared_ptr<ckdb::Key>, but the shared_ptr functionality is already within the Key and exposed with this wrapper.

**optional**

A key can be constructed with an null pointer, by using Key (static_cast<ckdb::Key*>(0)); or made empty afterwards by using release() or assign a null key. To check if there is an associated managed object the user can use isNull().

**references**

Copies of keys are cheap because they are only flat. If you really need a deep copy, you can use copy() or dup(). If you release() an object, the reference counter will stay All other operations operate on references.

**documentation**

Note that the documentation is typically copied from the underlying function which is wrapped and sometimes extended with C++ specific details. So you might find C examples within the C++ documentation.

**Invariant**

Key either has a working underlying Elektra Key object or a null pointer. The Key, however, might be invalid (see isValid()) or null (see isNull()).

**Note**

that the reference counting in the keys is mutable, so that const keys can be passed around by value.

## 572.30.2 Constructor & Destructor Documentation

### 572.30.2.1 Key() [1/7]

```
kdb::Key::Key ( )  [inline]
```

Constructs a key with the name /.

**Note**

That this is not a null key, so the key will evaluate to true.

**See also**

isValid(), isNull()

### 572.30.2.2 Key() [2/7]

```
kdb::Key::Key (
            ckdb::Key * k )  [inline]
```

Constructs a key out of a C key.

**Note**

If you pass a null pointer here, the key will evaluate to false.

**Parameters**

| | |
|---|---|
| *k* | the key to work with |

**See also**

isValid(), isNull()

### 572.30.2.3 Key() [3/7]

```
kdb::Key::Key (
            Key & k )  [inline]
```

Takes a reference of another key.
The key will not be copied, but the reference counter will be increased.

**Parameters**

| | |
|---|---|
| *k* | the key to work with |

### 572.30.2.4 Key() [4/7]

```
kdb::Key::Key (
            Key const & k )  [inline]
```

Takes a reference of another key.
The key will not be copied, but the reference counter will be increased.

**Parameters**

| | |
|---|---|
| *k* | the key to work with |

### 572.30.2.5 Key() [5/7]

```
kdb::Key::Key (
            const char * keyName,
             ...  )  [inline], [explicit]
```

A practical way to fully create a Key object in one step.
To just get a key object, simple do:

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/some/example", KEY_END);
// work with it
keyDel (k);
```

keyNew() allocates memory for a key object and keyDel() cleans everything up.
If you want the key object to contain a name, value, comment and other meta info read on.

**Note**

When you already have a key with similar properties its easier to keyDup() the key.

You can call keyNew() in many different ways depending on the attribute tags you pass as parameters. Tags are represented as elektraKeyFlags values, and tell keyNew() which Key attribute comes next. The Key attribute tags are the following:

- KEY_VALUE
  Next parameter is a pointer to the value that will be used. If no KEY_BINARY was used before, a string is assumed.

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex0",
        KEY_VALUE, "some data",     // set a string value
        KEY_END);                   // end of args
```

- KEY_SIZE
  Define a maximum length of the value. This is only used when setting a binary key.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex1",
        KEY_SIZE, 4,                // has no effect on strings
        KEY_VALUE, "some data",     // set a string value
        KEY_END);                   // end of args
```

- KEY_META
  Next two parameter is a metaname and a metavalue. See keySetMeta().
```
Key *k=keyNew("user:/tmp/ex3",
        KEY_META, "comment/#0", "a comment",  // with a comment
        KEY_META, "owner", "root",          // and an owner
        KEY_META, "special", "yes",         // and any other metadata
        KEY_END);                   // end of args
```

- KEY_END
  Must be the last parameter passed to keyNew(). It is always required, unless the `keyName` is 0.

- KEY_FLAGS
  Bitwise disjunction of flags, which don't require one or more values. recommended way to set multiple flags. overrides previously defined flags.
```
Key *k=keyNew("user:/tmp/ex3",
        KEY_BINARY,                   // binary key
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",                 // value that will be truncated in 7 bytes
        KEY_END);                       // end of args
```

- KEY_BINARY
  Allows one to change the key to a binary key. Make sure that you also pass KEY_SIZE before you set the value. Otherwise it will be cut off with first \0 in the string. So this flag toggle from keySetString() to keySetBinary(). If no value (nor size) is given, it will be a NULL key.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex2",
        KEY_BINARY,
        KEY_SIZE, 4,                // now the size is important
        KEY_VALUE, "some data",     // sets the binary value ("some")
        KEY_END);                   // end of args
Key *k=keyNew("user:/tmp/ex4",
        KEY_BINARY,                   // key type
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",                 // value that will be truncated in 7 bytes
        KEY_META, "comment/#0", "value is truncated",
        KEY_END);                       // end of args
```

**Precondition**

`name` is a valid Key name

Variable arguments are a valid combination

**Postcondition**

returns a new, fully initialized Key object with the valid Key name and all data given by variable arguments

**Parameters**

| | |
|---|---|
| *name* | a valid name to the key (see keySetName()) |

**Returns**

a pointer to a new allocated and initialized Key object.

**Return values**

| | |
|---|---|
| *NULL* | on allocation error or if an invalid `name` was passed (see keySetName()). |

**Since**

1.0.0

**See also**

keyDel() for deallocating a created Key object
keySetName() for rules about which names are considered valid

**Exceptions**

| | |
|---|---|
| *KeyInvalidName* | if key could not be constructed |

**Parameters**

| | |
|---|---|
| *keyName* | the name of the new key |

### 572.30.2.6 Key() [6/7]

```
kdb::Key::Key (
            const std::string keyName,
             ...  )  [inline], [explicit]
```

A practical way to fully create a Key object in one step.
To just get a key object, simple do:

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/some/example", KEY_END);
// work with it
keyDel (k);
```

keyNew() allocates memory for a key object and keyDel() cleans everything up.
If you want the key object to contain a name, value, comment and other meta info read on.

**Note**

When you already have a key with similar properties its easier to keyDup() the key.

You can call keyNew() in many different ways depending on the attribute tags you pass as parameters. Tags are represented as elektraKeyFlags values, and tell keyNew() which Key attribute comes next. The Key attribute tags are the following:

- KEY_VALUE
  Next parameter is a pointer to the value that will be used. If no KEY_BINARY was used before, a string is assumed.
  ```
  // Create and initialize a key with a name and nothing else
  Key *k=keyNew("user:/tmp/ex0",
          KEY_VALUE, "some data",    // set a string value
          KEY_END);                  // end of args
  ```
- KEY_SIZE
  Define a maximum length of the value. This is only used when setting a binary key.
  ```
  // Create and initialize a key with a name and nothing else
  Key *k=keyNew("user:/tmp/ex1",
          KEY_SIZE, 4,               // has no effect on strings
          KEY_VALUE, "some data",    // set a string value
          KEY_END);                  // end of args
  ```
- KEY_META
  Next two parameter is a metaname and a metavalue. See keySetMeta().
  ```
  Key *k=keyNew("user:/tmp/ex3",
          KEY_META, "comment/#0", "a comment",  // with a comment
          KEY_META, "owner", "root",          // and an owner
          KEY_META, "special", "yes",         // and any other metadata
          KEY_END);                  // end of args
  ```

- KEY_END
  Must be the last parameter passed to keyNew(). It is always required, unless the `keyName` is 0.
- KEY_FLAGS
  Bitwise disjunction of flags, which don't require one or more values. recommended way to set multiple flags. overrides previously defined flags.

```
Key *k=keyNew("user:/tmp/ex3",
        KEY_BINARY,                  // binary key
        KEY_SIZE, 7,                   // assume binary length 7
        KEY_VALUE, "some data",               // value that will be truncated in 7 bytes
        KEY_END);                     // end of args
```

- KEY_BINARY
  Allows one to change the key to a binary key. Make sure that you also pass KEY_SIZE before you set the value. Otherwise it will be cut off with first \0 in the string. So this flag toggle from keySetString() to keySetBinary(). If no value (nor size) is given, it will be a NULL key.

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex2",
        KEY_BINARY,
        KEY_SIZE, 4,               // now the size is important
        KEY_VALUE, "some data",    // sets the binary value ("some")
        KEY_END);                  // end of args
Key *k=keyNew("user:/tmp/ex4",
        KEY_BINARY,                   // key type
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",               // value that will be truncated in 7 bytes
        KEY_META, "comment/#0", "value is truncated",
        KEY_END);                       // end of args
```

#### Precondition

`name` is a valid Key name

Variable arguments are a valid combination

#### Postcondition

returns a new, fully initialized Key object with the valid Key name and all data given by variable arguments

### Parameters

| | |
|---|---|
| *name* | a valid name to the key (see keySetName()) |

#### Returns

a pointer to a new allocated and initialized Key object.

### Return values

| | |
|---|---|
| *NULL* | on allocation error or if an invalid `name` was passed (see keySetName()). |

#### Since

1.0.0

#### See also

keyDel() for deallocating a created Key object

keySetName() for rules about which names are considered valid

### Exceptions

| | |
|---|---|
| *KeyInvalidName* | if key could not be constructed |

**Warning**

Not supported on some compilers, e.g. clang which requires you to only pass non-POD in varg lists.

**Parameters**

| | |
|---|---|
| *keyName* | the name of the new key |

### 572.30.2.7 Key() [7/7]

```
kdb::Key::Key (
            const char * keyName,
            va_list ap )  [inline], [explicit]
```

A practical way to fully create a Key object in one step.
To just get a key object, simple do:

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/some/example", KEY_END);
// work with it
keyDel (k);
```

keyNew() allocates memory for a key object and keyDel() cleans everything up.
If you want the key object to contain a name, value, comment and other meta info read on.

**Note**

When you already have a key with similar properties its easier to keyDup() the key.

You can call keyNew() in many different ways depending on the attribute tags you pass as parameters. Tags are represented as elektraKeyFlags values, and tell keyNew() which Key attribute comes next. The Key attribute tags are the following:

- KEY_VALUE
  Next parameter is a pointer to the value that will be used. If no KEY_BINARY was used before, a string is assumed.
  ```
  // Create and initialize a key with a name and nothing else
  Key *k=keyNew("user:/tmp/ex0",
          KEY_VALUE, "some data",     // set a string value
          KEY_END);                  // end of args
  ```
- KEY_SIZE
  Define a maximum length of the value. This is only used when setting a binary key.
  ```
  // Create and initialize a key with a name and nothing else
  Key *k=keyNew("user:/tmp/ex1",
          KEY_SIZE, 4,               // has no effect on strings
          KEY_VALUE, "some data",    // set a string value
          KEY_END);                  // end of args
  ```
- KEY_META
  Next two parameter is a metaname and a metavalue. See keySetMeta().
  ```
  Key *k=keyNew("user:/tmp/ex3",
          KEY_META, "comment/#0", "a comment",  // with a comment
          KEY_META, "owner", "root",         // and an owner
          KEY_META, "special", "yes",        // and any other metadata
          KEY_END);                  // end of args
  ```
- KEY_END
  Must be the last parameter passed to keyNew(). It is always required, unless the `keyName` is 0.
- KEY_FLAGS
  Bitwise disjunction of flags, which don't require one or more values. recommended way to set multiple flags. overrides previously defined flags.
  ```
  Key *k=keyNew("user:/tmp/ex3",
          KEY_BINARY,                  // binary key
          KEY_SIZE, 7,                   // assume binary length 7
          KEY_VALUE, "some data",              // value that will be truncated in 7 bytes
          KEY_END);                    // end of args
  ```

- KEY_BINARY
  Allows one to change the key to a binary key. Make sure that you also pass KEY_SIZE before you set the value. Otherwise it will be cut off with first \0 in the string. So this flag toggle from keySetString() to keySetBinary(). If no value (nor size) is given, it will be a NULL key.

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex2",
        KEY_BINARY,
        KEY_SIZE, 4,               // now the size is important
        KEY_VALUE, "some data",    // sets the binary value ("some")
        KEY_END);                  // end of args
Key *k=keyNew("user:/tmp/ex4",
        KEY_BINARY,                   // key type
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",               // value that will be truncated in 7 bytes
        KEY_META, "comment/#0", "value is truncated",
        KEY_END);                       // end of args
```

**Precondition**

`name` is a valid Key name

Variable arguments are a valid combination

**Postcondition**

returns a new, fully initialized Key object with the valid Key name and all data given by variable arguments

**Parameters**

| | |
|---|---|
| *name* | a valid name to the key (see keySetName()) |

**Returns**

a pointer to a new allocated and initialized Key object.

**Return values**

| | |
|---|---|
| *NULL* | on allocation error or if an invalid `name` was passed (see keySetName()). |

**Since**

1.0.0

**See also**

keyDel() for deallocating a created Key object

keySetName() for rules about which names are considered valid

**Exceptions**

| | |
|---|---|
| *KeyInvalidName* | if key could not be constructed |

**Parameters**

| | |
|---|---|
| *keyName* | the name of the new key |
| *ap* | the variable argument list pointer |

#### 572.30.2.8 ~Key()

```
kdb::Key::~Key ( )  [inline]
```

Destructs the key.

**See also**

del()

### 572.30.3 Member Function Documentation

#### 572.30.3.1 addBaseName()

```
void kdb::Key::addBaseName (
            const std::string & baseName )  [inline]
```

Adds a base name for a key.

Adds `baseName` to the name of `key`. `baseName` will be escaped before adding it to the name of `key`. No other part of the Key's name will be affected.

Assumes that `key` is a directory and will append `baseName` to it. The function adds the path separator for concatenating.

If `key` has the name `"system:/dir1/dir2"` and this method is called with `baseName` `"mykey"`, the resulting key will have the name `"system:/dir1/dir2/mykey"`.

When `baseName` is 0, nothing will happen and the size of the name is returned.

The escaping rules apply as in above .

A simple example is:

```
Key * k = keyNew ("user:/my/long", KEY_END);
keyAddBaseName (k, "myname");
printf ("%s\n", keyName (k)); // will print user:/my/long/myname
keyDel (k);
```

E.g. if you add . it will be escaped:

```
        keySetName (k, "system:/valid");
        succeed_if (keyAddBaseName (k, ".") >= 0, "could not add a base name");
        succeed_if_same_string (keyName (k), "system:/valid/\\.");
        succeed_if_same_string (keyBaseName (k), ".");
```

**Parameters**

| | |
|---|---|
| *key* | the Key to add the basename to |
| *baseName* | the string to append to the Key's name |

**Returns**

the size in bytes of the Key's new name including the NULL terminator

**Return values**

| | |
|---|---|
| *-1* | if the Key has no name |
| *-1* | on NULL pointers |
| *-1* | if Key was inserted into KeySet before |
| *-1* | if the Key was read-only |
| *-1* | on memory allocation errors |

**Since**

1.0.0

**See also**

keySetBaseName() for setting the basename of a Key

keySetName() for setting the name of a Key

**Exceptions**

| | |
|---|---|
| *KeyInvalidName* | if the name is not valid |

### 572.30.3.2 clear()

```
void kdb::Key::clear ( )  [inline]
```

Clears/Invalidates a key.
Afterwards the object is empty again.

**Note**

This is not a null key, so it will evaluate to true. isValid() will, however, be false.

**See also**

release()

isValid(), isNull()

Will clear all internal data of a Key. After this call you will receive a fresh Key - with no value, metadata or name.
The reference counter will stay unmodified.

**Note**

that you might also clear() all aliases with this operation.

```
int f (Key *k)
{
        keyClear (k);
        // you have a fresh Key k here
        keySetString (k, "value");
        // the caller will get an empty Key k with an value
}
```

**Postcondition**

`key's` name is "/"

`key's` metadata is empty

**Parameters**

| | |
|---|---|
| *key* | the Key that should be cleared |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

keyDel() for completely deleting a Key

### 572.30.3.3 copy()

```
void kdb::Key::copy (
            const Key & other,
            elektraCopyFlags flags = KEY_CP_ALL )  [inline]
```

Copy or clear a key.
Depending on the chosen `flags` keyCopy() only copies certain parts of `source` into `dest`.

- If KEY_CP_NAME is set, the key name will be copied from `source` to `dest`.
- If KEY_CP_META is set, the meta keys will be copied from `source` to `dest`.
- If KEY_CP_VALUE is set, the key value will be copied from `source` to `dest`. Additionally, if `source` is a binary key (keyIsBinary()), `dest` will also be marked as binary. This means that even if KEY_CP_META is not set, the `binary` meta key will be copied with KEY_CP_VALUE.
- If KEY_CP_STRING is set, the key value will be copied from `source` to `dest`, but only, if `source` is *not* a binary key (keyIsBinary()). If `source` is binary, keyCopy() fails. If `dest` is binary, it will still be marked as binary after the copy. This cannot be used together with KEY_CP_VALUE. The main purpose of KEY_CP_STRING is for copying *into* known string keys. It ensure that you don't accidentally convert string keys into binary keys.

There is also the shorthand KEY_CP_ALL. It is equivalent to `KEY_CP_NAME | KEY_CP_VALUE | KEY_CP_META`, i.e. all key data supported by keyCopy() will be copied from `source` to `dest`.
Use this function when you need to copy into an existing key, e.g. because it was passed by a pointer in a function you can do so:

```
keyCopy (copy, orig, KEY_CP_ALL);
```

Most often you will want to duplicate an existing key. For this purpose the alias keyDup() exists. Calling

```
copy = keyDup (orig, KEY_CP_ALL);
```

is equivalent to

```
copy = keyCopy (keyNew ("/", KEY_END), orig, KEY_CP_ALL);
```

The reference counter will not be changed for both keys. Affiliation to keysets are also not affected.
Since metadata uses copy-on-write semantics there is only a constant memory cost to copying metadata.
When you pass a NULL-pointer as `source` the pieces of `dest` specified by `flags` will be cleared.
Calling `keyCopy (dest, NULL, KEY_CP_ALL)` is different from calling keyClear(). The key will not be fully reset, the reference counter and internal flags will remain unchanged. Additionally, keyCopy() respects keyLock() state, while keyClear() always works.

```
keyCopy (k, NULL, KEY_CP_ALL);
// name, value and metadata of k have now been clear
// lock flags, reference count, etc. remain unchanged
```

**Precondition**

`dest` must be a valid Key (created with keyNew)

`dest` must not have read-only flags set

`source` must be a valid Key or NULL

**Invariant**

Key name stays valid until delete

**Postcondition**

Value from Key source is written to Key dest

**Parameters**

| | |
|---|---|
| *dest* | the key which will be written to |
| *source* | the key which should be copied or NULL to clear the data of `dest` |
| *flags* | specifies which parts of the key should be copied |

**Returns**

`dest`

**Return values**

| | |
|---|---|
| *NULL* | on memory allocation problems |
| *NULL* | when a part of `dest` that should be modified (e.g. name, value) was marked read-only, e.g. the name of `dest` will be read-only if `dest` is part of a KeySet |
| *NULL* | when `dest` is NULL |
| *NULL* | when both KEY_CP_VALUE and KEY_CP_STRING are set in `flags` |
| *NULL* | when both KEY_CP_STRING is set in `flags` and `source` is a binary key (keyIsBinary()) |

**Since**

0.9.5

**See also**

keyDup() for duplicating an existing Key

### 572.30.3.4 copyAllMeta()

```
void kdb::Key::copyAllMeta (
            const Key & other )  [inline]
```

Do a shallow copy of all metadata from source to dest.

The key dest will additionally have all metadata the source had. Metadata not present in source will not be changed. Metadata which was present in source and dest will be overwritten. If the `dest` Key is read-only it will not be changed.

For example the metadata type is copied into the Key k:

```
void l (Key * k)
{
        // receive copy
        keyCopyAllMeta (k, copy);
        // the caller will see the changed key k
        // with all the metadata from copy
}
```

The main purpose of this function is for plugins or applications which want to add the same metadata to n keys. When you do that with keySetMeta() it will take n times the memory for the key. This can be considerable amount of memory for many keys with some metadata for each.

To avoid that problem you can use keyCopyAllMeta() or keyCopyMeta():

```
void o (KeySet * ks)
{
        Key * current;
        Key * shared = keyNew ("/", KEY_END);
        keySetMeta (shared, "shared1", "this metadata should be shared among many keys");
        keySetMeta (shared, "shared2", "this metadata should be shared among many keys also");
        keySetMeta (shared, "shared3", "this metadata should be shared among many keys too");
        for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
        {
                current = ksAtCursor (ks, it);
                if (needsSharedData (current)) keyCopyAllMeta (current, shared);
        }
        keyDel (shared);
}
```

#### Precondition

`dest's` metadata is not read-only

#### Postcondition

for every metaName present in source: keyGetMeta(source, metaName) == keyGetMeta(dest, metaName)

#### Parameters

| | |
|---|---|
| *dest* | the destination where the metadata should be copied too |
| *source* | the key where the metadata should be copied from |

#### Return values

| | |
|---|---|
| *1* | if metadata was successfully copied |
| *0* | if source did not have any metadata |
| *-1* | on null pointer of dest or source |
| *-1* | on memory problems |

#### Since

1.0.0

#### See also

keyCopyMeta() for copying one metadata Key from `dest` to `source`

getMeta(), setMeta(), copyMeta()

### 572.30.3.5 copyMeta()

```
void kdb::Key::copyMeta (
            const Key & other,
            const std::string & metaName )  [inline]
```

Do a shallow copy of metadata with name `metaName` from source to dest.

Afterwards `source` and `dest` will have the same metadata referred with `metaName`. If the Key with name `metaName` doesn't exist in `source` - it gets deleted in `dest`.

For example the metadata type is copied into the Key k.

```
void l(Key *k)
{
        // receive c
        keyCopyMeta(k, c, "type");
        // the caller will see the changed key k
        // with the metadata "type" from c
}
```

The main purpose of this function is for plugins or applications, which want to add the same metadata to n keys. When you do that keySetMeta() will take n times the memory for the key. This can be a considerable amount of memory for many keys with some metadata for each.

To avoid that problem you can use keyCopyAllMeta() or keyCopyMeta().

```
void o(KeySet *ks)
{
        Key *shared = keyNew ("/", KEY_END);
        keySetMeta(shared, "shared", "this metadata should be shared among many keys");
        for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
        {
                Key * current = ksAtCursor (ks, it);
                if (needs_shared_data(current)) keyCopyMeta(current, shared, "shared");
        }
}
```

**Precondition**

`dest's` metadata is not read-only

**Postcondition**

keyGetMeta(source, metaName) == keyGetMeta(dest, metaName)

**Parameters**

| | |
|---|---|
| *dest* | the destination where the metadata should be copied to |
| *source* | the key where the metadata should be copied from |
| *metaName* | the name of the metadata Key which should be copied |

**Return values**

| | |
|---|---|
| *1* | if was successfully copied |
| *0* | if the metadata in dest was removed too |
| *-1* | on null pointers (source or dest) |
| *-1* | on memory problems |
| *-1* | if metadata is read-only |

**Since**

1.0.0

**See also**

keyCopyAllMeta() copies all metadata from `dest` to `src`

getMeta(), setMeta(), copyAllMeta()

### 572.30.3.6 delBaseName()

```
void kdb::Key::delBaseName ( )  [inline]
```

Delete the baseName of a key.

**Exceptions**

| | |
|---|---|
| *KeyInvalidName* | if the name is not valid |

### 572.30.3.7 delMeta()

```
void kdb::Key::delMeta (
            const std::string & metaName )  [inline]
```

Delete metadata for key.

**See also**

setMeta(), getMeta(), copyMeta(), copyAllMeta()

#### 572.30.3.8 dup()

```
Key kdb::Key::dup (
            elektraCopyFlags flags = KEY_CP_ALL ) const  [inline]
```

#### 572.30.3.9 get()

```
template<class T >
T kdb::Key::get  [inline]
```

Get a key value.

You can write your own template specialication, e.g.:

```
template <>
inline QColor Key::get() const
{
        if (getStringSize() < 1)
        {
                throw KeyTypeConversion();
        }
        std::string str = getString();
        QColor c(str.c_str());
        return c;
}
```

**Returns**

the string directly from the key.

It should be the same as get().

**Returns**

empty string on null pointers

**Exceptions**

| | |
|---|---|
| *KeyException* | on null key or not a valid size |
| *KeyTypeMismatch* | if key holds binary data and not a string |

**Note**

unlike in the C version, it is safe to change the returned string.

**See also**

isString(), getBinary()

This method tries to serialise the string to the given type.

#### 572.30.3.10 getBaseName()

```
std::string kdb::Key::getBaseName ( ) const  [inline]
```

Returns a pointer to the unescaped Key's name where the basename starts.

This is a much more efficient version of keyGetBaseName() and you should use it if you are responsible enough to not mess up things. The name might change or even point to a wrong place after a keySetName(). So make sure to copy the memory before the name changes.

keyBaseName() returns "" when the Key has no basename. The reason is

```
        keySetName (k, "");
        succeed_if_same_string (keyBaseName (k), "");
        keySetName (k, "user:/");
        succeed_if_same_string (keyBaseName (k), "");
```

There is also support for really empty basenames:

```
        keySetName (k, "system:/valid");
        succeed_if (keyAddBaseName (k, "") >= 0, "could not add a base name");
        succeed_if_same_string (keyName (k), "system:/valid/%");
        succeed_if_same_string (keyBaseName (k), "");
```

**Note**

You must never use the pointer returned by keyBaseName() method to change the name. You should use keySetBaseName() instead.

Do not assume that keyBaseName() points to the same region as keyName() does.

**Parameters**

| | |
|---|---|
| *key* | the Key to obtain the basename from |

**Returns**

a pointer to the Key's basename

**Return values**

| | |
|---|---|
| *""* | when the Key has no (base)name |
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

keyGetBaseName() for getting a copy of the Key's basename

keyGetBaseNameSize() for getting the size of the Key's basename

keyName() for getting a pointer to the Key's name

### 572.30.3.11 getBaseNameSize()

```
ssize_t kdb::Key::getBaseNameSize ( ) const  [inline]
```

Calculates number of bytes needed to store basename of `key` (including NULL terminator).
Key names consisting of only root names (e.g. "`system:/`" or "`user:/`" or "`user:domain`" ) do not have basenames. In this case the function will return 1, because only a NULL terminator is needed for storage.
Basenames are denoted as:

- `system:/some/thing/basename` -> `basename`
- `user:domain/some/thing/base\/name` > `base\/name`

**Parameters**

| | |
|---|---|
| *key* | the Key to get the size of the basename from |

**Returns**

size in bytes of the Key's basename including NULL terminator

**Return values**

| | |
|---|---|
| *-1* | if the Key or the Key's basename is NULL |

**Since**

1.0.0

**See also**

keyBaseName() for getting a pointer to a Key's basename

keyGetBaseName() for getting a copy of a Key's basename

keyName(), keyGetName() for getting a pointer / copy of the whole name

keySetName() for setting a Key's name

### 572.30.3.12 getBinary()

`std::string kdb::Key::getBinary ( ) const [inline]`

Copy the binary value of a Key into `returnedBinary`.

**Returns**

the binary Value of the key.

**Return values**

| | |
|---|---|
| "" | on null pointers (size == 0) and on data only containing \0 |

**Note**

if you need to distinguish between null pointers and data containing \0 you can use getValue().

**Exceptions**

| | |
|---|---|
| *KeyException* | on invalid binary size |
| *KeyTypeMismatch* | if key is string and not a binary |

If the type is not binary -1 will be returned.
When the binary data is empty (this is not the same as ""!) 0 will be returned and `returnedBinary` will not be changed.
For string values see keyGetString() and keyIsString().
When `returnedBinary` is too small to hold the data (maximum size is given by maxSize), the returnedBinary will not be changed and -1 is returned.

**Example:**

```
Key *key = keyNew ("user:/keyname", KEY_BINARY, KEY_END);
char buffer[300];
if (keyGetBinary(key,buffer,sizeof(buffer)) == -1)
{
        // handle error
}
```

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the binary value from |
| *returnedBinary* | pre-allocated memory to store a copy of the Key's value |
| *maxSize* | number of bytes of pre-allocated memory in `returnedBinary` |

**Returns**

the number of bytes copied to `returnedBinary`

**Return values**

| | |
|---|---|
| *0* | if the binary is empty |
| *-1* | on NULL pointers |
| *-1* | if maxSize is 0, too small for the value or larger than SSIZE_MAX |
| *-1* | if the Key's value is a string |

**Since**

1.0.0

**See also**

keyValue() for getting a raw pointer to the Key's value

keyGetValueSize() for getting the size of the Key's value

keySetBinary() for setting the binary value of a Key

keyIsBinary() for checking whether a Key's value is binary

keyGetString(), keySetString() for working with string values

isBinary(), getString(), getValue()

### 572.30.3.13 getBinarySize()

```
ssize_t kdb::Key::getBinarySize ( ) const  [inline]
```

Returns the number of bytes needed to store the key value, including the NULL terminator.
It returns the correct size, independent of the Key Type. If the value is binary there might be '\0' values in it.
For an empty string you need one byte to store the ending NULL. For that reason 1 is returned. This is not true for binary data, so 0 will be returned.
A binary key has no '\0' termination. String types are null-terminated, and the terminator will be considered for the length.
This method can be used with elektraMalloc() before keyGetString() or keyGetBinary() is called.

```
char *buffer;
buffer = elektraMalloc (keyGetValueSize (key));
// use this buffer to store the value (binary or string)
// pass keyGetValueSize (key) for maxSize
```

**Postcondition**

returns the exact amount of bytes needed to store `key's` value (including NULL terminators)

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the size of the value from |

**Returns**

the number of bytes needed to store the Key's value

**Return values**

| | |
|---|---|
| *1* | when there is no data and type is a string |
| *0* | when there is no data and type is binary |
| *-1* | on null pointer |

**Since**

1.0.0

**See also**

keyGetString() for getting the Key's value as a string

keyGetBinary() for getting the Key's value as a binary

keyValue() for getting a pointer to the Key's value

### 572.30.3.14 getFunc()

```
Key::func_t kdb::Key::getFunc ( ) const  [inline]
```

Elektra can store function pointers as binary.
This function returns such a function pointer.

**Exceptions**

| | |
|---|---|
| *KeyTypeMismatch* | if no binary data found, or binary data has not correct length |

**Returns**

a function pointer stored with setBinary()

### 572.30.3.15 getKey()

```
ckdb::Key * kdb::Key::getKey ( ) const  [inline]
```

Passes out the raw key pointer.
This pointer can be used to directly change the underlying key object.

**Note**

that the ownership remains in the object

### 572.30.3.16 getMeta()

```
template<class T >
T kdb::Key::getMeta (
            const std::string & metaName ) const  [inline]
```

Returns the Key for a metadata entry with name `metaName`.
You are not allowed to modify the resulting key.
If `metaName` does not start with 'meta:/', it will be prefixed with 'meta:/'.

```
Key metaData = keyGetMeta(k, "type")
// keyType == "boolean"
char keyType[] = keyValue(metaData)
```

**Note**

You must not delete or change the returned key, use keySetMeta() if you want to delete or change it.

**Precondition**

`key` contains metadata

`metaName` is prefixed with "meta:/"

#### Parameters

| | |
|---|---|
| *key* | the Key from which to get metadata |
| *metaName* | the name of the meta information you want the Key from. |

#### Returns

value of meta-information if meta-information is found

#### Return values

| | |
|---|---|
| *0* | if key or metaName is NULL |
| *0* | if no such metaName is found |

#### Since

1.0.0

#### See also

keySetMeta() for setting metadata

keyMeta() for getting the KeySet containing metadata

You can specify your own template specialisation:

```
template<>
inline yourtype Key::getMeta(const std::string &name) const
{
        yourtype x;
        std::string str;
        str = std::string(
                static_cast<const char*>(
                        ckdb::keyValue(
                                ckdb::keyGetMeta(key, name.c_str())
                                )
                        )
                );
        return yourconversion(str);
}
```

#### Exceptions

| | |
|---|---|
| *KeyTypeConversion* | if metadata could not be parsed |

#### Note

No exception will be thrown if a const Key or char* is requested, but don't forget the const: getMeta<const Key>, otherwise you will get an compiler error.

If no meta is available:

- char* is null (evaluates to 0)
- const Key is null (evaluate to false)
- otherwise the default constructed type will be returned

#### See also

hasMeta

delMeta(), setMeta(), copyMeta(), copyAllMeta()

### 572.30.3.17 getName()

`std::string kdb::Key::getName ( ) const [inline]`

Returns a pointer to the abbreviated real internal `key` name.

This is a much more efficient version of keyGetName() and can use it if you are responsible enough to not mess up things. You are not allowed to change anything in the returned array. The content of that string may change after keySetName() and similar functions. If you need a copy of the name, consider using keyGetName().

#### Return values

| | |
|---|---|
| *""* | when there is no keyName. The reason is |

```
key=keyNew(0);
keySetName(key,"");
keyName(key); // you would expect "" here
keyDel(key);
```

Valid key names are:

- `spec:/something` for specification of other keys.
- `proc:/something` for in-memory keys, e.g. commandline.
- `dir:/something` for dir keys in current working directory
- `system:/something` for system keys in /etc or /
- `user:/something` for user keys in home directory
- `user:username/something` for other users (deprecated: kdbGet() + kdbSet() currently unsupported)
- `/something` for cascading keys (actually refers to one of the above, see also ksLookup())

#### Note

Note that the Key structure keeps its own size field that is calculated by library internal calls, so to avoid inconsistencies, you must never use the pointer returned by keyName() method to set a new value. Use keySetName() instead.

#### Parameters

| | |
|---|---|
| *key* | the Key you want to get the name from |

#### Returns

a pointer to the Key's name which must not be changed.

#### Return values

| | |
|---|---|
| *""* | when Key's name is empty |
| *0* | on NULL pointer |

#### Since

1.0.0

#### See also

keyGetNameSize() for the string length

keyGetName() as alternative to get a copy

keyUnescapedName to get an unescaped Key name

**Exceptions**

| | |
|---|---|
| *KeyException* | if key is null |

**Note**

unlike in the C version, it is safe to change the returned string.

### 572.30.3.18 getNameSize()

```
ssize_t kdb::Key::getNameSize ( ) const  [inline]
```

Bytes needed to store the Key's name (excluding owner).
For an empty Key name you need one byte to store the ending NULL. For that reason, 1 is returned when the name is empty.

**Parameters**

| | |
|---|---|
| *key* | the Key to get the name size from |

**Returns**

number of bytes needed, including NULL terminator, to store Key's name (excluding owner)

**Return values**

| | |
|---|---|
| *1* | if Key has no name |
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

keyGetName() for getting the Key's name

keyGetUnescapedNameSize() for getting the size of the unescaped name

### 572.30.3.19 getNamespace()

```
ElektraNamespace kdb::Key::getNamespace ( ) const  [inline]
```

**Returns**

namespace of the key

**See also**

ElektraNamespace, keyGetNamespace

### 572.30.3.20 getReferenceCounter()

```
uint16_t kdb::Key::getReferenceCounter ( ) const  [inline]
```

Return the current reference counter value of a Key object.

**Parameters**

| | |
|---|---|
| *key* | the Key whose reference counter to retrieve |

**Returns**

the value of the `key's` reference counter

**Return values**

| | |
|---|---|
| *-1* | on NULL pointer |

**Since**

1.0.0

**See also**

keyIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

keyDecRef() for decreasing the reference counter

### 572.30.3.21 getString()

```
std::string kdb::Key::getString ( ) const  [inline]
```

**Returns**

the string directly from the key.

It should be the same as get().

**Returns**

empty string on null pointers

**Exceptions**

| | |
|---|---|
| *KeyException* | on null key or not a valid size |
| *KeyTypeMismatch* | if key holds binary data and not a string |

**Note**

unlike in the C version, it is safe to change the returned string.

**See also**

isString(), getBinary()

### 572.30.3.22 getStringSize()

```
ssize_t kdb::Key::getStringSize ( ) const  [inline]
```

Returns the number of bytes needed to store the key value, including the NULL terminator.
It returns the correct size, independent of the Key Type. If the value is binary there might be '\0' values in it.

For an empty string you need one byte to store the ending NULL. For that reason 1 is returned. This is not true for binary data, so 0 will be returned.
A binary key has no '\0' termination. String types are null-terminated, and the terminator will be considered for the length.
This method can be used with elektraMalloc() before keyGetString() or keyGetBinary() is called.

```
char *buffer;
buffer = elektraMalloc (keyGetValueSize (key));
// use this buffer to store the value (binary or string)
// pass keyGetValueSize (key) for maxSize
```

**Postcondition**

returns the exact amount of bytes needed to store `key's` value (including NULL terminators)

**Parameters**

| | |
|---|---|
| *key* | the Key object to get the size of the value from |

**Returns**

the number of bytes needed to store the Key's value

**Return values**

| | |
|---|---|
| *1* | when there is no data and type is a string |
| *0* | when there is no data and type is binary |
| *-1* | on null pointer |

**Since**

1.0.0

**See also**

keyGetString() for getting the Key's value as a string

keyGetBinary() for getting the Key's value as a binary

keyValue() for getting a pointer to the Key's value

#### 572.30.3.23 getValue()

```
const void * kdb::Key::getValue ( ) const  [inline]
```

Return a pointer to the real internal `key` value.
This is a much more efficient version of keyGetString() keyGetBinary(). You should use it if you are responsible enough to not mess up things. You are not allowed to modify anything in the returned string. If you need a copy of the Value, consider to use keyGetString() or keyGetBinary() instead.

### 572.30.4 String Handling

If `key` is string (keyIsString()), you may cast the returned as a `"char *"` because you'll get a NULL terminated regular string.
keyValue() returns "" in string mode when there is no value. The reason is

```
key=keyNew(0);
keySetString(key,"");
keyValue(key); // you would expect "" here
keyDel(key);
```

### 572.30.5 Binary Data Handling

If the data is binary, the size of the value must be determined by keyGetValueSize(), any strlen() operations are not suitable to determine the size.
keyValue() returns 0 in binary mode when there is no value. The reason is

```
key=keyNew(0);
keySetBinary(key, 0, 0);
keyValue(key); // you would expect 0 here
keySetBinary(key,"", 1);
keyValue(key); // you would expect "" (a pointer to '\0') here
int i=23;
keySetBinary(key, (void*)&i, 4);
(int*)keyValue(key); // you would expect a pointer to (int)23 here
keyDel(key);
```

#### Note

Note that the Key structure keeps its own size field that is calculated by library internal calls, so to avoid inconsistencies, you must never use the pointer returned by keyValue() method to set a new value. Use keySetString() or keySetBinary() instead.

#### Warning

Binary keys will return a NULL pointer when there is no data in contrast to keyName(), keyBaseName() and keyComment(). For string value the behaviour is the same.

#### Example:

```
KDB *handle = kdbOpen();
KeySet *ks=ksNew(0, KS_END);
kdbGetByName(handle,ks,"system:/sw/my",KDB_O_SORT|KDB_O_RECURSIVE);
 for (elektraCursor it = 0; it < ksGetSize (ks); ++it)
 {
        Key * current = ksAtCursor (ks, it);
        size_t size=0;
        if (keyIsBinary(current)) {
                size=keyGetValueSize(current);
                printf("Key %s has a value of size %d bytes. Value: <BINARY>\nComment: %s",
                        keyName(current),
                        size,
                        keyComment(current));
        } else {
                size=elektraStrLen((char *)keyValue(current));
                printf("Key %s has a value of size %d bytes. Value: %s\nComment: %s",
                        keyName(current),
                        size,
                        (char *)keyValue(current),
                        keyComment(current));
        }
}
ksDel (ks);
kdbClose (handle);
```

#### Precondition

`key` is not NULL and has stored data

#### Postcondition

returned pointer points to the stored internal value

if the value is a string, the value is NULL terminated

#### Parameters

| | |
|---|---|
| *key* | the Key from which to get the value |

**Returns**

a pointer to the Key's internal value

**Return values**

| | |
|---|---|
| *""* | when there is no value and Key is not binary |
| *0* | where there is no value and Key is binary |
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

keyGetValueSize() to get the size of the Key's value

keyGetString() for getting the Key's value as string

keyGetBinary() for getting the Key's value as binary

**Returns**

the value of the key

**See also**

getBinary()

### 572.30.5.1 hasMeta()

```
bool kdb::Key::hasMeta (
            const std::string & metaName ) const  [inline]
```

**Return values**

| | |
|---|---|
| *true* | if there is a metadata with given name |
| *false* | if no such metadata exists |

**See also**

getMeta()

### 572.30.5.2 isBelow() [1/2]

```
bool kdb::Key::isBelow (
            const Key & k ) const  [inline]
```

Check if the Key `check` is below the Key `key` or not.

**Parameters**

| | |
|---|---|
| *k* | the other key |

#### Returns

true if our key is below k

Example:

```
key user:/sw/app
check user:/sw/app/key
```

returns true because `check` is below `key`
Example:

```
key user:/sw/app
check user:/sw/app/folder/key
```

returns also true because `check` is indirectly below `key`
Obviously, there is no Key above a namespace (e.g. user, system, /):

```
key *
check user
```

#### Parameters

| | |
|---|---|
| *key* | the Key object to check against |
| *check* | the Key object for which it should be checked whether it is below `key` |

#### Return values

| | |
|---|---|
| *1* | if `check` is below `key` |
| *0* | if it is not below or if it is the same key |
| *-1* | if key or check is null |

#### Since

1.0.0

#### See also

keyIsDirectlyBelow() for checking whether a Key is directly below another

keyGetName(), keySetName() for getting / setting the Key's name

### 572.30.5.3 isBelow() [2/2]

```
bool kdb::Key::isBelow (
            std::string const & name ) const  [inline]
```

Check if the Key `check` is below the Key `key` or not.

#### Parameters

| | |
|---|---|
| *name* | the name of the other key |

#### Returns

true if our key is below k

Example:

```
key user:/sw/app
check user:/sw/app/key
```

returns true because `check` is below `key`
Example:

```
key user:/sw/app
check user:/sw/app/folder/key
```

returns also true because `check` is indirectly below `key`
Obviously, there is no Key above a namespace (e.g. user, system, /):

```
key *
check user
```

**Parameters**

| | |
|---|---|
| *key* | the Key object to check against |
| *check* | the Key object for which it should be checked whether it is below `key` |

**Return values**

| | |
|---|---|
| *1* | if `check` is below `key` |
| *0* | if it is not below or if it is the same key |
| *-1* | if key or check is null |

**Since**

1.0.0

**See also**

keyIsDirectlyBelow() for checking whether a Key is directly below another

keyGetName(), keySetName() for getting / setting the Key's name

### 572.30.5.4 isBelowOrSame() [1/2]

```
bool kdb::Key::isBelowOrSame (
            const Key & k ) const  [inline]
```

Check if a key is below or same.

**Parameters**

| | |
|---|---|
| *k* | the other key |

**Returns**

true if our key is below k or the same as k

**Parameters**

| | |
|---|---|
| *key* | the key object to work with |

**See also**

keyIsBelow()

#### 572.30.5.5 isBelowOrSame() [2/2]

```
bool kdb::Key::isBelowOrSame (
            std::string const & name ) const  [inline]
```

Check if a key is below or same.

**Parameters**

| | |
|---|---|
| *name* | the name of the other key |

**Returns**

true if our key is below k or the same as k

**Parameters**

| | |
|---|---|
| *key* | the key object to work with |

**See also**

keyIsBelow()

#### 572.30.5.6 isBinary()

```
bool kdb::Key::isBinary ( ) const  [inline]
```

Check if the value of a `key` is of binary type.
The function checks if the value of `key` is binary. Contrary to string values binary values can have '\0' inside the value and may not be terminated by a null character. Their disadvantage is that you need to pass their size.
Make sure to use this function and don't test the binary type another way to ensure compatibility and to write less error prone programs.

**Parameters**

| | |
|---|---|
| *key* | the Key to check |

**Return values**

| | |
|---|---|
| *1* | if the value of `key` is binary |
| *0* | if the value of `key` is not binary |
| *-1* | on NULL pointer |

**See also**

keyGetBinary(), keySetBinary() for getting / setting a Key's value as binary

#### 572.30.5.7 isCascading()

```
bool kdb::Key::isCascading ( ) const  [inline]
```

Determines if the key is in cascading namespace.

**Return values**

| | |
|---|---|
| *true* | if it is a cascading key |

**Return values**

| | |
|---|---|
| *false* | otherwise |

### 572.30.5.8 isDir()

```
bool kdb::Key::isDir ( ) const  [inline]
```

Determines if the key is in dir namespace.

**Return values**

| | |
|---|---|
| *true* | if it is a dir key |
| *false* | otherwise |

### 572.30.5.9 isDirectBelow() [1/2]

```
bool kdb::Key::isDirectBelow (
            const Key & k ) const  [inline]
```

Check whether the Key `check` is directly below the Key `key`.

**Parameters**

| | |
|---|---|
| *k* | the other key |

**Returns**

true if our key is direct below k

```
Example:
key user:/sw/app
check user:/sw/app/key
```

returns true because check is directly below key

```
Example:
key user:/sw/app
check user:/sw/app/folder/key
```

does not return true, because it is only indirectly below

**Parameters**

| | |
|---|---|
| *key* | the Key object to check against |
| *check* | the Key object for which it should be checked whether it is directly below `key` |

**Return values**

| | |
|---|---|
| *1* | if `check` is directly below `key` |
| *0* | if `check` is not directly below `key` or if it is the same |
| *-1* | on null pointer |

**Since**

1.0.0

**See also**

keyIsBelow() for checking whether a Key is below another

keyGetName(), keySetName() for getting / setting the Key's name

### 572.30.5.10 isDirectBelow() [2/2]

```
bool kdb::Key::isDirectBelow (
            std::string const & name ) const  [inline]
```

Check whether the Key `check` is directly below the Key `key`.

**Parameters**

| | |
|---|---|
| *name* | the name of the other key |

**Returns**

true if our key is direct below k

```
Example:
key user:/sw/app
check user:/sw/app/key
```

returns true because check is directly below key

```
Example:
key user:/sw/app
check user:/sw/app/folder/key
```

does not return true, because it is only indirectly below

**Parameters**

| | |
|---|---|
| *key* | the Key object to check against |
| *check* | the Key object for which it should be checked whether it is directly below `key` |

**Return values**

| | |
|---|---|
| *1* | if `check` is directly below `key` |
| *0* | if `check` is not directly below `key` or if it is the same |
| *-1* | on null pointer |

**Since**

1.0.0

**See also**

keyIsBelow() for checking whether a Key is below another

keyGetName(), keySetName() for getting / setting the Key's name

#### 572.30.5.11 isMetaLocked()

```
bool kdb::Key::isMetaLocked ( ) const  [inline]
```

**Returns**

true if the metadata of our key has been locked

#### 572.30.5.12 isNameLocked()

```
bool kdb::Key::isNameLocked ( ) const  [inline]
```

**Returns**

true if the name of our key has been locked

#### 572.30.5.13 isNull()

```
bool kdb::Key::isNull ( ) const  [inline]
```

Checks if C++ wrapper has an underlying key.

**See also**

operator bool(), isValid()

**Returns**

true if no underlying key exists

#### 572.30.5.14 isProc()

```
bool kdb::Key::isProc ( ) const  [inline]
```

Determines if the key is in proc namespace.

**Return values**

| | |
|---|---|
| *true* | if it is a proc key |
| *false* | otherwise |

#### 572.30.5.15 isSpec()

```
bool kdb::Key::isSpec ( ) const  [inline]
```

Determines if the key is in spec namespace.

**Return values**

| | |
|---|---|
| *true* | if it is a spec key |
| *false* | otherwise |

#### 572.30.5.16 isString()

```
bool kdb::Key::isString ( ) const  [inline]
```

Check if the value of `key` is of string type.
String values are null terminated and are not allowed to have any '\0' characters inside the string.
Make sure to use this function and don't test the string type another way to ensure compatibility and to write less error prone programs.

**Parameters**

| | |
|---|---|
| *key* | the Key to check |

**Return values**

| | |
|---|---|
| *1* | if the value of `key` is string |
| *0* | if the value of `key` is not string |
| *-1* | on NULL pointer |

**See also**

keyGetString(), keySetString() for getting / setting a Key's value as string

#### 572.30.5.17 isSystem()

```
bool kdb::Key::isSystem ( ) const  [inline]
```

Determines if the key is in system namespace.

**Return values**

| | |
|---|---|
| *true* | if it is a system key |
| *false* | otherwise |

#### 572.30.5.18 isUser()

```
bool kdb::Key::isUser ( ) const  [inline]
```

Determines if the key is in user namespace.

**Return values**

| | |
|---|---|
| *true* | if it is a user key |
| *false* | otherwise |

#### 572.30.5.19 isValid()

```
bool kdb::Key::isValid ( ) const  [inline]
```

**Returns**

if the key is valid

An invalid key has no name. The name of valid keys either start with user or system.

**Return values**

| | |
|---|---|
| *true* | if the key has a valid name |

**Return values**

| | |
|---|---|
| *false* | if the key has an invalid name |

**See also**

getName(), isUser(), isSystem(), getNamespace()

### 572.30.5.20 isValueLocked()

```
bool kdb::Key::isValueLocked ( ) const  [inline]
```

**Returns**

true if the value of our key has been locked

### 572.30.5.21 operator bool()

```
kdb::Key::operator bool ( ) const  [inline]
```

This is for loops and lookups only.
Opposite of isNull()
For loops it checks if there are still more keys. For lookups it checks if a key could be found.

**Warning**

you should not construct or use null keys

**See also**

isNull(), isValid()

**Returns**

false on null keys

true otherwise

### 572.30.5.22 operator"!=()

```
bool kdb::Key::operator!= (
            const Key & k ) const  [inline]
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

#### Note

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

#### Precondition

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

#### Invariant

All parts of the Keys remain unchanged

#### Postcondition

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

#### Parameters

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

#### Return values

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

#### Since

1.0.0

#### See also

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

#### Return values

| | |
|---|---|
| *true* | != 0 |

### 572.30.5.23 operator*()

```
ckdb::Key * kdb::Key::operator* ( ) const  [inline]
```

Is an abbreviation for getKey.

Passes out the raw key pointer. This pointer can be used to directly change the underlying key object.

**Note**

that the ownership remains in the object

**See also**

getKey()

### 572.30.5.24 operator++() [1/2]

```
void kdb::Key::operator++ ( ) const  [inline]
```

Increment the reference counter of a Key object.
As long as the reference counter is non-zero, `keyDel()` operations on `key` will be a no-op and return an error code.
Elektra's system for reference counting is not based on a concept of shared ownership. It is more similar to a shared lock, where the counter is used to keep track of how many clients hold the lock.
Initially, the reference counter will be 0. This is can be interpreted as the lock being unlocked. When you increment the reference counter, the lock becomes locked and `keyDel()` is blocked and fails. Only when the reference counter is fully decremented back down to 0 again, will `keyDel()` work again.

**Note**

The reference counter can never exceed `UINT16_MAX - 1`. `UINT16_MAX` is reserved as an error code.

**Postcondition**

`key's` reference counter is > 0

`key's` reference counter is <= UINT16_MAX - 1

**Parameters**

| | |
|---|---|
| *key* | the Key object whose reference counter should be increased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *UINT16_MAX* | when the reference counter already was the maximum value `UINT16_MAX - 1`, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

keyGetRef() to retrieve the current reference count

keyDecRef() for decreasing the reference counter

keyDel() for deleting a Key

### 572.30.5.25 operator++() [2/2]

```
void kdb::Key::operator++ (
            int  ) const  [inline]
```

Increment the reference counter of a Key object.
As long as the reference counter is non-zero, `keyDel()` operations on `key` will be a no-op and return an error code.
Elektra's system for reference counting is not based on a concept of shared ownership. It is more similar to a shared lock, where the counter is used to keep track of how many clients hold the lock.
Initially, the reference counter will be 0. This is can be interpreted as the lock being unlocked. When you increment the reference counter, the lock becomes locked and `keyDel()` is blocked and fails. Only when the reference counter is fully decremented back down to 0 again, will `keyDel()` work again.

**Note**

The reference counter can never exceed `UINT16_MAX - 1`. `UINT16_MAX` is reserved as an error code.

**Postcondition**

`key's` reference counter is > 0

`key's` reference counter is <= UINT16_MAX - 1

**Parameters**

| | |
|---|---|
| *key* | the Key object whose reference counter should be increased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *UINT16_MAX* | when the reference counter already was the maximum value `UINT16_MAX - 1`, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

keyGetRef() to retrieve the current reference count

keyDecRef() for decreasing the reference counter

keyDel() for deleting a Key

### 572.30.5.26 operator--() [1/2]

```
void kdb::Key::operator-- ( ) const  [inline]
```

Decrement the reference counter of a Key object.
As long as the reference counter is non-zero, `keyDel()` operations on `key` will be a no-op and return an error code.

**Postcondition**

`key's` reference counter is >= 0

`key's` reference counter is < SSIZE_MAX

**Parameters**

| | |
|---|---|
| *key* | the Key object whose reference counter should get decreased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *0* | when the reference counter already was the minimum value 0, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

keyGetRef() to retrieve the current reference count

keyIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

keyDel() for deleting a Key

### 572.30.5.27 operator--() [2/2]

```
void kdb::Key::operator-- (
            int  ) const  [inline]
```

Decrement the reference counter of a Key object.
As long as the reference counter is non-zero, `keyDel()` operations on `key` will be a no-op and return an error code.

**Postcondition**

`key`'s reference counter is >= 0

`key`'s reference counter is < SSIZE_MAX

**Parameters**

| | |
|---|---|
| *key* | the Key object whose reference counter should get decreased |

**Returns**

the updated value of the reference counter

**Return values**

| | |
|---|---|
| *UINT16_MAX* | on NULL pointer |
| *0* | when the reference counter already was the minimum value 0, the reference counter will not be modified in this case |

**Since**

1.0.0

**See also**

keyGetRef() to retrieve the current reference count

keyIncRef() for increasing the reference counter and for a more complete explanation of the reference counting system

keyDel() for deleting a Key

### 572.30.5.28 operator->()

```
Key * kdb::Key::operator-> ( )  [inline]
```

**Returns**

a pointer to this object

Needed for KeySet iterators.

**See also**

KeySetIterator

### 572.30.5.29 operator<()

```
bool kdb::Key::operator< (
            const Key & other ) const  [inline]
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

**Note**

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

**Precondition**

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

**Invariant**

All parts of the Keys remain unchanged

**Postcondition**

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

**Parameters**

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

**Return values**

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

**Since**

1.0.0

**See also**

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

**Return values**

| | |
|---|---|
| *true* | < 0 |

### 572.30.5.30 operator<=()

```
bool kdb::Key::operator<= (
            const Key & other ) const  [inline]
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

#### Note

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

#### Precondition

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

#### Invariant

All parts of the Keys remain unchanged

#### Postcondition

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

#### Parameters

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

#### Return values

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

#### Since

1.0.0

#### See also

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

#### Return values

| | |
|---|---|
| *true* | <= 0 |

### 572.30.5.31 operator=() [1/2]

```
Key & kdb::Key::operator= (
            ckdb::Key * k )  [inline]
```

Assign a C key.
Will call del() on the old key.

#### 572.30.5.32 operator=() [2/2]

```
Key & kdb::Key::operator= (
            const Key & k )  [inline]
```

Assign a key.
Will call del() on the old key.

#### 572.30.5.33 operator==()

```
bool kdb::Key::operator== (
            const Key & k ) const  [inline]
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

**Note**

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

**Precondition**

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

**Invariant**

All parts of the Keys remain unchanged

**Postcondition**

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

**Parameters**

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

**Return values**

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

#### Since

1.0.0

#### See also

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

#### Return values

| | |
|---|---|
| *true* | == 0 |

### 572.30.5.34 operator>()

```
bool kdb::Key::operator> (
            const Key & other ) const  [inline]
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

#### Note

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

#### Precondition

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

#### Invariant

All parts of the Keys remain unchanged

#### Postcondition

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

**Parameters**

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

**Return values**

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

**Since**

1.0.0

**See also**

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

**Return values**

| | |
|---|---|
| *true* | > 0 |

### 572.30.5.35 operator>=()

```
bool kdb::Key::operator>= (
            const Key & other ) const  [inline]
```

Compare the name of two Keys.
The comparison is based on a memcmp of the Key's names. If the names match, the Keys are found to be exactly the same and 0 is returned. These two keys can't be used in the same KeySet.
keyCmp() defines the sorting order for a KeySet.
The following 3 points are the rules for NULL values:

- A NULL pointer will be found to be smaller than every other Key. If both are NULL pointers, 0 is returned.
- A NULL name will be found to be smaller than every other name. If both are NULL names, 0 is returned.

If the name is equal then:

- No owner will be found to be smaller than every other owner. If both don't have an owner, 0 is returned.

**Note**

the owner will only be used if the names are equal.

Given any Keys k1 and k2 constructed with keyNew(), following equation hold true:

```
succeed_if (keyCmp (0, 0) == 0, "all null pointers same");
succeed_if (keyCmp (k1, 0) == 1, "null pointer is smaller");
succeed_if (keyCmp (0, k2) == -1, "null pointer is smaller");
```

Here are some more examples:

```
Key *k1 = keyNew("user:/a", KEY_END);
Key *k2 = keyNew("user:/b", KEY_END);
// keyCmp(k1,k2) < 0
// keyCmp(k2,k1) > 0
```

Do not strcmp the keyName() yourself, because the result differs from simple ascii comparison.

**Precondition**

The Keys `k1` and `k2` have been properly initialized via keyNew() or are NULL

**Invariant**

All parts of the Keys remain unchanged

**Postcondition**

If the result is 0, `k1` and `k2` cannot be used in the same KeySet

**Parameters**

| | |
|---|---|
| *k1* | the first Key to be compared |
| *k2* | the second Key to be compared |

**Return values**

| | |
|---|---|
| *<0* | if k1 < k2 |
| *0* | if k1 == k2 |
| *>0* | if k1 > k2 |

**Since**

1.0.0

**See also**

ksAppendKey(), ksAppend() will compare Keys via keyCmp() when appending

ksLookup() will compare Keys via keyCmp() during searching

**Return values**

| | |
|---|---|
| *true* | >= 0 |

### 572.30.5.36 release()

```
ckdb::Key * kdb::Key::release ( )  [inline]
```

Passes out the raw key pointer and resets internal key handle.

**Note**

that the ownership is moved outside.

**Return values**

| | |
|---|---|
| *0* | if no key is held (null pointer), no action is done then. |

### 572.30.5.37 set()

```
template<class T >
void kdb::Key::set (
            T x )  [inline]
```

Set a key value.
Set the value for `key` as `newStringValue`. The function will allocate and save a private copy of `newString↩Value`, so the parameter can be freed after the call.
String values will be saved in backend storage in UTF-8 universal encoding, regardless of the program's current encoding (if the iconv plugin is available).

#### Precondition

`newStringValue` is a NULL terminated string

#### Postcondition

Value of the Key is set to the UTF-8 encoded value of `newStringValue`

Metakey `meta:/binary` is cleared

#### Parameters

| | |
|---|---|
| *key* | the Key for which to set the string value |
| *newStringValue* | NULL-terminated string to be set as `key's` value |

#### Returns

the number of bytes actually saved in private struct including final NULL

#### Return values

| | |
|---|---|
| *1* | if `newStringValue` is a NULL pointer, this will make the string empty (string only containing null termination) |
| *-1* | if `key` is a NULL pointer |

#### Since

1.0.0

#### See also

keyString() for getting a pointer to the Key's value

keyGetString() for getting a copy of the Key's value

keySetBinary() for setting binary data

This method tries to deserialise the string to the given type.

### 572.30.5.38 setBaseName()

```
void kdb::Key::setBaseName (
            const std::string & baseName )  [inline]
```

Sets a base name for a key.
Sets `baseName` as the new basename for `key`. Only the basename of the Key will be affected.
A simple example is:

```
Key * k = keyNew ("user:/my/long/name", KEY_END);
keySetBaseName (k, "myname");
printf ("%s\n", keyName (k)); // will print user:/my/long/myname
keyDel (k);
```

All text after the last '/' in the Key's name is erased and baseName is appended. If baseName is 0 (NULL), then the last part of the Key's name is removed without replacement. The root name of the Key will not be removed though.
Let us suppose key has name "system:/dir1/dir2/key1". If baseName is "key2", the resulting key name will be "system:/dir1/dir2/key2". If baseName is 0 (NULL), the resulting key name will be "system:/dir1/dir2". If baseName is empty, the resulting key name will be "system:/dir1/dir2/%", where "%" denotes an empty base name, as also shown in the following code:

```
keySetName (k, "system:/valid");
keySetBaseName (k, "");
succeed_if_same_string (keyName (k), "system:/%");
succeed_if_same_string (keyBaseName (k), "");
```

keySetBaseName() does proper escaping on the supplied name argument.
You can use character sequences as baseName (e.g. "." (dot), ".." (dot-dot), "%" (empty basename)). They will be properly escaped and will not have their usual meaning.
If you want to add to the basename instead of changing it, use keyAddBaseName(). If you do not want any escaping, use keyAddName().

**Parameters**

| | |
|---|---|
| *key* | the Key whose basename to set |
| *baseName* | the new basename for the Key |

**Returns**

the size in bytes of the new key name

**Return values**

| | |
|---|---|
| *-1* | if Key is NULL |
| *-1* | if Key was inserted into KeySet before |
| *-1* | if Key is read-only |
| *-1* | on allocation errors |

**Since**

1.0.0

**See also**

keyAddBaseName() for adding a basename instead of changing it

keyAddName() for adding a name without escaping

keySetName() for setting a completely new name

Name Manipulation Methods for more details on special names

**Exceptions**

| | |
|---|---|
| *KeyInvalidName* | if the name is not valid |

### 572.30.5.39 setBinary()

```
ssize_t kdb::Key::setBinary (
            const void * newBinary,
            size_t dataSize ) [inline]
```

Set the value of a Key to the binary value newBinary.

A private copy of `newBinary` will be allocated and saved inside `key`, so the parameter can be deallocated after the call.
Binary values might be encoded in another way than string values depending on the plugin. Typically character encodings should not take place on binary data. Consider using a string Key instead, if encoding should occur.
When `newBinary` is a NULL pointer the value will be freed and 0 will be returned.
Read-only keys will stay unchanged after calling this function.

**Note**

The metadata "binary" will be set to mark that the key is binary from now on. When the Key is already binary the metadata won't be changed. This will only happen in the successful case, but not when -1 is returned.

**Precondition**

`dataSize` matches the length of `newBinary`

`newBinary` is not NULL and `dataSize` > 0

`key` is not read-only

**Postcondition**

`key`'s value set exactly to the data in `newBinary`

"binary" key set in `key`'s metadata

**Parameters**

| | |
|---|---|
| *key* | the Key object where the value should be set |
| *newBinary* | a pointer to any binary data or NULL (to clear the stored value) |
| *dataSize* | number of bytes to copy from `newBinary` |

**Returns**

the number of bytes actually copied to internal struct storage

**Return values**

| | |
|---|---|
| *0* | when the internal binary was freed and is now a null pointer |
| *-1* | if `key` is NULL |
| *-1* | when `dataSize` is 0 (and newBinary not NULL) or larger than SSIZE_MAX |
| *-1* | if `key` is read-only |

**Since**

1.0.0

**See also**

keyGetBinary() for getting a Key's value as binary

keyIsBinary() to check if the Key's value is binary

keyGetString() and keySetString() for working with string values

### 572.30.5.40 setMeta()

```
template<class T >
void kdb::Key::setMeta (
```

```
const std::string & metaName,
T x )  [inline]
```

Set metadata for key.

Set a new metadata Key. Will set a new metadata pair with name `metaName` and value `newMetaString`.
Will add a new metadata Key, if `metaName` was unused until now.
It will modify an existing Pair of metadata if `metaName` was already present.
It will remove a metadata Key if `newMetaString` is 0.
If `metaName` does not start with 'meta:/', it will be prefixed with 'meta:/'.

#### Precondition

`metaName` is prefixed with "meta:/"

`key's` metadata is not read-only

#### Postcondition

The value in `key's` metadata Keyset for `metaName` is `newMetaString`

#### Parameters

| | |
|---|---|
| *key* | Key whose metadata should be set |
| *metaName* | name of the metadata Key that should be set |
| *newMetaString* | new value for the metadata Key |

#### Returns

size (>0) of `newMetaString` if metadata has been successfully added

#### Return values

| | |
|---|---|
| *0* | if the meta-information for metaName was removed |
| *-1* | if key or metaName is 0 |
| *-1* | if system is out of memory |
| *-1* | if `metaName` is not a valid metadata name |

#### Since

1.0.0

#### See also

keyGetMeta() for getting the value of a metadata Key

keyMeta() for getting the KeySet containing metadata

#### Warning

unlike the C Interface, it is not possible to remove metadata with this method. k.setMeta("something", NULL) will lead to set the number 0 or to something different (may depend on compiler definition of NULL). See discussion in Issue https://github.com/ElektraInitiative/libelektra/issues/8

Use delMeta() to avoid these issues.

#### See also

delMeta(), getMeta(), copyMeta(), copyAllMeta()

### 572.30.5.41 setName()

```
void kdb::Key::setName (
            const std::string & newName )  [inline]
```

Set a new name to a Key.
A valid name is one of the forms:

- `spec:/something` for specification of other keys.
- `proc:/something` for in-memory keys, e.g. commandline.
- `dir:/something` for dir keys in current working directory
- `system:/something` for system keys in /etc or /
- `user:/something` for user keys in home directory
- `user:username/something` for other users (deprecated: kdbGet() + kdbSet() currently unsupported)
- `/something` for cascading keys (actually refers to one of the above, see also ksLookup())

An invalid name either has an invalid namespace or a wrongly escaped \ at the end of the name.
See key names for the exact rules.
The last form has explicitly set the owner, to let the library know in which user folder to save the Key. A owner is a user name. If it is not defined (the second form), current user is used.
You should always follow the guidelines for Key tree structure creation.
A private copy of the Key name will be stored, and the `newName` parameter can be freed after this call.
.., . and / will be handled as in filesystem paths. A valid name will be build out of the (valid) name what you pass, e.g. user:///sw/../sw//./../MyApp -> user:/sw/MyApp
Trailing slashes will be stripped.
On invalid names, the name stays unchanged.

#### Returns

size of the new Key name in bytes, including NULL terminator

#### Return values

| | |
|---|---|
| *-1* | if `key` or `keyName` is NULL or `keyName` is empty or invalid |
| *-1* | if Key was inserted to a KeySet before |
| *-1* | if Key name is read-only |

#### Parameters

| | |
|---|---|
| *key* | the Key whose name to set |
| *newName* | the new name for the Key |

#### Since

1.0.0

#### See also

keyGetName() for getting a copy of the Key's name

keyName() for getting a pointer to the Key's name

keySetBaseName(), keyAddBaseName() for manipulating the base name

**Exceptions**

| | |
|---|---|
| *KeyInvalidName* | if the name is not valid |

### 572.30.5.42 setNamespace()

```
ssize_t kdb::Key::setNamespace (
            ElektraNamespace ns ) const  [inline]
```

Set the namespace of the key.

**See also**

ElektraNamespace, keySetNamespace

### 572.30.5.43 setString()

```
void kdb::Key::setString (
            const char * newString )  [inline]
```

Set the value for `key` as `newStringValue`.
The function will allocate and save a private copy of `newStringValue`, so the parameter can be freed after the call.
String values will be saved in backend storage in UTF-8 universal encoding, regardless of the program's current encoding (if the iconv plugin is available).

**Precondition**

`newStringValue` is a NULL terminated string

**Postcondition**

Value of the Key is set to the UTF-8 encoded value of `newStringValue`

Metakey `meta:/binary` is cleared

**Parameters**

| | |
|---|---|
| *key* | the Key for which to set the string value |
| *newStringValue* | NULL-terminated string to be set as `key's` value |

**Returns**

the number of bytes actually saved in private struct including final NULL

**Return values**

| | |
|---|---|
| *1* | if `newStringValue` is a NULL pointer, this will make the string empty (string only containing null termination) |
| *-1* | if `key` is a NULL pointer |

**Since**

1.0.0

**See also**

keyString() for getting a pointer to the Key's value

keyGetString() for getting a copy of the Key's value

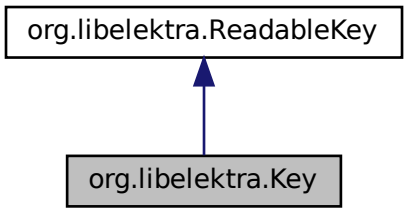
keySetBinary() for setting binary data

The documentation for this class was generated from the following files:

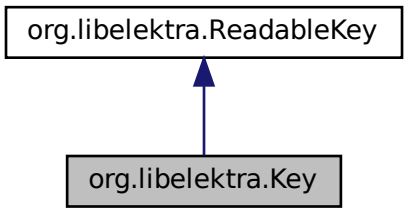
- key.hpp
- kdbvalue.hpp

## 572.31 org.libelektra.Key Class Reference

Key represents a native Elektra key providing access to its name, value and meta information.
Inheritance diagram for org.libelektra.Key:




Collaboration diagram for org.libelektra.Key:




### Classes

- enum CreateArgumentTag

  *Argument tags for use with create(String, Object...).*

### Public Member Functions

- byte[ ] getBinary ()
- Key setBoolean (boolean value)

  *Sets the key's value by converting.*

- Key setByte (byte value)

  *Sets the key's value by converting.*
- Key setShort (short value)

  *Sets the key's value by converting.*
- Key setInt (int value)

  *Sets the key's value by converting.*
- Key setLong (long value)

  *Sets the key's value by converting.*
- Key setFloat (float value)

  *Sets the key's value by converting.*
- Key setDouble (double value)

  *Sets the key's value by converting.*
- Key setString (String value)

  *Sets the key's value.*
- Key setBinary (byte[ ] value)

  *Sets the key's binary value.*
- Key setNull ()

  *Removes the key's value without changing the type.*
- Key setError (ErrorCode code, String reason)

  *Sets proper error meta for key.*
- Key addWarning (ErrorCode code, String reason)

  *Adds warning meta for key.*
- Key copy (Key source, int flags)

  *Copies the information from the.*
- boolean copyMeta (Key source, String metaName)

  *Copies some meta information from a.*
- boolean copyAllMeta (Key source)

  *Copies all meta information from a.*
- Optional< ReadableKey > getMeta (String metaName)

  *Getter for meta information.*
- Key setMeta (String metaName, String newMetaString)

  *Sets meta information.*
- Key removeMeta (String metaName)

  *Removes meta information.*
- KeySet meta ()

  *Get KeySet with metakeys.*
- Key setName (String name)

  *Sets the key's name.*
- Key setBaseName (String baseName) throws KeyNameException

  *Sets the key's base name; will replace current base name with new base name.*
- Key addBaseName (String baseName)

  *Adds key base name; will add given base name to current key so that new key is sub key of current key.*
- Iterator< ReadableKey > iterator ()

## Static Public Member Functions

- static Key create ()

  *Constructs a temporary nameless Key which cannot be saved to the key data base but used for transferring warnings and error information.*
- static Key create (String name, @Nullable Object value, Key... meta)

  *Constructs a new Key with the specified content and arguments*

- static Key create (String name, Key... meta)

  *Basic constructor of key class.*

## Protected Member Functions

- Key (long nativePointer, boolean suppressCleanUp)

  *Constructor associating a new Key instance with a native pointer in long format*

  *Suppressing clean-up has been introduced for usage of this binding as JNI plug-in and should normally not be used in any other case.*
- Key (Pointer pointer)

  *Constructor associating a new Key instance with a JNA pointer.*
- void release ()

  *Clean-up method to release key reference by first decrementing its reference counter and then trying to free the native reference*

  *keys, will get cleaned up by garbage collection as soon as they get phantom reachable.*

## Static Protected Member Functions

- static Optional< Key > create (@Nullable Pointer pointer)

  *Constructs a new Key instance associated with a JNA pointer.*
- static Key create (String name, Object... args)

  *Constructs a new Key with the specified content and arguments*

## Additional Inherited Members

### 572.31.1 Detailed Description

Key represents a native Elektra key providing access to its name, value and meta information.

### 572.31.2 Constructor & Destructor Documentation

#### 572.31.2.1 Key() [1/2]

```
org.libelektra.Key.Key (
            long nativePointer,
            boolean suppressCleanUp )  [inline], [protected]
```

Constructor associating a new Key instance with a native pointer in long format

Suppressing clean-up has been introduced for usage of this binding as JNI plug-in and should normally not be used in any other case.

**Parameters**

| | |
|---|---|
| *nativePointer* | Native pointer to key in long format |
| *suppressCleanUp* | True to suppress native reference clean-up as soon as this Key instance becomes phantom reachable, false otherwise @implNote Increased the native key's reference counter, even if `suppressCleanUp` is `true` |

#### 572.31.2.2 Key() [2/2]

```
org.libelektra.Key.Key (
```

```
            Pointer pointer )  [inline], [protected]
```

Constructor associating a new Key instance with a JNA pointer.

**Parameters**

| | |
|---|---|
| *pointer* | JNA Pointer to key |

## 572.31.3 Member Function Documentation

### 572.31.3.1 addBaseName()

```
Key org.libelektra.Key.addBaseName (
            String baseName )  [inline]
```

Adds key base name; will add given base name to current key so that new key is sub key of current key.

**Parameters**

| | |
|---|---|
| *baseName* | New key base name to add |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *KeyNameException* | if `baseName` is invalid, the key was inserted in a key set before or the key name is read-only |
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `baseName` is blank |

### 572.31.3.2 addWarning()

```
Key org.libelektra.Key.addWarning (
            ErrorCode code,
            String reason )  [inline]
```

Adds warning meta for key.

**Parameters**

| | |
|---|---|
| *code* | ErrorCode of the warning |
| *reason* | Reason for the error |

**Returns**

This Key, enabling a fluent interface

### 572.31.3.3 copy()

```
Key org.libelektra.Key.copy (
            Key source,
            int flags )  [inline]
```

Copies the information from the.
`source`
key into **this** key.

**Parameters**

| | |
|---|---|
| *source* | Source Key object containing the information to copy |
| *flags* | Flags indicating which parts of the key to copy<br>Example:<br>KEY_CP_NAME \| KEY_CP_VALUE |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *KeyException* | if copying failed |
| *IllegalStateException* | if this or the<br>`source`<br>Key has already been released |
| *IllegalArgumentException* | if<br>`source`<br>is<br>`null` |

**See also**

dup()

dup(int)

KEY_CP_ALL

KEY_CP_META

KEY_CP_NAME

KEY_CP_STRING

KEY_CP_VALUE

### 572.31.3.4 copyAllMeta()

```
boolean org.libelektra.Key.copyAllMeta (
            Key source )  [inline]
```

Copies all meta information from a.
`source`
key to this key

**Parameters**

| | |
|---|---|
| *source* | Key used as source |

**Returns**

True, if meta was successfully copied, false if `source`

does not contain any meta and nothing had to be done

**Exceptions**

| | |
|---|---|
| *KeyMetaException* | if copying failed |
| *IllegalStateException* | if this or the `source` Key has already been released |
| *IllegalArgumentException* | if `source` is `null` |

**See also**

copyMeta(Key, String)

### 572.31.3.5 copyMeta()

```
boolean org.libelektra.Key.copyMeta (
            Key source,
            String metaName )  [inline]
```

Copies some meta information from a.

`source`

key to this key

**Parameters**

| | |
|---|---|
| *source* | Key used as source |
| *metaName* | Key name of the meta information to be copied |

**Exceptions**

**Returns**

True, if meta was successfully copied, false if source does not contain the specified meta information and nothing had to be done

**Exceptions**

| | |
|---|---|
| *KeyMetaException* | if this key's meta information is read-only of copying failed |
| *IllegalStateException* | if this or the `source` Key has already been released |
| *IllegalArgumentException* | if `source` is `null` or `metaName` is blank |

**See also**

copyAllMeta(Key)

### 572.31.3.6 create() [1/5]

```
static Key org.libelektra.Key.create ( )  [inline], [static]
```

Constructs a temporary nameless Key which cannot be saved to the key data base but used for transferring warnings and error information.

**Returns**

New nameless key

**Exceptions**

| | |
|---|---|
| *KeyException* | on allocation problems |

### 572.31.3.7 create() [2/5]

```
static Optional<Key> org.libelektra.Key.create (
            @Nullable Pointer pointer )  [inline], [static], [protected]
```

Constructs a new Key instance associated with a JNA pointer.

**Parameters**

| | |
|---|---|
| *pointer* | Optional JNA Pointer to key |

**Returns**

New Key instance if `pointer` is non-null, Optional#empty() otherwise

#### 572.31.3.8 create() [3/5]

```
static Key org.libelektra.Key.create (
            String name,
            @Nullable Object value,
            Key... meta ) [inline], [static]
```

Constructs a new Key with the specified content and arguments

**Parameters**

| | |
|---|---|
| *name* | Name of the key (first part of key-value pair) |
| *value* | Optional Value of key. will be determine from the object by calling {}. To set a binary value, please see #setBinary(byte[ ])}. meta Metadata that should be added to this key, null keys will be f |

#### 572.31.3.9 create() [4/5]

```
static Key org.libelektra.Key.create (
            String name,
            Key... meta ) [inline], [static]
```

Basic constructor of key class.

**Parameters**

| | |
|---|---|
| *name* | Key name; first part of key-value pair |
| *meta* | Metadata that should be added to this key. Will filter null values. |

**Returns**

New key object

**Exceptions**

| | |
|---|---|
| *KeyException* | if the key name is invalid or there have been allocation problems |

#### 572.31.3.10 create() [5/5]

```
static Key org.libelektra.Key.create (
            String name,
            Object... args ) [inline], [static], [protected]
```

Constructs a new Key with the specified content and arguments

**Parameters**

| | |
|---|---|
| *name* | Key name; first part of key-value pair |
| *args* | Arguments used for key value<br>Example:<br>CreateArgumentTag#KEY_VALUE, "custom key value", CreateArgumentTag#KEY_END |

**Returns**

New key

**Exceptions**

| | |
|---|---|
| *KeyException* | if the key name is invalid or there have been allocation problems |

**See also**

CreateArgumentTag

### 572.31.3.11 getBinary()

```
byte [] org.libelektra.Key.getBinary ( )  [inline]
```

**Returns**

This key's value as string

**Exceptions**

| | |
|---|---|
| *KeyBinaryValueException* | if the underlying native key is not of type binary |
| *IllegalStateException* | if this Key has already been released |

### 572.31.3.12 getMeta()

```
Optional<ReadableKey> org.libelektra.Key.getMeta (
            String metaName )  [inline]
```

Getter for meta information.

**Parameters**

| | |
|---|---|
| *metaName* | Key name of meta information to be fetched |

**Returns**

New ReadableKey object containing the requested meta information or {}, if metaName was not found IllegalStateException if thi

### 572.31.3.13 iterator()

```
Iterator<ReadableKey> org.libelektra.Key.iterator ( )  [inline]
```

**Returns**

KeySetIterator for the meta data of this Key

### 572.31.3.14 meta()

```
KeySet org.libelektra.Key.meta ( )  [inline]
```

Get KeySet with metakeys.

**Returns**

A KeySet with all metakeys if the given key

**Exceptions**

| | |
|---|---|
| *KeyMetaException* | if `k` is invalid |
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `k` is null} |

### 572.31.3.15 removeMeta()

```
Key org.libelektra.Key.removeMeta (
            String metaName )  [inline]
```

Removes meta information.

**Parameters**

| | |
|---|---|
| *metaName* | Key name of meta information to be removed |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *KeyMetaException* | if `metaName` is invalid |
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `metaName` is blank |

### 572.31.3.16 setBaseName()

```
Key org.libelektra.Key.setBaseName (
            String baseName ) throws KeyNameException  [inline]
```

Sets the key's base name; will replace current base name with new base name.

**Parameters**

| | |
|---|---|
| *baseName* | New key base name to use |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *KeyNameException* | if `baseName` is invalid, the key was inserted in a key set before or the key name is read-only |
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `baseName` is `null` |

### 572.31.3.17 setBinary()

```
Key org.libelektra.Key.setBinary (
            byte[] value )  [inline]
```

Sets the key's binary value.

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `value` is `null` |
| *KeyException* | if the key's value is read-only or there have been allocation problems |

### 572.31.3.18 setBoolean()

```
Key org.libelektra.Key.setBoolean (
            boolean value )  [inline]
```

Sets the key's value by converting.
`value`
to string

**See also**

Definition of Bool

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

### 572.31.3.19 setByte()

```
Key org.libelektra.Key.setByte (
            byte value )  [inline]
```

Sets the key's value by converting.

`value`

to string

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

### 572.31.3.20 setDouble()

```
Key org.libelektra.Key.setDouble (
            double value )  [inline]
```

Sets the key's value by converting.

`value`

to string

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

#### 572.31.3.21 setError()

```
Key org.libelektra.Key.setError (
            ErrorCode code,
            String reason )  [inline]
```

Sets proper error meta for key.

**Parameters**

| | |
|---|---|
| *code* | ErrorCode of the error |
| *reason* | Reason for the error |

**Returns**

This Key, enabling a fluent interface

#### 572.31.3.22 setFloat()

```
Key org.libelektra.Key.setFloat (
            float value )  [inline]
```

Sets the key's value by converting.

`value`

to string

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

#### 572.31.3.23 setInt()

```
Key org.libelektra.Key.setInt (
            int value )  [inline]
```

Sets the key's value by converting.

`value`

to string

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

### 572.31.3.24 setLong()

```
Key org.libelektra.Key.setLong (
            long value )  [inline]
```

Sets the key's value by converting.

`value`

to string

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

### 572.31.3.25 setMeta()

```
Key org.libelektra.Key.setMeta (
            String metaName,
            String newMetaString )  [inline]
```

Sets meta information.

**Parameters**

| | |
|---|---|
| *metaName* | Key name of meta information to be set |
| *newMetaString* | Meta value to be set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *KeyMetaException* | if `metaName` is invalid |
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `metaName` is blank or `newMetaString` is `null` |

### 572.31.3.26 setName()

```
Key org.libelektra.Key.setName (
            String name )  [inline]
```

Sets the key's name.

#### Parameters

| | |
|---|---|
| *name* | New key name to use |

#### Returns

This Key, enabling a fluent interface

#### Exceptions

| | |
|---|---|
| *KeyNameException* | if `name` is invalid, the key was inserted in a key set before or the key name is read-only |
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `baseName` is blank |

### 572.31.3.27 setNull()

```
Key org.libelektra.Key.setNull ( )  [inline]
```

Removes the key's value without changing the type.

#### Returns

This Key, enabling a fluent interface

#### Exceptions

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |
| *KeyException* | if the key's value is read-only or there have been allocation problems |

### 572.31.3.28 setShort()

```
Key org.libelektra.Key.setShort (
            short value )  [inline]
```

Sets the key's value by converting.
`value`
to string

#### Parameters

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |

#### 572.31.3.29 setString()

```
Key org.libelektra.Key.setString (
            String value )  [inline]
```

Sets the key's value.

**Parameters**

| | |
|---|---|
| *value* | Value to set |

**Returns**

This Key, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this Key has already been released |
| *IllegalArgumentException* | if `value` is `null` |
| *KeyException* | if the key's value is read-only or there have been allocation problems |

The documentation for this class was generated from the following file:

- Key.java

## 572.32 org.libelektra.exception.KeyBinaryValueException Class Reference

Indicates a key's underlying native key value is not of type binary, and therefore is not compatible with the invoked functionality raising this exception.
Inherits IllegalStateException.

### 572.32.1 Detailed Description

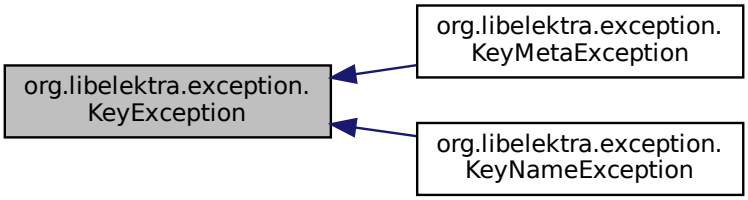
Indicates a key's underlying native key value is not of type binary, and therefore is not compatible with the invoked functionality raising this exception.
The documentation for this class was generated from the following file:

- KeyBinaryValueException.java

## 572.33 org.libelektra.exception.KeyException Class Reference

Indicates a generic exception while calling one of Key's methods

This exception is related to unrecoverable C API specific errors (primarily allocation problems).
Inheritance diagram for org.libelektra.exception.KeyException:




### 572.33.1 Detailed Description

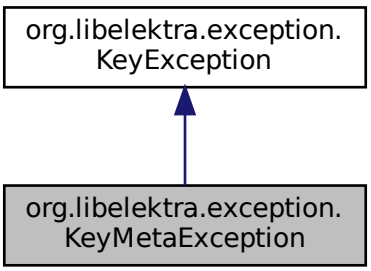
Indicates a generic exception while calling one of Key's methods

This exception is related to unrecoverable C API specific errors (primarily allocation problems).
All other exceptional states are represented by more specific exceptions. For more detailed information see exceptions in this package as well as a method's individual documentation.
The documentation for this class was generated from the following file:

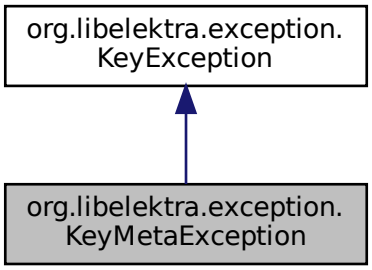
- KeyException.java

## 572.34 org.libelektra.exception.KeyMetaException Class Reference

Indicates Key#copyMeta(Key, String), Key#copyAllMeta(Key), {} or Key#removeMeta(String)} failed because the key name is invalid, t
Inheritance diagram for org.libelektra.exception.KeyMetaException:



Collaboration diagram for org.libelektra.exception.KeyMetaException:



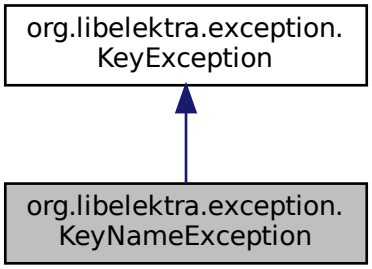

### 572.34.1 Detailed Description

Indicates Key#copyMeta(Key, String), Key#copyAllMeta(Key), {} or Key#removeMeta(String)} failed because the key name is invalid, t
The documentation for this class was generated from the following file:

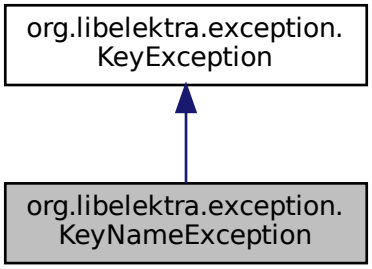
- KeyMetaException.java

## 572.35 org.libelektra.exception.KeyNameException Class Reference

Indicates Key#setName(String), Key#setBaseName(String) or {} failed because the key name is invalid, the key was inserted in a key
Inheritance diagram for org.libelektra.exception.KeyNameException:



Collaboration diagram for org.libelektra.exception.KeyNameException:




### 572.35.1 Detailed Description

Indicates Key#setName(String), Key#setBaseName(String) or {} failed because the key name is invalid, the key was inserted in a key
The documentation for this class was generated from the following file:

- KeyNameException.java

## 572.36 kdb::KeySet Class Reference

A keyset holds together a set of keys.
`#include <keyset.hpp>`

### Public Member Functions

- KeySet ()
  *Creates a new empty keyset with no keys.*
- KeySet (ckdb::KeySet ∗k)
  *Take ownership of a `ckdb::KeySet *`.*
- KeySet (const KeySet &other)
  *Duplicate a keyset.*
- KeySet (size_t alloc,...) ELEKTRA_SENTINEL
  *Create a new keyset.*
- KeySet (VaAlloc alloc, va_list ap)
  *Create a new keyset.*
- ~KeySet ()
  *Deconstruct a keyset.*
- ckdb::KeySet ∗ release ()
  *If you don't want destruction of keyset at the end you can release the pointer.*
- ckdb::KeySet ∗ getKeySet () const
  *Passes out the raw keyset pointer.*
- KeySet & operator= (KeySet const &other)
  *Duplicate a keyset.*
- ssize_t size () const
  *The size of the keyset.*
- KeySet dup () const
  *Duplicate a keyset.*

- void copy (const KeySet &other)

  *Copy a keyset.*
- void clear ()

  *Clear the keyset.*
- ssize_t append (const Key &toAppend)

  *append a key*
- ssize_t append (const KeySet &toAppend)

  *append a keyset*
- Key pop ()

  *Remove and return the last Key of `ks`.*
- Key at (elektraCursor pos) const

  *Lookup a key by index.*
- KeySet cut (const Key &k)

  *Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.*
- KeySet cut (std::string const &name)

  *Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.*
- Key lookup (const Key &k, const elektraLookupFlags options=KDB_O_NONE) const

  *Look for a Key contained in `ks` that matches the name of the `key`.*
- Key lookup (std::string const &name, const elektraLookupFlags options=KDB_O_NONE) const

  *Lookup a key by name.*
- template<typename T >
  T get (std::string const &name, const elektraLookupFlags options=KDB_O_NONE) const

  *Generic lookup+get for keysets.*
- ssize_t search (const Key &toSearch) const

  *Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.*
- ssize_t search (std::string const &name) const

  *Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.*

### 572.36.1 Detailed Description

A keyset holds together a set of keys.
Methods to manipulate KeySets. A KeySet is a set of keys.
Most important properties of a KeySet:

- Allows us to iterate over all keys (in any depth)
- Iteration is always sorted
- Fast key lookup
- A Key may be shared among many KeySets.

The most important methods of KeySet:

- With ksNew() you can create a new KeySet.
- You can append keys with ksAppendKey() or with ksAppend() you can append a whole keyset.
- Using ksLookup() you can lookup (or pop with KDB_O_POP) a key.
- With ksGetSize() and ksAtCursor() you can iterate through the keyset. Be assured that you will get every key of the set in a stable order (parents before children).

KeySet is the most important data structure in Elektra. It makes it possible to get and store many keys at once inside the database. In addition to that, the class can be used as high level datastructure in applications and it can be used in plugins to manipulate or check configuration.
With ksLookupByName() it is possible to fetch easily specific keys out of the list of keys.
You can easily create and iterate keys:

```
// create a new keyset with 3 keys
// with a hint that about 20 keys will be inside
KeySet * myConfig = ksNew (20, keyNew ("user:/name1", KEY_END), keyNew ("user:/name2", KEY_END), keyNew
       ("user:/name3", KEY_END), KS_END);
// append a key in the keyset
ksAppendKey (myConfig, keyNew ("user:/name4", KEY_END));
Key * current;
for (elektraCursor it = 0; it < ksGetSize (myConfig); ++it)
{
        current = ksAtCursor (myConfig, it);
        printf ("Key name is %s.\n", keyName (current));
}
ksDel (myConfig); // delete keyset and all keys appended
```

**Copy-on-Write**

Keysets employ copy-on-write techniques to minimize memory footprint. If you create a copy or a duplication of a keyset, the resulting keyset initially references the same data as the source keyset. Only if add or remove keys from a keyset additional memory is allocated.

**Invariant**

always holds an underlying elektra keyset.

**Note**

that the cursor is mutable, so it might be changed even in const functions as described.

## 572.36.2 Constructor & Destructor Documentation

### 572.36.2.1 KeySet() [1/5]

`kdb::KeySet::KeySet ( ) [inline]`

Creates a new empty keyset with no keys.
Allocate, initialize and return a new KeySet object. Objects created with ksNew() must be destroyed with ksDel().
You can use an arbitrary long list of parameters to preload the KeySet with a list of Keys. Either your first and only parameter is 0 or your last parameter must be KS_END.
So, terminate with ksNew(0, KS_END) or ksNew(20, ..., KS_END)

**Warning**

Never use ksNew(0, keyNew(...), KS_END). If the first parameter is 0, other arguments are ignored.

The first parameter `alloc` defines how many Keys can be added without reallocation. If you pass any alloc size greater than 0, but less than 16, it will default to 16.
For most uses

```
KeySet * keys = ksNew (1, KS_END);
// enough memory for up to 16 keys, without needing reallocation
ksDel (keys);
```

will be fine. The alloc size will be 16 and will double whenever size reaches alloc size, so it also performs well with large KeySets.
You can defer the allocation of the internal array that holds the Keys, by passing 0 as the alloc size. This is useful if it is unclear whether your KeySet will actually hold any Keys and you want to avoid a malloc call.

```
// Create KeySet without allocating memory for keys
KeySet * keys = ksNew (0, KS_END);
// The first allocation will happen in ksAppendKey
ksAppendKey(keys, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END));
// work with the KeySet
ksDel (keys);
```

If the size of the KeySet is known in advance, use the `alloc` parameter to hint the size of the KeySet.
If your application only needs up to 15 Keys you can request a KeySet of size 15:

```
KeySet * keys = ksNew (15, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key01", KEY_VALUE,
       "value01", KEY_END),
                          keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END),
                          keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key03", KEY_VALUE, "value03",
       KEY_END),
                          // ...
                          keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key15", KEY_VALUE, "value15",
       KEY_END), KS_END);
// work with it
ksDel (keys);
```

If you start having 3 Keys, and your application needs approximately 200 up to 500 Keys, you can use:

```
KeySet * config = ksNew (500, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key1", KEY_VALUE,
       "value1", KEY_END),
                          keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key2", KEY_VALUE, "value2",
       KEY_END),
                          keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key3", KEY_VALUE, "value3",
       KEY_END),
                          KS_END); // don't forget the KS_END at the end!
// work with it
ksDel (config);
```

Alloc size is 500, the size of the KeySet will be 3 after ksNew. This means the KeySet will reallocate when appending more than 497 keys.
The main benefit of taking a list of variant length parameters is to be able to have one C-Statement for any possible KeySet. If you prefer, you can always create an empty KeySet and use ksAppendKey().

#### Postcondition

the KeySet is rewinded properly

#### Parameters

| | |
|---|---|
| *alloc* | gives a hint for how many Keys may be stored initially |

#### Returns

a ready to use KeySet object

#### Return values

| | |
|---|---|
| *0* | on memory error |

#### Since

1.0.0

#### See also

ksDel() to free the KeySet afterwards

ksDup() to duplicate an existing KeySet

ksAppendKey() to append individual Keys to a KeySet

### 572.36.2.2 KeySet() [2/5]

```
kdb::KeySet::KeySet (
            ckdb::KeySet * keyset )  [inline]
```

Take ownership of a `ckdb::KeySet *`.
// TODO: use ksIncRef/ksDecRef

**Parameters**

| | |
|---|---|
| *keyset* | the `KeySet *` to take the ownership of |

**See also**

release()

### 572.36.2.3 KeySet() [3/5]

```
kdb::KeySet::KeySet (
            const KeySet & other )  [inline]
```

Duplicate a keyset.
This keyset will be a duplicate of the other afterwards.

**Note**

that they still reference to the same Keys, so if you change key values also the keys in the original keyset will be changed.

So it is shallow copy, to create a deep copy you have to dup() every key (it still won't copy metadata, but they are COW):

```
kdb::KeySet ksDeepCopy (kdb::KeySet orig)
{
        kdb::KeySet deepCopy;
        for (ssize_t it = 0; it < orig.size (); ++it)
        {
                deepCopy.append (orig.at (it).dup ());
        }
        return deepCopy;
}
```

**See also**

dup

### 572.36.2.4 KeySet() [4/5]

```
kdb::KeySet::KeySet (
            size_t alloc,
             ...  )  [inline], [explicit]
```

Create a new keyset.

**Parameters**

| | |
|---|---|
| *alloc* | minimum number of keys to allocate |
| *...* | variable argument list |

Allocate, initialize and return a new KeySet object. Objects created with ksNew() must be destroyed with ksDel(). You can use an arbitrary long list of parameters to preload the KeySet with a list of Keys. Either your first and only parameter is 0 or your last parameter must be KS_END.
So, terminate with ksNew(0, KS_END) or ksNew(20, ..., KS_END)

**Warning**

Never use ksNew(0, keyNew(...), KS_END). If the first parameter is 0, other arguments are ignored.

The first parameter `alloc` defines how many Keys can be added without reallocation. If you pass any alloc size greater than 0, but less than 16, it will default to 16.
For most uses

```
KeySet * keys = ksNew (1, KS_END);
```

```
// enough memory for up to 16 keys, without needing reallocation
ksDel (keys);
```

will be fine. The alloc size will be 16 and will double whenever size reaches alloc size, so it also performs well with large KeySets.

You can defer the allocation of the internal array that holds the Keys, by passing 0 as the alloc size. This is useful if it is unclear whether your KeySet will actually hold any Keys and you want to avoid a malloc call.

```
// Create KeySet without allocating memory for keys
KeySet * keys = ksNew (0, KS_END);
// The first allocation will happen in ksAppendKey
ksAppendKey(keys, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END));
// work with the KeySet
ksDel (keys);
```

If the size of the KeySet is known in advance, use the `alloc` parameter to hint the size of the KeySet.

If your application only needs up to 15 Keys you can request a KeySet of size 15:

```
KeySet * keys = ksNew (15, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key01", KEY_VALUE,
       "value01", KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key03", KEY_VALUE, "value03",
       KEY_END),
                       // ...
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key15", KEY_VALUE, "value15",
       KEY_END), KS_END);
// work with it
ksDel (keys);
```

If you start having 3 Keys, and your application needs approximately 200 up to 500 Keys, you can use:

```
KeySet * config = ksNew (500, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key1", KEY_VALUE,
       "value1", KEY_END),
                        keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key2", KEY_VALUE, "value2",
       KEY_END),
                        keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key3", KEY_VALUE, "value3",
       KEY_END),
                        KS_END); // don't forget the KS_END at the end!
// work with it
ksDel (config);
```

Alloc size is 500, the size of the KeySet will be 3 after ksNew. This means the KeySet will reallocate when appending more than 497 keys.

The main benefit of taking a list of variant length parameters is to be able to have one C-Statement for any possible KeySet. If you prefer, you can always create an empty KeySet and use ksAppendKey().

#### Postcondition

the KeySet is rewinded properly

#### Parameters

| | |
|---|---|
| *alloc* | gives a hint for how many Keys may be stored initially |

#### Returns

a ready to use KeySet object

#### Return values

| | |
|---|---|
| *0* | on memory error |

#### Since

1.0.0

#### See also

ksDel() to free the KeySet afterwards

ksDup() to duplicate an existing KeySet

ksAppendKey() to append individual Keys to a KeySet

**Precondition**

caller must call va_start and va_end

**va the list of arguments**

**Parameters**

| | |
|---|---|
| *alloc* | the allocation size |
| *va* | the list of variable arguments |

### 572.36.2.5 KeySet() [5/5]

```
kdb::KeySet::KeySet (
            VaAlloc alloc,
            va_list av )  [inline], [explicit]
```

Create a new keyset.

**Parameters**

| | |
|---|---|
| *alloc* | minimum number of keys to allocate |
| *ap* | variable arguments list |

Use va as first argument to use this constructor, e.g.:

```
KeySet ks(va, 23, ...);
```

Allocate, initialize and return a new KeySet object. Objects created with ksNew() must be destroyed with ksDel().
You can use an arbitrary long list of parameters to preload the KeySet with a list of Keys. Either your first and only parameter is 0 or your last parameter must be KS_END.
So, terminate with ksNew(0, KS_END) or ksNew(20, ..., KS_END)

**Warning**

Never use ksNew(0, keyNew(...), KS_END). If the first parameter is 0, other arguments are ignored.

The first parameter `alloc` defines how many Keys can be added without reallocation. If you pass any alloc size greater than 0, but less than 16, it will default to 16.
For most uses

```
KeySet * keys = ksNew (1, KS_END);
// enough memory for up to 16 keys, without needing reallocation
ksDel (keys);
```

will be fine. The alloc size will be 16 and will double whenever size reaches alloc size, so it also performs well with large KeySets.
You can defer the allocation of the internal array that holds the Keys, by passing 0 as the alloc size. This is useful if it is unclear whether your KeySet will actually hold any Keys and you want to avoid a malloc call.

```
// Create KeySet without allocating memory for keys
KeySet * keys = ksNew (0, KS_END);
// The first allocation will happen in ksAppendKey
ksAppendKey(keys, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END));
// work with the KeySet
ksDel (keys);
```

If the size of the KeySet is known in advance, use the `alloc` parameter to hint the size of the KeySet.
If your application only needs up to 15 Keys you can request a KeySet of size 15:

```
KeySet * keys = ksNew (15, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key01", KEY_VALUE,
       "value01", KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key02", KEY_VALUE, "value02",
       KEY_END),
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key03", KEY_VALUE, "value03",
       KEY_END),
                       // ...
                       keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key15", KEY_VALUE, "value15",
       KEY_END), KS_END);
```

```
// work with it
ksDel (keys);
```

If you start having 3 Keys, and your application needs approximately 200 up to 500 Keys, you can use:

```
KeySet * config = ksNew (500, keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key1", KEY_VALUE,
       "value1", KEY_END),
                        keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key2", KEY_VALUE, "value2",
       KEY_END),
                        keyNew ("user:/sw/org/app/#0/current/fixedConfiguration/key3", KEY_VALUE, "value3",
       KEY_END),
                        KS_END); // don't forget the KS_END at the end!
// work with it
ksDel (config);
```

Alloc size is 500, the size of the KeySet will be 3 after ksNew. This means the KeySet will reallocate when appending more than 497 keys.
The main benefit of taking a list of variant length parameters is to be able to have one C-Statement for any possible KeySet. If you prefer, you can always create an empty KeySet and use ksAppendKey().

#### Postcondition

the KeySet is rewinded properly

#### Parameters

| | |
|---|---|
| *alloc* | gives a hint for how many Keys may be stored initially |

#### Returns

a ready to use KeySet object

#### Return values

| | |
|---|---|
| *0* | on memory error |

#### Since

1.0.0

#### See also

ksDel() to free the KeySet afterwards

ksDup() to duplicate an existing KeySet

ksAppendKey() to append individual Keys to a KeySet

#### Precondition

caller must call va_start and va_end

#### va the list of arguments

#### Parameters

| | |
|---|---|
| *alloc* | the allocation size |
| *va* | the list of variable arguments |

#### 572.36.2.6 ~KeySet()

```
kdb::KeySet::~KeySet ( )  [inline]
```

Deconstruct a keyset.

A destructor for KeySet objects. Every KeySet created by ksNew() must be deleted with ksDel().

When the reference counter of `ks` is non-zero, this function will do nothing and simply return the current value of the reference counter.

It is therefore safe to call `ksDel (ks)` on any `KeySet * ks`.

**Parameters**

| | |
|---|---|
| *ks* | the KeySet object to delete |

**Return values**

| | |
|---|---|
| *0* | when the KeySet was freed |
| *-1* | on NULL pointers |

**Returns**

the value of the reference counter, if it was non-zero

**Since**

1.0.0

**See also**

ksNew() for creating a new KeySet

ksIncRef() for more information about the reference counter

### 572.36.3 Member Function Documentation

#### 572.36.3.1 append() [1/2]

```
ssize_t kdb::KeySet::append (
            const Key & toAppend )  [inline]
```

append a key

**Parameters**

| | |
|---|---|
| *toAppend* | key to append |

**Returns**

number of keys in the keyset

Appends a Key to the end of `ks`. Hands the ownership of the Key `toAppend` to the KeySet `ks`. ksDel(ks) uses keyDel(k) to delete every Key unless it got its reference counter incremented by keyIncRef(), e.g. by another KeySet that contains this Key.

The reference counter of the Key will be incremented to indicate this ownership, and thus `toAppend` is not const.

**See also**

keyGetRef()

If the Key's name already exists in the KeySet, it will be replaced with the new Key.
ksAppendKey() will also lock the Key's name from `toAppend`. This is necessary so that the order of the KeySet cannot be destroyed via calls to keySetName().
The KeySet internal cursor will be set to the new Key.
It is safe to directly append newly created Keys:

```
KeySet * ks = ksNew (1, KS_END);
ksAppendKey (ks, keyNew ("user:/my/new/key", KEY_END));
ksDel (ks);
// key deleted, too!
```

If you want the key to outlive the KeySet, make sure to do proper ref counting:

```
KeySet * ks = ksNew (1, KS_END);
Key * k = keyNew ("user:/ref/key", KEY_END);
keyIncRef (k);
ksAppendKey (ks, k);
ksDel (ks);
// now we still can work with the key k!
keyDecRef (k);
keyDel (k);
```

You can duplicate the Key to avoid aliasing, but then the Key in the KeySet has another identity:

```
KeySet * ks = ksNew (1, KS_END);
Key * k = keyNew ("user:/ref/key", KEY_END);
ksAppendKey (ks, keyDup (k, KEY_CP_ALL));
ksDel (ks);
// now we still can work with the key k!
keyDel (k);
```

**Parameters**

| | |
|---|---|
| *ks* | KeySet where `toAppend` should be append |
| *toAppend* | Key that will be appended to `ks` or deleted |

**Returns**

the size of the KeySet after appending

**Return values**

| | |
|---|---|
| *-1* | on NULL pointers |
| *-1* | if appending failed (only on memory problems). The Key will be deleted then. |

**Since**

1.0.0

**See also**

ksAppend() for appending a KeySet to another KeySet

keyIncRef() for manually increasing a Key's reference counter

### 572.36.3.2 append() [2/2]

```
ssize_t kdb::KeySet::append (
            const KeySet & toAppend )  [inline]
```

append a keyset

**Parameters**

| | |
|---|---|
| *toAppend* | keyset to append |

**Returns**

number of keys in the keyset

Append all Keys in `toAppend` to the end of the KeySet `ks`. `toAppend` KeySet will be left unchanged.
If a Key is both in `toAppend` and `ks`, the Key in `ks` will be overwritten.

**Postcondition**

Sorted KeySet ks with all Keys it had before and additionally the Keys from toAppend

**Parameters**

| | |
|---|---|
| *ks* | the KeySet that will receive the Keys |
| *toAppend* | the KeySet that provides the Keys that will be transferred |

**Returns**

the size of the KeySet `ks` after transfer

**Return values**

| | |
|---|---|
| *-1* | on NULL pointers |

**Since**

1.0.0

**See also**

ksAppendKey()

### 572.36.3.3 at()

```
Key kdb::KeySet::at (
            elektraCursor pos ) const  [inline]
```

Lookup a key by index.

**Parameters**

| | |
|---|---|
| *pos* | cursor position |

**Returns**

the found key

### 572.36.3.4 clear()

```
void kdb::KeySet::clear ( )  [inline]
```

Clear the keyset.
Keyset will be empty afterwards.

### 572.36.3.5 copy()

```
void kdb::KeySet::copy (
```

```
            const KeySet & other )  [inline]
```

Copy a keyset.
Replaces all keys in `this` with the ones from `other`. This is only a shallow copy. For a deep copy you need to manually Key::dup every key.

**Parameters**

| | |
|---|---|
| *other* | other keyset to copy |

Replace the content of a KeySet with another one. Most often you may want a duplicate of a KeySet, see ksDup() or append keys, see ksAppend(). In some situations you need to copy Keys from a KeySet to another KeySet, for which this function exists.

**Note**

You can also use it to clear a KeySet when you pass a NULL pointer as `source`.

**Implementation:**

First all Keys in `dest` will be deleted. Afterwards the content of `source` will be added to the destination.

A flat copy is made, so Keys will not be duplicated, but their reference counter is updated, so both KeySets need to be deleted via ksDel().

```
int f (KeySet *ks)
{
        KeySet *c = ksNew (20, ..., KS_END);
        // c receives keys
        ksCopy (ks, c); // pass the KeySet to the caller
        ksDel (c);
}       // caller needs to ksDel (ks)
```

**Parameters**

| | |
|---|---|
| *source* | an initialized KeySet or NULL |
| *dest* | an initialized KeySet, where the Keys from `source` get copied to |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | if `dest` was cleared successfully (`source` is NULL) |
| *-1* | when `dest` is a NULL pointer |

**Since**

1.0.0

**See also**

ksNew() for creating a new KeySet

ksDel() for deleting an existing KeySet

ksDup() for duplicating an existing KeySet

keyCopy() for copying Keys

### 572.36.3.6 cut() [1/2]

```
KeySet kdb::KeySet::cut (
            const Key & k )  [inline]
```

Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.
Searches for the `cutpoint` inside the KeySet `ks`. If found, it cuts out this Key and everything which is below (see keyIsBelow()) this Key. These Keys will be missing in the keyset `ks`. Instead, they will be moved to the returned KeySet. If `cutpoint` is not found an empty KeySet is returned and `ks` is not changed.
The cursor will stay at the same Key as it was before. If the cursor was inside the region of cut (moved) Keys, the cursor will be set to the Key before the `cutpoint`.
If you use ksCut() on a KeySet you got from kdbGet() and plan to use kdbSet() later, make sure that you keep all Keys that should not be removed permanently. You have to keep the KeySet that was returned and the KeySet `ks`.

**Example:**

You have the keyset `ks`:

- `system:/mountpoint/interest`
- `system:/mountpoint/interest/folder`
- `system:/mountpoint/interest/folder/key1`
- `system:/mountpoint/interest/folder/key2`
- `system:/mountpoint/other/key1`

When you use

```
Key * parentKey = keyNew ("system:/mountpoint/interest", KEY_END);
KDB * kdb = kdbOpen (NULL, parentKey);
KeySet * ks = ksNew (0, KS_END);
kdbGet (kdb, ks, parentKey);
KeySet * returned = ksCut (ks, parentKey);
kdbSet (kdb, ks, parentKey); // all keys below cutpoint are now removed
kdbClose (kdb, parentKey);
```

Then in `returned` are:

- `system:/mountpoint/interest`
- `system:/mountpoint/interest/folder`
- `system:/mountpoint/interest/folder/key1`
- `system:/mountpoint/interest/folder/key2`

And in `ks` are:

- `system:/mountpoint/other/key1`

So kdbSet() permanently removes all keys at or below `system:/mountpoint/interest`.

**Parameters**

| | |
|---|---|
| *ks* | the Keyset to cut. It will be modified by removing all Keys at or below the cutpoint. |
| *cutpoint* | the point where to cut out the Keyset |

**Returns**

a new allocated KeySet which needs to be deleted with ksDel(). The KeySet consists of all Keys (of the original KeySet ks) below the cutpoint. If the Key cutpoint exists, it will also be appended.

**Return values**

| | |
|---|---|
| *0* | on NULL pointers, no Key name or allocation problems |

**Since**

1.0.0

**See also**

kdbGet() for an explanation on why you might get more Keys than you requested.

### 572.36.3.7 cut() [2/2]

```
KeySet kdb::KeySet::cut (
            std::string const & name )  [inline]
```

Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.
Searches for the `cutpoint` inside the KeySet `ks`. If found, it cuts out this Key and everything which is below (see keyIsBelow()) this Key. These Keys will be missing in the keyset `ks`. Instead, they will be moved to the returned KeySet. If `cutpoint` is not found an empty KeySet is returned and `ks` is not changed.
The cursor will stay at the same Key as it was before. If the cursor was inside the region of cut (moved) Keys, the cursor will be set to the Key before the `cutpoint`.
If you use ksCut() on a KeySet you got from kdbGet() and plan to use kdbSet() later, make sure that you keep all Keys that should not be removed permanently. You have to keep the KeySet that was returned and the KeySet `ks`.

**Example:**

You have the keyset `ks`:

- `system:/mountpoint/interest`
- `system:/mountpoint/interest/folder`
- `system:/mountpoint/interest/folder/key1`
- `system:/mountpoint/interest/folder/key2`
- `system:/mountpoint/other/key1`

When you use

```
Key * parentKey = keyNew ("system:/mountpoint/interest", KEY_END);
KDB * kdb = kdbOpen (NULL, parentKey);
KeySet * ks = ksNew (0, KS_END);
kdbGet (kdb, ks, parentKey);
KeySet * returned = ksCut (ks, parentKey);
kdbSet (kdb, ks, parentKey); // all keys below cutpoint are now removed
kdbClose (kdb, parentKey);
```

Then in `returned` are:

- `system:/mountpoint/interest`
- `system:/mountpoint/interest/folder`
- `system:/mountpoint/interest/folder/key1`
- `system:/mountpoint/interest/folder/key2`

And in `ks` are:

- `system:/mountpoint/other/key1`

So kdbSet() permanently removes all keys at or below `system:/mountpoint/interest`.

**Parameters**

| | |
|---|---|
| *ks* | the Keyset to cut. It will be modified by removing all Keys at or below the cutpoint. |
| *cutpoint* | the point where to cut out the Keyset |

**Returns**

a new allocated KeySet which needs to be deleted with ksDel(). The KeySet consists of all Keys (of the original KeySet ks) below the cutpoint. If the Key cutpoint exists, it will also be appended.

**Return values**

| | |
|---|---|
| *0* | on NULL pointers, no Key name or allocation problems |

**Since**

1.0.0

**See also**

kdbGet() for an explanation on why you might get more Keys than you requested.

### 572.36.3.8 dup()

`KeySet kdb::KeySet::dup ( ) const [inline]`

Duplicate a keyset.

**Returns**

a copy of the keys

This is only a shallow copy. For a deep copy you need to dup every key.
Return a duplicate of a KeySet. Objects created with ksDup() must be destroyed with ksDel().
Memory will be allocated as needed for dynamic properties, so you need to ksDel() the returned pointer.
A flat copy is made, so the Keys will not be duplicated, but their reference counter is updated, so both KeySets need to be deleted via ksDel().

**Parameters**

| | |
|---|---|
| *source* | has to be an initialized KeySet |

**Returns**

a flat copy of source on success

**Return values**

| | |
|---|---|
| *0* | on NULL pointer |

**Since**

1.0.0

**See also**

ksNew() for creating a new KeySet

ksDel() for deleting a KeySet

keyDup() for Key duplication

#### 572.36.3.9 get()

```
template<typename T >
T kdb::KeySet::get (
            std::string const & name,
            const elektraLookupFlags options = KDB_O_NONE ) const  [inline]
```

Generic lookup+get for keysets.

**Parameters**

| | |
|---|---|
| *name* | the key name to get |
| *options* | the options to be passed to lookup() |

**Exceptions**

| | |
|---|---|
| *KeyNotFoundException* | if no key found |

**Note**

To specialize more complex types (which are generic themselves) you can also specialize KeySetType↩ Wrapper<T>.

Use
`#include <keysetget.hpp>`
to include specializations for std types.

**Returns**

the requested type

#### 572.36.3.10 getKeySet()

```
ckdb::KeySet * kdb::KeySet::getKeySet ( ) const  [inline]
```

Passes out the raw keyset pointer.
This function exists so that pure C functions that do not have a C++ binding can be called.

**Returns**

pointer to internal ckdb KeySet

#### 572.36.3.11 lookup() [1/2]

```
Key kdb::KeySet::lookup (
            const Key & key,
            const elektraLookupFlags options = KDB_O_NONE ) const  [inline]
```

Look for a Key contained in `ks` that matches the name of the `key`.

**Note**

Applications should only use ksLookup() with cascading Keys (Key name starting with `/`). Furthermore, a lookup should be done for every Key (also when iterating over Keys) so that the specifications are honored correctly. Keys of all namespaces need to be present so that ksLookup() can work correctly, so make sure to also use kdbGet() with a cascading Key.

ksLookup() is designed to let you work with a KeySet containing all Keys of the application. The idea is to fully kdbGet() the whole configuration of your application and process it all at once with many `ksLookup()`.
This function is efficient (at least using binary search). Together with kdbGet(), which you can use to load the whole configuration, you can write very effective and short code for configuration:

```
Key * key = keyNew ("/sw/tests/myapp/#0/current/",  KEY_END);
KDB * handle = kdbOpen (NULL, key);
kdbGet (handle, myConfig, key);
Key * result = ksLookupByName (myConfig, "/sw/tests/myapp/#0/current/testkey1", 0);
```

This is the way programs should get their configuration and search for the values. It is guaranteed, that more namespaces can be added easily and that all values can be set by admin and user. Furthermore, using the kdb-tool, it is possible to introspect which values an application will get (by doing the same cascading lookup).
If found, a pointer to the Key is returned. If not found a NULL pointer is returned.
Cascading lookups will by default search in all namespaces (proc:/, dir:/, user:/ and system:/), but will also correctly consider the specification (=metadata) in spec:/:

- `override/#` will make sure that another Key is considered before
- `namespace/#` will change the number and/or order in which the namespaces are searched
- `fallback/#` will search for other Keys when the other possibilities up to now were not successful
- `default` to return the given value when not even `fallback` Keys were found.

#### Note

override and fallback work recursively, while default does not.

This process is very flexible, but it would be boring to manually follow all this links to find out which Key will be taken in the end. Use `kdb get -v` to trace the Keys.

#### KDB_O_POP

When KDB_O_POP is set the Key which was found will be ksPop()ed.

#### Note

Like in ksPop() the popped Key always needs to be keyDel() afterwards, even if it is appended to another KeySet.

```
void f (KeySet * iterator, KeySet * lookup)
{
        KeySet * append = ksNew (ksGetSize (lookup), KS_END);
        ssize_t ksSize = ksGetSize (iterator);
        for (elektraCursor it = 0; it < ksSize; ++it)
        {
                Key * current = ksAtCursor (iterator, it);
                Key * key = ksLookup (lookup, current, KDB_O_POP);
                // do something...
                ksAppendKey (append, key); // now append it to append, not lookup!
                keyDel (key);                // make sure to ALWAYS delete poped keys.
        }
        ksAppend (lookup, append);
        // now lookup needs to be sorted only once, append never
        ksDel (append);
}
```

This is also a nice example how a complete application with ksLookup() can look like.

#### KDB_O_DEL

Passing KDB_O_DEL will cause the deletion of the parameter `key` using keyDel().

#### Hybrid search

When Elektra is compiled with `ENABLE_OPTIMIZATIONS=ON` a hybrid search decides dynamically between the binary search and the `OPMPHM`. The hybrid search can be overruled by passing KDB_O_OPMPHM or KDB_O_BINSEARCH in the options to ksLookup().

#### Parameters

| | |
|---|---|
| *ks* | the KeySet that should be searched |
| *key* | the Key object you are looking for |
| *options* | of type elektraLookupFlags with some `KDB_O_*` option bits - as explained above |

**Returns**

pointer to the Key found

**Return values**

| | |
|---|---|
| *0* | if no Key has been found |
| *0* | on NULL pointers |

**Since**

1.0.0

**See also**

ksLookupByName() to search by a name given by a string

ksGetSize(), ksAtCursor() for iterating over a KeySet

**Note**

That the internal key cursor will point to the found key

### 572.36.3.12 lookup() [2/2]

```
Key kdb::KeySet::lookup (
            std::string const & name,
            const elektraLookupFlags options = KDB_O_NONE ) const  [inline]
```

Lookup a key by name.

**Parameters**

| | |
|---|---|
| *name* | the name to look for |
| *options* | some options to pass |

**Returns**

the found key

**See also**

lookup (const Key &Key, const elektraLookupFlags options)

**Note**

That the internal key cursor will point to the found key

### 572.36.3.13 operator=()

```
KeySet & kdb::KeySet::operator= (
            KeySet const & other )  [inline]
```

Duplicate a keyset.
This keyset will be a duplicate of the other afterwards.

**Note**

that they still reference to the same Keys, so if you change key values also the keys in the original keyset will be changed.

#### 572.36.3.14 pop()

```
Key kdb::KeySet::pop ( )  [inline]
```

Remove and return the last Key of `ks`.
The reference counter of the Key will be decremented by one.
The KeySet's cursor will not be affected if it did not point to the popped Key.

**Note**

You need to keyDel() the Key afterwards, if you don't append it to another KeySet. It has the same semantics like a Key allocated with keyNew() or keyDup().

```
ks1=ksNew(0, KS_END);
ks2=ksNew(0, KS_END);
k1=keyNew("user:/name", KEY_END); // ref counter 0
ksAppendKey(ks1, k1); // ref counter 1
ksAppendKey(ks2, k1); // ref counter 2
k1=ksPop (ks1); // ref counter 1
k1=ksPop (ks2); // ref counter 0, like after keyNew()
ksAppendKey(ks1, k1); // ref counter 1
ksDel (ks1); // key is deleted too
ksDel (ks2);
```

**Parameters**

| | |
|---|---|
| *ks* | KeySet to pop a Key from |

**Returns**

the last Key of `ks`

**Return values**

| | |
|---|---|
| *NULL* | if `ks` is empty or a NULL pointer |

**Since**

1.0.0

**See also**

ksLookup() to pop Keys by name

ksCopy() to pop all Keys

#### 572.36.3.15 search() [1/2]

```
ssize_t kdb::KeySet::search (
              const Key & toSearch ) const  [inline]
```

Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.

```
ssize_t result = ksSearch(ks, key);
if (result >= 0)
{
        ssize_t position = result;
        // The key already exist in key set.
} else {
        ssize_t insertpos = -result-1;
        // The key was not found in key set.
}
```

**Parameters**

| | |
|---|---|
| *ks* | the keyset to work with |
| *key* | the key to check |

**Returns**

position where the key is (>=0) if the key was found

-insertpos -1 (< 0) if the key was not found so to get the insertpos simple do: -insertpos -1

**See also**

ksLookup() for retrieving the found Key

#### 572.36.3.16 search() [2/2]

```
ssize_t kdb::KeySet::search (
            std::string const & name ) const  [inline]
```

Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.

```
ssize_t result = ksSearch(ks, key);
if (result >= 0)
{
        ssize_t position = result;
        // The key already exist in key set.
} else {
        ssize_t insertpos = -result-1;
        // The key was not found in key set.
}
```

**Parameters**

| | |
|---|---|
| *ks* | the keyset to work with |
| *key* | the key to check |

**Returns**

position where the key is (>=0) if the key was found

-insertpos -1 (< 0) if the key was not found so to get the insertpos simple do: -insertpos -1

**See also**

ksLookup() for retrieving the found Key

**Note**

Accepts a keyname as string instead of a Key object.

#### 572.36.3.17 size()

```
ssize_t kdb::KeySet::size ( ) const  [inline]
```

The size of the keyset.

**Returns**

the number of keys in the keyset

The documentation for this class was generated from the following file:

- keyset.hpp

## 572.37 org.libelektra.KeySet Class Reference

Java representation of a native Elektra key set, a container for keys.
Inherits AbstractSet< Key >, and NavigableSet< Key >.

## Public Member Functions

- KeySet dup ()

  *Duplicates the key set.*
- KeySet copy (KeySet source)

  *Copies key references from.*
- KeySet append (Key key)

  *Append key to key set.*
- KeySet append (KeySet source)

  *Appends keys from key set.*
- KeySet cut (Key cutpoint)

  *Creates new key set with help of a cut point.*
- Optional< Key > remove (ReadableKey key)

  *Removes the specified key from key set.*
- Optional< Key > remove (String find)

  *Removes the key with the specified name from key set.*
- Key remove (int cursor)

  *Returns key from key set and also removes it from the set.*
- Key at (int cursor)

  *Gets the key at the given cursor position.*
- int indexOf (Key key)
- Optional< Key > lookup (Key find)

  *Search for a key in the key set.*
- Optional< Key > lookup (String find)

  *Search for a key in the key set.*
- String toString ()

  *Iterates though all keys in this key set and appends their representation to the output.*
- int size ()
- boolean isEmpty ()
- boolean contains (@Nullable Object o)
- Key lower (Key key)
- Key floor (Key key)
- Key ceiling (Key key)
- Key higher (Key key)
- Key pollFirst ()
- Key pollLast ()
- Iterator< Key > iterator ()
- NavigableSet< Key > descendingSet ()
- Iterator< Key > descendingIterator ()
- NavigableSet< Key > subSet (Key fromElement, boolean fromInclusive, Key toElement, boolean toInclusive)
- NavigableSet< Key > headSet (Key toElement, boolean inclusive)
- NavigableSet< Key > tailSet (Key fromElement, boolean inclusive)
- Object[ ] toArray ()
- boolean add (Key e)
- boolean remove (Object o)
- boolean containsAll (Collection<?> c)
- boolean addAll (Collection<? extends Key > c)
- boolean retainAll (Collection<?> c)
- boolean removeAll (Collection<?> c)
- void clear ()

  *Removes all elements form this KeySet.*
- Comparator<? super Key > comparator ()
- SortedSet< Key > subSet (Key fromElement, Key toElement)

- SortedSet< Key > headSet (Key toElement)
- SortedSet< Key > tailSet (Key fromElement)
- Key first ()
- Key last ()

## Static Public Member Functions

- static KeySet create (Key... keys)

  *Constructs a new KeySet containing the specified keys*

  *Example: KeySet keySet = KeySet.create(Key.create("A"), Key.create("B"));.*
- static KeySet create (int allocationHint, Key... keys)

  *Constructs a new KeySet containing the specified keys*

  *Example: KeySet keySet = KeySet.create(10, Key.create("A"), Key.create("B"));.*
- static KeySet create ()

  *Constructs an empty KeySet with a default allocation hint of 16.*

## Protected Member Functions

- KeySet (long nativePointer)

  *Constructor associating a new KeySet instance with a native pointer in long format.*
- KeySet (long nativePointer, boolean suppressCleanUp)

  *Constructor associating a new KeySet instance with a native pointer in long format*

  *Suppressing clean-up has been introduced for usage of this binding as JNI plug-in and should normally not be used in any other case.*
- KeySet (Pointer pointer)

  *Constructor associating a new KeySet instance with a JNA pointer.*
- void release ()

  *Clean-up method to release key set reference by trying to free the native reference*

  *key sets, will get cleaned up by garbage collection as soon as they get phantom reachable.*
- Pointer getPointer ()

### 572.37.1 Detailed Description

Java representation of a native Elektra key set, a container for keys.

### 572.37.2 Constructor & Destructor Documentation

#### 572.37.2.1 KeySet() [1/3]

```
org.libelektra.KeySet.KeySet (
            long nativePointer )  [inline], [protected]
```

Constructor associating a new KeySet instance with a native pointer in long format.

**Parameters**

| | |
|---|---|
| *nativePointer* | Native pointer to key set in long format |

#### 572.37.2.2 KeySet() [2/3]

```
org.libelektra.KeySet.KeySet (
            long nativePointer,
            boolean suppressCleanUp )  [inline], [protected]
```

Constructor associating a new KeySet instance with a native pointer in long format

Suppressing clean-up has been introduced for usage of this binding as JNI plug-in and should normally not be used in any other case.

**Parameters**

| | |
|---|---|
| *nativePointer* | Native pointer to key set in long format |
| *suppressCleanUp* | True to suppress native reference clean-up as soon as this {} instance becomes phantom reachable, false otherwise |

#### 572.37.2.3 KeySet() [3/3]

```
org.libelektra.KeySet.KeySet (
            Pointer pointer )  [inline], [protected]
```

Constructor associating a new KeySet instance with a JNA pointer.

**Parameters**

| | |
|---|---|
| *pointer* | JNA Pointer to key set |

**Exceptions**

| | |
|---|---|
| *IllegalArgumentException* | if `pointer` is `null` |

### 572.37.3 Member Function Documentation

#### 572.37.3.1 add()

```
boolean org.libelektra.KeySet.add (
            Key e )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet or the specified `key` has already been released |
| *NullPointerException* | if `key` is `null` |
| *KeySetException* | if inserting the `key` failed because of allocation problems |

**See also**

append(Key)

### 572.37.3.2 addAll()

```
boolean org.libelektra.KeySet.addAll (
            Collection<?  extends Key > c )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet or any Key contained in the specified collection `c` has already been released |
| *NullPointerException* | if the specified collection is `null` or contains `null` elements |
| *KeySetException* | if inserting the `key` failed because of allocation problems |

### 572.37.3.3 append() [1/2]

```
KeySet org.libelektra.KeySet.append (
            Key key )  [inline]
```

Append key to key set.

**Parameters**

| | |
|---|---|
| *key* | Key to append |

**Returns**

This KeySet, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet or the specified `key` has already been released |
| *IllegalArgumentException* | if `key` is `null` |
| *KeySetException* | if appending the `key` failed because of allocation problems |

**See also**

add(Key)

#### 572.37.3.4 append() [2/2]

```
KeySet org.libelektra.KeySet.append (
            KeySet source )  [inline]
```

Appends keys from key set.

**Parameters**

| | |
|---|---|
| *source* | Source KeySet to append all of its keys |

**Returns**

This KeySet, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet or the specified `source` has already been released |
| *IllegalArgumentException* | if `source` is `null` |
| *KeySetException* | if appending the `source` failed because of allocation problems |

#### 572.37.3.5 at()

```
Key org.libelektra.KeySet.at (
            int cursor )  [inline]
```

Gets the key at the given cursor position.

**Parameters**

| | |
|---|---|
| *cursor* | Cursor position used to fetch key; starting from 0 |

**Returns**

Key found at specified cursor position

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *IndexOutOfBoundsException* | if position is out of bounds |

#### 572.37.3.6 ceiling()

```
Key org.libelektra.KeySet.ceiling (
            Key key )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.7 clear()

```
void org.libelektra.KeySet.clear ( )  [inline]
```

Removes all elements form this KeySet.

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.8 comparator()

```
Comparator<?  super Key> org.libelektra.KeySet.comparator ( )  [inline]
```

@implSpec Returns
`null`
because natural ordering of keys is used (ReadableKey implements Comparable)

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.9 contains()

```
boolean org.libelektra.KeySet.contains (
            @Nullable Object o )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released or the passed o is a Key that has already been released |
| *NullPointerException* | if the specified element `o` is `null` |

**See also**

lookup(Key)

#### 572.37.3.10 containsAll()

```
boolean org.libelektra.KeySet.containsAll (
            Collection<?> c )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

**Exceptions**

| | |
|---|---|
| *NullPointerException* | if the specified collection is `null` or contains `null` elements |

#### 572.37.3.11 copy()

```
KeySet org.libelektra.KeySet.copy (
            KeySet source )  [inline]
```

Copies key references from.
`source`
to **this** KeySet

**Parameters**

| | |
|---|---|
| *source* | Key set that is used as source |

**Returns**

This KeySet, enabling a fluent interface

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet or the specified `source` has already been released |
| *IllegalArgumentException* | if `source` is `null` |

#### 572.37.3.12 create() [1/3]

```
static KeySet org.libelektra.KeySet.create ( )  [inline], [static]
```

Constructs an empty KeySet with a default allocation hint of 16.

**Returns**

Newly allocated key set

**Exceptions**

| | |
|---|---|
| *KeySetException* | on allocation problems |

#### 572.37.3.13 create() [2/3]

```
static KeySet org.libelektra.KeySet.create (
            int allocationHint,
            Key...  keys )  [inline], [static]
```

Constructs a new KeySet containing the specified keys

Example: KeySet keySet = KeySet.create(10, Key.create("A"), Key.create("B"));.

**Parameters**

| | |
|---|---|
| *allocationHint* | Hint indicating the expected size of the key set |
| *keys* | List of initial keys for the key set |

**Returns**

New key set containing the specified initial keys

**Exceptions**

| | |
|---|---|
| *KeySetException* | on allocation problems |

#### 572.37.3.14 create() [3/3]

```
static KeySet org.libelektra.KeySet.create (
            Key...  keys )  [inline], [static]
```

Constructs a new KeySet containing the specified keys

Example: KeySet keySet = KeySet.create(Key.create("A"), Key.create("B"));.

**Parameters**

| | |
|---|---|
| *keys* | List of initial keys for the key set |

**Returns**

New key set containing the specified initial keys

**Exceptions**

| | |
|---|---|
| *KeySetException* | on allocation problems |

#### 572.37.3.15 cut()

```
KeySet org.libelektra.KeySet.cut (
            Key cutpoint )  [inline]
```

Creates new key set with help of a cut point.

**Parameters**

| | |
|---|---|
| *cutpoint* | Key that is used as cutting point |

**Returns**

New KeySet containing all keys until the cutting point

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *IllegalArgumentException* | if `cutpoint` is `null` |
| *KeySetException* | if `cutpoint` is missing a key name or on allocation problems |

#### 572.37.3.16 descendingIterator()

```
Iterator<Key> org.libelektra.KeySet.descendingIterator ( )  [inline]
```

**Returns**

New DescendingKeySetIterator backed by this KeySet

#### 572.37.3.17 descendingSet()

```
NavigableSet<Key> org.libelektra.KeySet.descendingSet ( )  [inline]
```

**Returns**

New DescendingKeySetView backed by this KeySet

#### 572.37.3.18 dup()

```
KeySet org.libelektra.KeySet.dup ( )  [inline]
```

Duplicates the key set.

**Returns**

New KeySet containing the same key references as this KeySet does

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.19 first()

```
Key org.libelektra.KeySet.first ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NoSuchElementException* | |

#### 572.37.3.20 floor()

```
Key org.libelektra.KeySet.floor (
            Key key )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.21 getPointer()

```
Pointer org.libelektra.KeySet.getPointer ( )  [inline], [protected]
```

**Returns**

JNA pointer to the native pointer for this key set

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.22 headSet() [1/2]

```
SortedSet<Key> org.libelektra.KeySet.headSet (
            Key toElement )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NullPointerException* | if `toElement` is `null` |
| *IllegalArgumentException* | |

#### 572.37.3.23 headSet() [2/2]

```
NavigableSet<Key> org.libelektra.KeySet.headSet (
            Key toElement,
            boolean inclusive )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NullPointerException* | if `toElement` is `null` |
| *IllegalArgumentException* | |

#### 572.37.3.24 higher()

```
Key org.libelektra.KeySet.higher (
            Key key )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.25 indexOf()

```
int org.libelektra.KeySet.indexOf (
            Key key )  [inline]
```

**Parameters**

| | |
|---|---|
| *key* | Key to look for |

**Returns**

Index of the
`key`
in this KeySet

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if KeySet or `key` has already been released |
| *IllegalArgumentException* | if `key` was not found in this KeySet |

#### 572.37.3.26 isEmpty()

```
boolean org.libelektra.KeySet.isEmpty ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.27 iterator()

```
Iterator<Key> org.libelektra.KeySet.iterator ( )  [inline]
```

**Returns**

New KeySetIterator backed by this KeySet

#### 572.37.3.28 last()

```
Key org.libelektra.KeySet.last ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NoSuchElementException* | |

### 572.37.3.29 lookup() [1/2]

```
Optional<Key> org.libelektra.KeySet.lookup (
            Key find )  [inline]
```

Search for a key in the key set.

**Parameters**

| | |
|---|---|
| *find* | Key used in search |

**Returns**

Key if search successful, Optional#empty() otherwise

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *IllegalArgumentException* | if `key` is `null` |

**See also**

#contains(Object)

### 572.37.3.30 lookup() [2/2]

```
Optional<Key> org.libelektra.KeySet.lookup (
            String find )  [inline]
```

Search for a key in the key set.

**Parameters**

| | |
|---|---|
| *find* | Key name used in search |

**Returns**

Key if search successful, Optional#empty() otherwise

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *IllegalArgumentException* | if `find` is blank |

#### 572.37.3.31 lower()

```
Key org.libelektra.KeySet.lower (
            Key key )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.32 pollFirst()

```
Key org.libelektra.KeySet.pollFirst ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.33 pollLast()

```
Key org.libelektra.KeySet.pollLast ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.34 remove() [1/4]

```
Key org.libelektra.KeySet.remove (
            int cursor )  [inline]
```

Returns key from key set and also removes it from the set.

**Parameters**

| | |
|---|---|
| *cursor* | Cursor position of the key to remove; starting from 0 |

**Returns**

Key found at given cursor position

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *IndexOutOfBoundsException* | if position is out of bounds |

#### 572.37.3.35 remove() [2/4]

```
boolean org.libelektra.KeySet.remove (
            Object o )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

**See also**

remove(ReadableKey)

#### 572.37.3.36 remove() [3/4]

```
Optional<Key> org.libelektra.KeySet.remove (
            ReadableKey key )  [inline]
```

Removes the specified key from key set.

**Parameters**

| | |
|---|---|
| *key* | Key to remove |

**Returns**

Removed Key from the key set, matching the specified `key`'s name. May or may not reference the same native key resource. Optional#empty() if the specified `key` was not found.

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if KeySet or `key` has already been released |
| *IllegalArgumentException* | if `key` is `null` |

**See also**

remove(Object)

#### 572.37.3.37 remove() [4/4]

```
Optional<Key> org.libelektra.KeySet.remove (
            String find )  [inline]
```

Removes the key with the specified name from key set.

**Parameters**

| | |
|---|---|
| *find* | Name of the key to remove |

**Returns**

Removed Key from the key set, matching the specified
`key`

's name. {} if the no key matching the specified name was not found. IllegalStateException if this KeySet} has already been rele

### 572.37.3.38 removeAll()

```
boolean org.libelektra.KeySet.removeAll (
            Collection<?> c )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NullPointerException* | if the specified collection is `null` or contains `null` elements |

### 572.37.3.39 retainAll()

```
boolean org.libelektra.KeySet.retainAll (
            Collection<?> c )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NullPointerException* | if the specified collection is `null` or contains `null` elements |

### 572.37.3.40 size()

```
int org.libelektra.KeySet.size ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

### 572.37.3.41 subSet() [1/2]

```
NavigableSet<Key> org.libelektra.KeySet.subSet (
            Key fromElement,
            boolean fromInclusive,
            Key toElement,
            boolean toInclusive )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

**Exceptions**

| | |
|---|---|
| *NullPointerException* | if `fromElement` or `toElement` is `null` |
| *IllegalArgumentException* | |

### 572.37.3.42 subSet() [2/2]

```
SortedSet<Key> org.libelektra.KeySet.subSet (
            Key fromElement,
            Key toElement ) [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NullPointerException* | if `fromElement` or `toElement` is `null` |
| *IllegalArgumentException* | |

### 572.37.3.43 tailSet() [1/2]

```
SortedSet<Key> org.libelektra.KeySet.tailSet (
            Key fromElement ) [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |
| *NullPointerException* | if `fromElement` is `null` |
| *IllegalArgumentException* | |

### 572.37.3.44 tailSet() [2/2]

```
NavigableSet<Key> org.libelektra.KeySet.tailSet (
            Key fromElement,
            boolean inclusive ) [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

**Exceptions**

| | |
|---|---|
| *NullPointerException* | if `fromElement` is `null` |
| *IllegalArgumentException* | |

#### 572.37.3.45 toArray()

```
Object [] org.libelektra.KeySet.toArray ( )  [inline]
```

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this KeySet has already been released |

#### 572.37.3.46 toString()

```
String org.libelektra.KeySet.toString ( )  [inline]
```

Iterates though all keys in this key set and appends their representation to the output.
Uses the toString() function of the Key objects.

**Returns**

Represents this KeySet as string

The documentation for this class was generated from the following file:

- KeySet.java

## 572.38 org.libelektra.exception.KeySetException Class Reference

Indicates a generic exception while calling one of KeySet's methods

This exception is related to unrecoverable C API specific errors (primarily allocation problems).
Inherits RuntimeException.

### 572.38.1 Detailed Description

Indicates a generic exception while calling one of KeySet's methods

This exception is related to unrecoverable C API specific errors (primarily allocation problems).
All other exceptional states are represented by more specific exceptions. For more detailed information see exceptions in this package as well as a method's individual documentation.
The documentation for this class was generated from the following file:

- KeySetException.java

## 572.39 kdb::KeySetIterator Class Reference

For C++ forward Iteration over KeySets.
`#include <keyset.hpp>`

### 572.39.1 Detailed Description

For C++ forward Iteration over KeySets.
(External Iterator)

```
for (Key k:ks3)
{
   std::cout « k.getName() « std::endl;
}
```

The documentation for this class was generated from the following file:

- keyset.hpp

## 572.40 kdb::KeySetReverseIterator Class Reference

For C++ reverse Iteration over KeySets.
`#include <keyset.hpp>`

### 572.40.1 Detailed Description

For C++ reverse Iteration over KeySets.
(External Iterator)
The documentation for this class was generated from the following file:

- keyset.hpp

## 572.41 org.libelektra.exception.KeyStringValueException Class Reference

Indicates a key's underlying native key value is not of type string, and therefore is not compatible with the invoked functionality raising this exception.
Inherits IllegalStateException.

### 572.41.1 Detailed Description

Indicates a key's underlying native key value is not of type string, and therefore is not compatible with the invoked functionality raising this exception.
The documentation for this class was generated from the following file:

- KeyStringValueException.java

## 572.42 kdb::Layer Class Reference

Base class for all layers.
`#include <kdbvalue.hpp>`
Inherited by kdb::KeyValueLayer, and kdb::WrapLayer.

### 572.42.1 Detailed Description

Base class for all layers.
The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.43 kdb::LockPolicyIs< Policy > Class Template Reference

Needed by the user to set one of the policies.
`#include <kdbvalue.hpp>`
Inherits kdb::DefaultPolicies.

### 572.43.1 Detailed Description

**template<typename Policy>**
**class kdb::LockPolicyIs< Policy >**

Needed by the user to set one of the policies.

**Template Parameters**

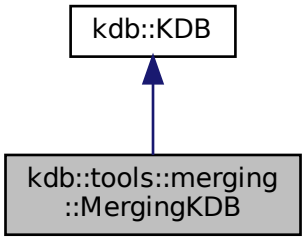
| | |
|---|---|
| *Policy* | |

The documentation for this class was generated from the following file:

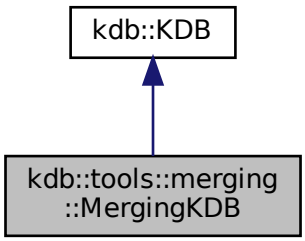
- kdbvalue.hpp

## 572.44 kdb::tools::merging::MergingKDB Class Reference

Provides a merging wrapper around a KDB instance.
`#include <mergingkdb.hpp>`
Inheritance diagram for kdb::tools::merging::MergingKDB:




Collaboration diagram for kdb::tools::merging::MergingKDB:




### Public Member Functions

- int get (KeySet &returned, std::string const &keyname) override

*Behaves like the KDB get function.*

- int get (KeySet &returned, Key &parentKey) override

  *Behaves like the KDB get function.*

- virtual int synchronize (KeySet &returned, std::string const &keyname, ThreeWayMerge &merger)

  *Synchronizes the file with the supplied KeySet.*

- virtual int synchronize (KeySet &returned, Key &parentKey, ThreeWayMerge &merger)

  *If a conflict occurs during set, the supplied merger is used to resolve the conflict.*

## 572.44.1 Detailed Description

Provides a merging wrapper around a KDB instance.
The wrapper allows to pass a three way merger instance that is used to resolve conflicts during KDB set.

## 572.44.2 Member Function Documentation

### 572.44.2.1 get() [1/2]

```
int kdb::tools::merging::MergingKDB::get (
            KeySet & returned,
            Key & parentKey )  [override], [virtual]
```

Behaves like the KDB get function.

**See also**

KDB

Reimplemented from kdb::KDB.

### 572.44.2.2 get() [2/2]

```
int kdb::tools::merging::MergingKDB::get (
            KeySet & returned,
            std::string const & keyname )  [override], [virtual]
```

Behaves like the KDB get function.

**See also**

KDB

Reimplemented from kdb::KDB.

### 572.44.2.3 synchronize() [1/2]

```
int kdb::tools::merging::MergingKDB::synchronize (
            KeySet & returned,
            Key & parentKey,
            ThreeWayMerge & merger )  [virtual]
```

If a conflict occurs during set, the supplied merger is used to resolve the conflict.
If the conflict cannot be solved, an exception is thrown. If the KeySet was successfully written (either by merging or due the absence of a conflict) the supplied KeySet is updated with the new content of the file.

**See also**

KDB

**Exceptions**

| *MergingKDBException* | |
|---|---|

#### 572.44.2.4 synchronize() [2/2]

```
int kdb::tools::merging::MergingKDB::synchronize (
            KeySet & returned,
            std::string const & keyname,
            ThreeWayMerge & merger )  [virtual]
```

Synchronizes the file with the supplied KeySet.
If a conflict occurs during set, the supplied merger is used to resolve the conflict. If the conflict cannot be solved, an exception is thrown. If the KeySet was successfully written (either by merging or due the absence of a conflict) the supplied KeySet is updated with the new content of the file.

**See also**

KDB

**Exceptions**

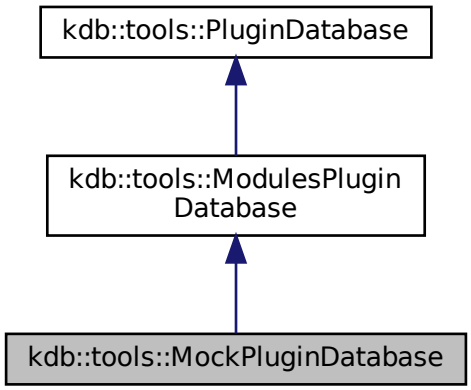
| *MergingKDBException* | |
|---|---|

The documentation for this class was generated from the following files:

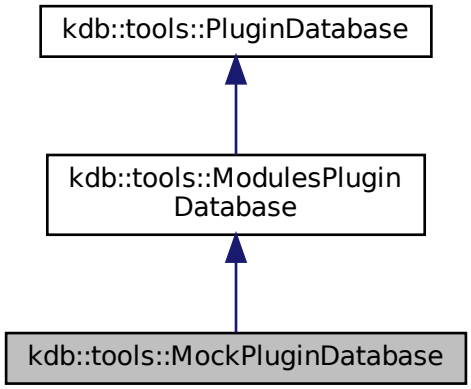
- mergingkdb.hpp
- mergingkdb.cpp

## 572.45 kdb::tools::MockPluginDatabase Class Reference

A plugin database that works with added fake data.
`#include <plugindatabase.hpp>`
Inheritance diagram for kdb::tools::MockPluginDatabase:



Collaboration diagram for kdb::tools::MockPluginDatabase:




## Public Member Functions

- std::vector< std::string > listAllPlugins () const

  *list all plugins*
- std::string lookupInfo (PluginSpec const &spec, std::string const &which) const

  *lookup contract clauses or dynamic information*
- func_t getSymbol (PluginSpec const &whichplugin, std::string const &which) const

  *get exported plugin symbol*

## Public Attributes

- std::unordered_map< PluginSpec, std::unordered_map< std::string, std::string >, PluginSpecHash, PluginSpecName > data

  *only data from here will be returned*

## Additional Inherited Members

### 572.45.1 Detailed Description

A plugin database that works with added fake data.

### 572.45.2 Member Function Documentation

#### 572.45.2.1 getSymbol()

```
PluginDatabase::func_t kdb::tools::MockPluginDatabase::getSymbol (
            PluginSpec const & whichplugin,
            std::string const & which ) const  [virtual]
```

get exported plugin symbol

**Parameters**

| | |
|---|---|
| *whichplugin* | from which plugin? |
| *which* | which symbol would you like to look up? |

**Returns**

the function pointer to the exported symbol or NULL if the symbol was not found

Reimplemented from kdb::tools::ModulesPluginDatabase.

#### 572.45.2.2 listAllPlugins()

```
std::vector< std::string > kdb::tools::MockPluginDatabase::listAllPlugins ( ) const  [virtual]
```

list all plugins
If Elektra is compiled with plugins, it will search for shared libraries. In any case, if no shared libraries were found it will fallback to an internal list (plugins that were compiled together with Elektra).

**Returns**

a list of all available plugins

Reimplemented from kdb::tools::ModulesPluginDatabase.

#### 572.45.2.3 lookupInfo()

```
std::string kdb::tools::MockPluginDatabase::lookupInfo (
            PluginSpec const & whichplugin,
            std::string const & which ) const  [virtual]
```

lookup contract clauses or dynamic information

**Parameters**

| | |
|---|---|
| *whichplugin* | about which plugin? |
| *which* | about which clause in the contract? |

**Returns**

the clause of the contract

Reimplemented from kdb::tools::ModulesPluginDatabase.

### 572.45.3 Member Data Documentation

#### 572.45.3.1 data

```
std::unordered_map<PluginSpec, std::unordered_map<std::string, std::string>, PluginSpecHash,
PluginSpecName> kdb::tools::MockPluginDatabase::data  [mutable]
```

only data from here will be returned

**Note**

that it is ordered by name, i.e., different ref-names cannot be distinguished

The documentation for this class was generated from the following files:

- plugindatabase.hpp
- plugindatabase.cpp

## 572.46 kdb::tools::Modules Class Reference

Allows one to load plugins.
`#include <modules.hpp>`

### Public Member Functions

- PluginPtr load (std::string const &pluginName)
- PluginPtr load (std::string const &pluginName, kdb::KeySet const &config)
- PluginPtr load (PluginSpec const &spec)

### 572.46.1 Detailed Description

Allows one to load plugins.

### 572.46.2 Member Function Documentation

#### 572.46.2.1 load() [1/3]

```
PluginPtr kdb::tools::Modules::load (
            PluginSpec const & spec )
```

**Returns**

a newly created plugin

#### 572.46.2.2 load() [2/3]

```
PluginPtr kdb::tools::Modules::load (
            std::string const & pluginName )
```

**Deprecated** do not use

#### 572.46.2.3 load() [3/3]

```
PluginPtr kdb::tools::Modules::load (
            std::string const & pluginName,
            kdb::KeySet const & config )
```

**Deprecated** do not use

The documentation for this class was generated from the following files:

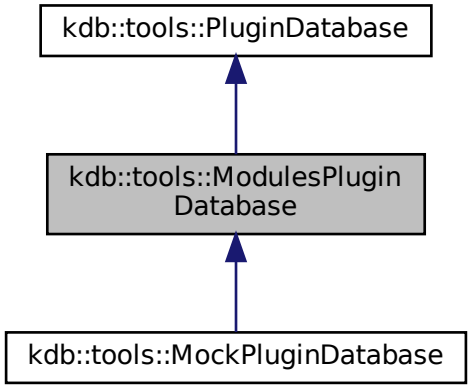
- modules.hpp
- modules.cpp

## 572.47 kdb::tools::ModulesPluginDatabase Class Reference

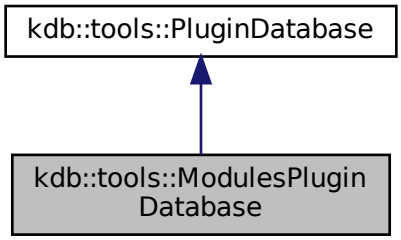
A plugin database that works with installed modules.

```
#include <plugindatabase.hpp>
```

Inheritance diagram for kdb::tools::ModulesPluginDatabase:




Collaboration diagram for kdb::tools::ModulesPluginDatabase:




## Public Member Functions

- std::vector< std::string > listAllPlugins () const

  *list all plugins*
- std::string lookupInfo (PluginSpec const &spec, std::string const &which) const

  *lookup contract clauses or dynamic information*
- func_t getSymbol (PluginSpec const &whichplugin, std::string const &which) const

  *get exported plugin symbol*
- PluginSpec lookupMetadata (std::string const &which) const

  *lookup which plugin handles metadata*
- PluginSpec lookupProvides (std::string const &provides) const

  *lookup which plugin is a provider for that plugin*
- std::map< int, PluginSpec > lookupAllProvidesWithStatus (std::string const &provides) const

  *looks up all plugins which are a suitable provider*
- std::vector< PluginSpec > lookupAllProvides (std::string const &provides) const

  *looks up all plugins which are a suitable provider*

## Additional Inherited Members

### 572.47.1 Detailed Description

A plugin database that works with installed modules.

### 572.47.2 Member Function Documentation

#### 572.47.2.1 getSymbol()

```
PluginDatabase::func_t kdb::tools::ModulesPluginDatabase::getSymbol (
            PluginSpec const & whichplugin,
            std::string const & which ) const  [virtual]
```

get exported plugin symbol

**Parameters**

| | |
|---|---|
| *whichplugin* | from which plugin? |
| *which* | which symbol would you like to look up? |

**Returns**

the function pointer to the exported symbol or NULL if the symbol was not found

Implements kdb::tools::PluginDatabase.
Reimplemented in kdb::tools::MockPluginDatabase.

#### 572.47.2.2 listAllPlugins()

```
std::vector< std::string > kdb::tools::ModulesPluginDatabase::listAllPlugins ( ) const  [virtual]
```

list all plugins
If Elektra is compiled with plugins, it will search for shared libraries. In any case, if no shared libraries were found it will fallback to an internal list (plugins that were compiled together with Elektra).

**Returns**

a list of all available plugins

Implements kdb::tools::PluginDatabase.
Reimplemented in kdb::tools::MockPluginDatabase.

#### 572.47.2.3 lookupAllProvides()

```
std::vector< PluginSpec > kdb::tools::ModulesPluginDatabase::lookupAllProvides (
            std::string const & provides ) const  [virtual]
```

looks up all plugins which are a suitable provider

**Note**

in case a plugin name is provided, the plugin with the name will also be part of the result. But if there are other plugins providing the requirement, then they will also be part of the result. The ordering of the resulting vector has no special meaning.

**Parameters**

| | |
|---|---|
| *provides* | is the provider to find |

**Returns**

a vector of plugins offering the requirement or are named after it

Implements kdb::tools::PluginDatabase.

#### 572.47.2.4 lookupAllProvidesWithStatus()

```
std::map< int, PluginSpec > kdb::tools::ModulesPluginDatabase::lookupAllProvidesWithStatus (
            std::string const & provides ) const  [virtual]
```

looks up all plugins which are a suitable provider

**Note**

in case a plugin name is provided, the plugin with the name will also be part of the result. But if there are other plugins providing the requirement, then they will also be part of the result.

**Parameters**

| | |
|---|---|
| *provides* | is the provider to find |

**Returns**

a map of plugins with their status offering the requirement or are named after it

Implements kdb::tools::PluginDatabase.

#### 572.47.2.5 lookupInfo()

```
std::string kdb::tools::ModulesPluginDatabase::lookupInfo (
            PluginSpec const & whichplugin,
            std::string const & which ) const  [virtual]
```

lookup contract clauses or dynamic information

**Parameters**

| | |
|---|---|
| *whichplugin* | about which plugin? |
| *which* | about which clause in the contract? |

**Returns**

the clause of the contract

Implements kdb::tools::PluginDatabase.
Reimplemented in kdb::tools::MockPluginDatabase.

#### 572.47.2.6 lookupMetadata()

```
PluginSpec kdb::tools::ModulesPluginDatabase::lookupMetadata (
            std::string const & which ) const  [virtual]
```

lookup which plugin handles metadata

**Parameters**

| | |
|---|---|
| *which* | the metadata of interest |

**Returns**

the best suited plugin specification which provides it

Implements kdb::tools::PluginDatabase.

#### 572.47.2.7 lookupProvides()

```
PluginSpec kdb::tools::ModulesPluginDatabase::lookupProvides (
            std::string const & provides ) const  [virtual]
```

lookup which plugin is a provider for that plugin

**Note**

will return a PluginSpec with getName() == provides if the string provides actually is a plugin name.

**Parameters**

| | |
|---|---|
| *provides* | is the provider to find |

**Exceptions**

| | |
|---|---|
| *NoPlugin* | if no plugin that provides the functionality could be found |

**Returns**

the plugin itself or the best suited plugin specification which provides it

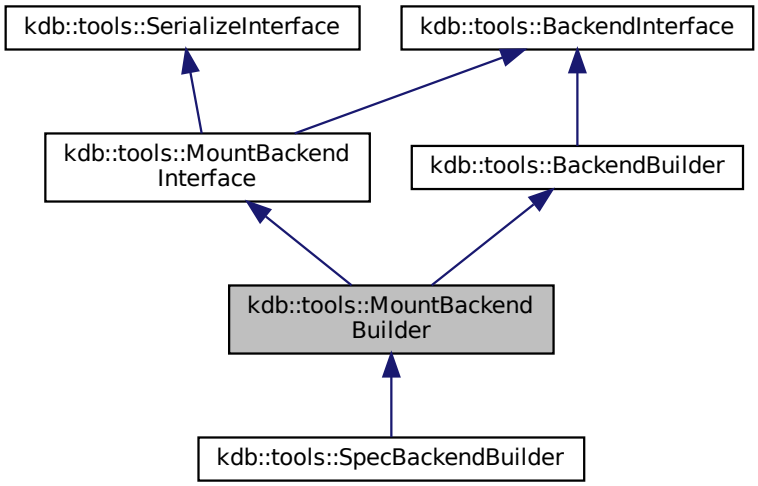
Implements kdb::tools::PluginDatabase.
The documentation for this class was generated from the following files:

- plugindatabase.hpp
- plugindatabase.cpp

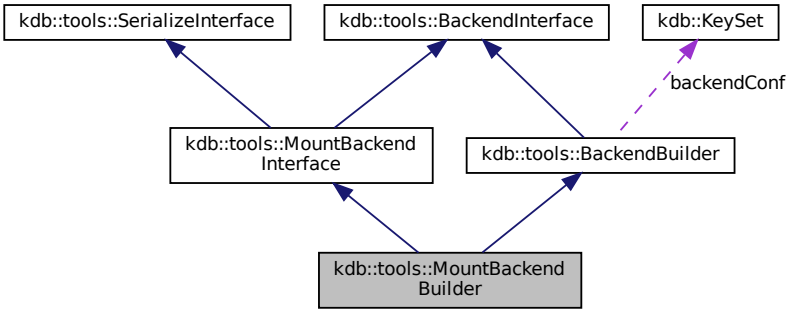
## 572.48 kdb::tools::MountBackendBuilder Class Reference

High-level functionality to build a mountpoint.
`#include <backendbuilder.hpp>`

Inheritance diagram for kdb::tools::MountBackendBuilder:




Collaboration diagram for kdb::tools::MountBackendBuilder:




## Public Member Functions

- void addPlugin (PluginSpec const &spec)

  *Add a plugin.*

### 572.48.1 Detailed Description

High-level functionality to build a mountpoint.
will enforce resolver and storage to be present

### 572.48.2 Member Function Documentation

#### 572.48.2.1 addPlugin()

```
void kdb::tools::MountBackendBuilder::addPlugin (
            PluginSpec const & plugin )  [inline], [virtual]
```

Add a plugin.

**Precondition**

Needs to be a unique new name (use refname if you want to add the same module multiple times)

Will automatically resolve virtual plugins to actual plugins.
Also calls the checkconf function if provided by the plugin. The checkconf function has the following signature: int checkconf (Key * errorKey, KeySet * config) and allows a plugin to verify its configuration at mount time.

**See also**

resolveNeeds()

**Parameters**

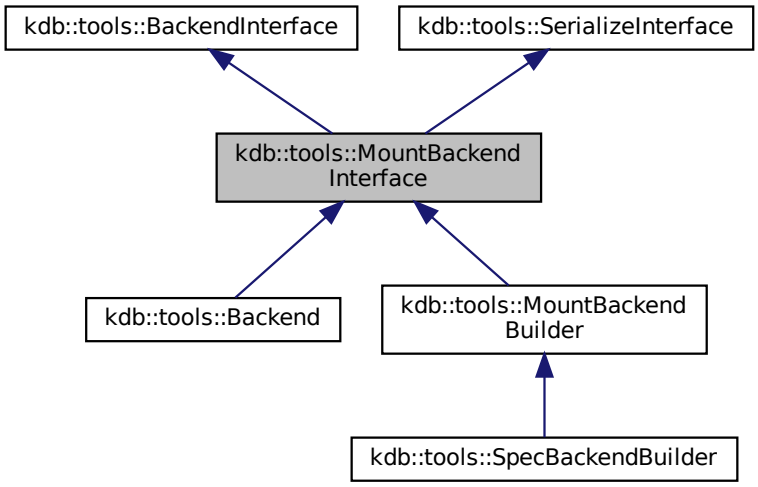
| | |
|---|---|
| *plugin* | |

Reimplemented from kdb::tools::BackendBuilder.
The documentation for this class was generated from the following files:

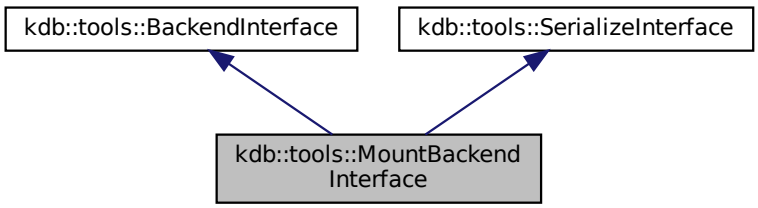
- backendbuilder.hpp
- backendbuilder.cpp

## 572.49 kdb::tools::MountBackendInterface Class Reference

Interface to work with mountpoints (backends) for factory.
`#include <backend.hpp>`
Inheritance diagram for kdb::tools::MountBackendInterface:



Collaboration diagram for kdb::tools::MountBackendInterface:



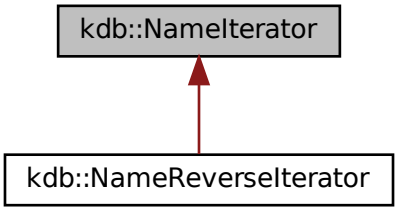

### 572.49.1 Detailed Description

Interface to work with mountpoints (backends) for factory.
The documentation for this class was generated from the following files:

- backend.hpp
- src/backend.cpp

## 572.50 kdb::NameIterator Class Reference

For C++ forward Iteration over Names.
`#include <key.hpp>`
Inheritance diagram for kdb::NameIterator:




### 572.50.1 Detailed Description

For C++ forward Iteration over Names.
(External Iterator)

```
for (std::string s:k3)
{
   std::cout « s « std::endl;
}
```

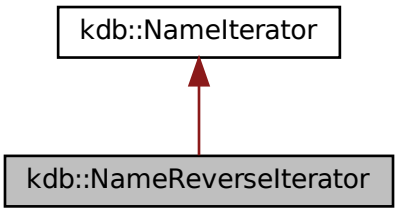
The documentation for this class was generated from the following file:

- key.hpp

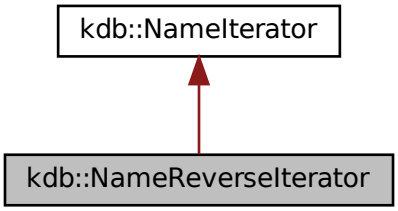
## 572.51 kdb::NameReverseIterator Class Reference

For C++ reverse Iteration over Names.

`#include <key.hpp>`
Inheritance diagram for kdb::NameReverseIterator:




Collaboration diagram for kdb::NameReverseIterator:




### 572.51.1 Detailed Description

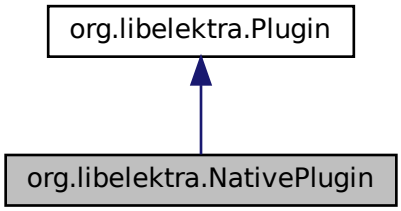
For C++ reverse Iteration over Names.
(External Iterator)
The documentation for this class was generated from the following file:

- key.hpp

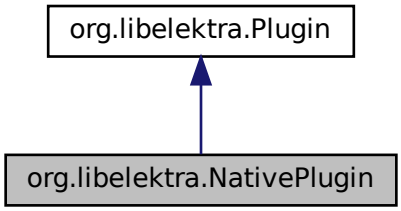
## 572.52 org.libelektra.NativePlugin Class Reference

This class can be used to load native Elektra plugins to be used by Java directly.

Inheritance diagram for org.libelektra.NativePlugin:




Collaboration diagram for org.libelektra.NativePlugin:




## Public Member Functions

- NativePlugin (String pluginName, Key errorKey, KeySet modules) throws InstallationException

  *Constructor for loading an Elektra plugin.*
- NativePlugin (String pluginName, KeySet modules, KeySet config, Key errorKey)

  *Constructor for loading an Elektra plugin.*
- KeySet getConfig ()

  *Gets the config which was used to configure the plugin.*
- int open (KeySet conf, Key errorKey)

  *Calls the plugin's open function.*
- int kdbOpen (Key errorKey)

  *Opens the session with the KeyDatabase.*
- int close (Key errorKey)

  *Closes the session with the Key database.*
- int set (KeySet keySet, Key errorKey) throws KDBException

  *Lets the plugin transform the given KeySet.*

## Additional Inherited Members

### 572.52.1 Detailed Description

This class can be used to load native Elektra plugins to be used by Java directly.

## 572.52.2 Constructor & Destructor Documentation

### 572.52.2.1 NativePlugin() [1/2]

```
org.libelektra.NativePlugin.NativePlugin (
            String pluginName,
            Key errorKey,
            KeySet modules ) throws InstallationException  [inline]
```

Constructor for loading an Elektra plugin.

**Parameters**

| | |
|---|---|
| *pluginName* | The plugin name |
| *errorKey* | The errorKey |
| *modules* | TODO #3754 add parameter description |

**Exceptions**

| | |
|---|---|
| *InstallationException* | if the plugin does not exist |
| *IllegalStateException* | if `modules` or `errorKey` has already been released |

### 572.52.2.2 NativePlugin() [2/2]

```
org.libelektra.NativePlugin.NativePlugin (
            String pluginName,
            KeySet modules,
            KeySet config,
            Key errorKey )  [inline]
```

Constructor for loading an Elektra plugin.

**Parameters**

| | |
|---|---|
| *pluginName* | The plugin name |
| *errorKey* | The errorKey |
| *config* | TODO #3754 add parameter description and update other |
| *modules* | TODO #3754 add parameter description and update other |

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if `modules` , `config` or `errorKey` has already been released |

## 572.52.3 Member Function Documentation

### 572.52.3.1 close()

```
int org.libelektra.NativePlugin.close (
            Key errorKey )  [inline]
```

Closes the session with the Key database.

**Parameters**

| | |
|---|---|
| *errorKey* | must be a valid key, e.g. created with Key.create() |

**Returns**

0 if success or -1 otherwise

Implements org.libelektra.Plugin.

### 572.52.3.2 getConfig()

```
KeySet org.libelektra.NativePlugin.getConfig ( )  [inline]
```

Gets the config which was used to configure the plugin.

**Returns**

A KeySet containing the configuration of the plugin

### 572.52.3.3 kdbOpen()

```
int org.libelektra.NativePlugin.kdbOpen (
            Key errorKey )  [inline]
```

Opens the session with the KeyDatabase.

**Parameters**

| | |
|---|---|
| *errorKey* | must be a valid key, e.g. created with Key.create() |

**Returns**

0 if success or -1 otherwise

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if `errorKey` has already been released |

### 572.52.3.4 open()

```
int org.libelektra.NativePlugin.open (
            KeySet config,
            Key errorKey )  [inline]
```

Calls the plugin's open function.

**Parameters**

| | |
|---|---|
| *config* | Plugin configuration key set |
| *errorKey* | Used to store warnings and error information |

**Returns**

Plugin's return value for open

**See also**

STATUS_SUCCESS

STATUS_ERROR

Implements org.libelektra.Plugin.

#### 572.52.3.5 set()

```
int org.libelektra.NativePlugin.set (
            KeySet keySet,
            Key errorKey ) throws KDBException  [inline]
```

Lets the plugin transform the given KeySet.

**Parameters**

| | |
|---|---|
| *keySet* | The KeySet to transform |
| *errorKey* | must be a valid key, e.g. created with Key.create() |

**Returns**

0 if success or -1 otherwise

**Exceptions**

| | |
|---|---|
| *KDBException* | if return value was -1 |
| *IllegalStateException* | if `keySet` or `errorKey` has already been released |

Implements org.libelektra.Plugin.
The documentation for this class was generated from the following file:

- NativePlugin.java

## 572.53 kdb::none_t Class Reference

This type is being used as bottom type that always fails.
`#include <kdbvalue.hpp>`

### 572.53.1 Detailed Description

This type is being used as bottom type that always fails.

The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.54 kdb::ObserverPolicyIs< Policy > Class Template Reference

Needed by the user to set one of the policies.
`#include <kdbvalue.hpp>`
Inherits kdb::DefaultPolicies.

### 572.54.1 Detailed Description

**template<typename Policy>**
**class kdb::ObserverPolicyIs< Policy >**

Needed by the user to set one of the policies.

**Template Parameters**

| | |
|---|---|
| *Policy* | |

The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.55 Opmphm Struct Reference

The opmphm.
`#include <kdbopmphm.h>`

### Public Attributes

- int32_t * hashFunctionSeeds
- size_t componentSize
- uint32_t * graph
- size_t size
- opmphmflag_t flags

### 572.55.1 Detailed Description

The opmphm.

### 572.55.2 Member Data Documentation

#### 572.55.2.1 componentSize

`size_t Opmphm::componentSize`
< number of components in the r-uniform r-partite hypergraph the number of vertices in one part of the r-uniform r-partite hypergraph

#### 572.55.2.2 flags

`opmphmflag_t Opmphm::flags`
internal flags

#### 572.55.2.3 graph

`uint32_t* Opmphm::graph`
array containing the final OPMPHM

#### 572.55.2.4 hashFunctionSeeds

`int32_t* Opmphm::hashFunctionSeeds`
arary with Opmphm->rUniPar seeds for the hash function calls

#### 572.55.2.5 size

`size_t Opmphm::size`
size of g in bytes
The documentation for this struct was generated from the following file:

- kdbopmphm.h

## 572.56 OpmphmEdge Struct Reference

Order Preserving Minimal Perfect Hash Map.
`#include <kdbopmphm.h>`

### Public Attributes

- uint32_t order
- uint32_t * nextEdge
- uint32_t * vertices

### 572.56.1 Detailed Description

Order Preserving Minimal Perfect Hash Map.
Based on the work of
Fabiano C. Botelho and Nivio Ziviani "Near-Optimal Space Perfect Hashing Algorithms"
and
Zbigniew J. Czech, George Havas, and Bohdan S. Majewski "An Optimal Algorithm for Generating Minimal Perfect Hash Functions" In: Information Processing Letters 43 (1992), pp. 257–264
For usage look in datastructures.md
Theoretical limit of elements:
The whole OPMPHM is limited to the opmphmHashfunction (...) that returns a uint32_t. The limit of elements is than ((2^32) - 1) * r / c, since (2^32) - 1 is the maximum component size of the r-uniform r-partite hypergraph.
To save space the limit of elements is set to (2^32) - 1. The r-uniform r-partite hypergraph

### 572.56.2 Member Data Documentation

#### 572.56.2.1 nextEdge

`uint32_t* OpmphmEdge::nextEdge`
arary with Opmphm->rUniPar indices of the next edge in the lists

#### 572.56.2.2 order

`uint32_t OpmphmEdge::order`
desired hash map return value

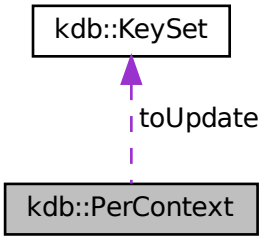
#### 572.56.2.3 vertices

`uint32_t* OpmphmEdge::vertices`
array with Opmphm->rUniPar indices of vertices that the edge connects
The documentation for this struct was generated from the following file:

- kdbopmphm.h

## 572.57 kdb::PerContext Struct Reference

A data structure that is stored by context inside the Coordinator.
`#include <kdbthread.hpp>`
Collaboration diagram for kdb::PerContext:




### 572.57.1 Detailed Description

A data structure that is stored by context inside the Coordinator.
The documentation for this struct was generated from the following file:

- kdbthread.hpp

## 572.58 kdb::tools::Plugin Class Reference

This is a C++ representation of a plugin.
`#include <plugin.hpp>`

### Public Member Functions

- Plugin (PluginSpec const &spec, kdb::KeySet &modules)

  *Do not construct a plugin yourself, use Modules.load.*
- void loadInfo ()

  *Gets the configuration for the plugin.*
- void parse ()

  *Creates symbol and info table.*
- void check (std::vector< std::string > &warnings)

  *Does various checks on the Plugin and throws exceptions if something is not ok.*
- std::string lookupInfo (std::string item, std::string section="infos")

  *Gets the whole string of an information item.*
- bool findInfo (std::string check, std::string item, std::string section="infos")

  *Searches within a string of an information item.*

- kdb::KeySet getInfo ()

  *Returns the whole keyset of information.*
- kdb::KeySet getNeededConfig ()

  *In the plugin's contract there is a description of which config is needed in order to work together with a backend properly.*
- kdb::KeySet getConfig ()

  *return the plugin config*
- func_t getSymbol (std::string which)

  *Returns symbol to a function.*
- int open (kdb::Key &errorKey)

  *Calls the open function of the plugin.*
- int close (kdb::Key &errorKey)

  *Calls the close function of the plugin.*
- int get (kdb::KeySet &ks, kdb::Key &parentKey)

  *Calls the get function of the plugin.*
- int set (kdb::KeySet &ks, kdb::Key &parentKey)

  *Calls the set function of the plugin.*
- int commit (kdb::KeySet &ks, kdb::Key &parentKey)

  *Calls the commit function of the plugin.*
- int error (kdb::KeySet &ks, kdb::Key &parentKey)

  *Calls the error function of the plugin.*
- std::string name ()
- std::string getFullName ()

## Public Attributes

- bool firstRef

  *Is toggled during serialization.*

## 572.58.1 Detailed Description

This is a C++ representation of a plugin.
It will load an Elektra plugin using the module loader from Elektra.
Then you can either check the plugins configuration using loadInfo(), parse() and check. Symbols can then be retrieved with getSymbol().
Or you can use the normal open(), close(), get(), set() and error() API which every plugin exports.

## 572.58.2 Member Function Documentation

### 572.58.2.1 check()

```
void kdb::tools::Plugin::check (
            std::vector< std::string > & warnings )
```

Does various checks on the Plugin and throws exceptions if something is not ok.

- Check if Plugin is compatible to current Version of Backend-API.

**Exceptions**

| | |
|---|---|
| *PluginCheckException* | if there are errors |

**Parameters**

| | |
|---|---|
| *warnings* | for warnings |

**Precondition**

parse()

#### 572.58.2.2 close()

```
int kdb::tools::Plugin::close (
            kdb::Key & errorKey )
```

Calls the close function of the plugin.

**Precondition**

parse()

#### 572.58.2.3 commit()

```
int kdb::tools::Plugin::commit (
            kdb::KeySet & ks,
            kdb::Key & parentKey )
```

Calls the commit function of the plugin.

**Precondition**

parse()

#### 572.58.2.4 error()

```
int kdb::tools::Plugin::error (
            kdb::KeySet & ks,
            kdb::Key & parentKey )
```

Calls the error function of the plugin.

**Precondition**

parse()

#### 572.58.2.5 findInfo()

```
bool kdb::tools::Plugin::findInfo (
            std::string check,
            std::string item,
            std::string section = "infos" )
```

Searches within a string of an information item.

**Precondition**

loadInfo()

#### 572.58.2.6 get()

```
int kdb::tools::Plugin::get (
            kdb::KeySet & ks,
            kdb::Key & parentKey )
```

Calls the get function of the plugin.

**Precondition**

parse()

#### 572.58.2.7 getConfig()

```
kdb::KeySet kdb::tools::Plugin::getConfig ( )
```

return the plugin config

**Returns**

the config supplied with constructor

**See also**

getNeededConfig()

#### 572.58.2.8 getFullName()

```
std::string kdb::tools::Plugin::getFullName ( )
```

**Returns**

the fullname of the plugin

#### 572.58.2.9 getInfo()

```
kdb::KeySet kdb::tools::Plugin::getInfo ( )  [inline]
```

Returns the whole keyset of information.

**Precondition**

loadInfo()

#### 572.58.2.10 getNeededConfig()

```
kdb::KeySet kdb::tools::Plugin::getNeededConfig ( )
```

In the plugin's contract there is a description of which config is needed in order to work together with a backend properly.

**Returns**

the keyset with the config needed for the backend.

**See also**

getConfig()

**Precondition**

loadInfo()

#### 572.58.2.11 getSymbol()

```
func_t kdb::tools::Plugin::getSymbol (
            std::string which )  [inline]
```

Returns symbol to a function.

**Precondition**

parse()

#### 572.58.2.12 loadInfo()

```
void kdb::tools::Plugin::loadInfo ( )
```

Gets the configuration for the plugin.
(will be done in Modules.load)

#### 572.58.2.13 lookupInfo()

```
std::string kdb::tools::Plugin::lookupInfo (
            std::string item,
            std::string section = "infos" )
```

Gets the whole string of an information item.

**Precondition**

loadInfo()

#### 572.58.2.14 name()

```
std::string kdb::tools::Plugin::name ( )
```

**Returns**

the name of the plugin (module)

#### 572.58.2.15 open()

```
int kdb::tools::Plugin::open (
            kdb::Key & errorKey )
```

Calls the open function of the plugin.

**Precondition**

parse()

#### 572.58.2.16 parse()

```
void kdb::tools::Plugin::parse ( )
```

Creates symbol and info table.
(will be done in Modules.load)

#### 572.58.2.17 set()

```
int kdb::tools::Plugin::set (
            kdb::KeySet & ks,
            kdb::Key & parentKey )
```

Calls the set function of the plugin.

**Precondition**

parse()

### 572.58.3 Member Data Documentation

#### 572.58.3.1 firstRef

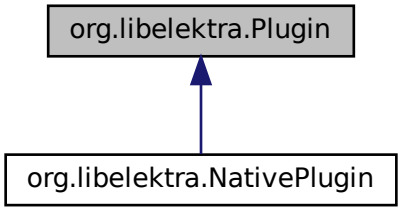
`bool kdb::tools::Plugin::firstRef`

Is toggled during serialization.
(is a hack, only allows a single serialization!)
The documentation for this class was generated from the following files:

- plugin.hpp
- plugin.cpp

## 572.59 org.libelektra.Plugin Interface Reference

Java representation of an Elektra plugin.
Inheritance diagram for org.libelektra.Plugin:




### Public Member Functions

- String getName ()
- int open (KeySet config, Key errorKey)

  *Calls the plugin's open function.*
- int get (KeySet keySet, Key parentKey) throws KDBException

  *Calls the plugin's get function.*
- int set (KeySet keySet, Key parentKey) throws KDBException

  *Calls the set function of the plugin.*
- int error (KeySet keySet, Key parentKey)

  *Calls the error function of the plugin.*
- int close (Key parentKey)

  *Calls the close function of the plugin.*

### Static Public Attributes

- static final String JNI_MODULE_CONTRACT_ROOT = "system:/elektra/modules/jni"

  *This is the root key of the JNI plugin wrapping a Java plugin for use by Elektra.*
- static final String PROCESS_CONTRACT_ROOT = "system:/elektra/modules/java"

*This is the root key of the process plugin wrapping a Java plugin for use by Elektra.*

- static final int STATUS_ERROR = -1

  *Return value for plugin methods: An error occurred inside the plugin function.*

- static final int STATUS_SUCCESS = 1

  *Return value for plugin methods: Everything went fine.*

- static final int STATUS_NO_UPDATE = 0

  *Return value for plugin methods: Everything went fine and the function **did not** update the given key set / configuration.*

## 572.59.1 Detailed Description

Java representation of an Elektra plugin.
@implNote because of interface inheritance, it is required that all methods (open, get, set, error, close) are implemented, even if they are not supported. Whether or not a method is supported, must be defined via the correspoding `exports/has` key of the contract. Any method that is not supported, should simply be implemented as `throw new UnsupportedOperationException().` If get isn't supported, you must still implement it and return the contract, when the parent key is below (or the same as) PROCESS_CONTRACT_ROOT. For other parent keys, you can safely throw UnsupportedOperationException.

## 572.59.2 Member Function Documentation

### 572.59.2.1 close()

```
int org.libelektra.Plugin.close (
            Key parentKey )
```

Calls the close function of the plugin.

**Parameters**

| | |
|---|---|
| *parentKey* | a key |

**Returns**

the plugin's return value for close

Implemented in org.libelektra.NativePlugin.

### 572.59.2.2 error()

```
int org.libelektra.Plugin.error (
            KeySet keySet,
            Key parentKey )
```

Calls the error function of the plugin.

**Parameters**

| | |
|---|---|
| *keySet* | a keyset |
| *parentKey* | a key |

**Returns**

the plugin's return value for error

#### 572.59.2.3 get()

```
int org.libelektra.Plugin.get (
            KeySet keySet,
            Key parentKey ) throws KDBException
```

Calls the plugin's get function.

**Parameters**

| | |
|---|---|
| *keySet* | Key set to store the retrieved keys in |
| *parentKey* | Parent key for retrieval |

**Exceptions**

| | |
|---|---|
| *KDBException* | if Elektra could not set the key set |

**Returns**

the plugin's return value for get

**See also**

STATUS_SUCCESS

STATUS_ERROR

#### 572.59.2.4 getName()

```
String org.libelektra.Plugin.getName ( )
```

**Returns**

Name of the plugin

#### 572.59.2.5 open()

```
int org.libelektra.Plugin.open (
            KeySet config,
            Key errorKey )
```

Calls the plugin's open function.

**Parameters**

| | |
|---|---|
| *config* | Plugin configuration key set |
| *errorKey* | Used to store warnings and error information |

**Returns**

Plugin's return value for open

**See also**

STATUS_SUCCESS

STATUS_ERROR

Implemented in org.libelektra.NativePlugin.

#### 572.59.2.6 set()

```
int org.libelektra.Plugin.set (
            KeySet keySet,
            Key parentKey ) throws KDBException
```

Calls the set function of the plugin.

**Parameters**

| | |
|---|---|
| *keySet* | a keyset |
| *parentKey* | a key |

**Exceptions**

| | |
|---|---|
| *KDBException* | when Elektra could not set the keyset |

**Returns**

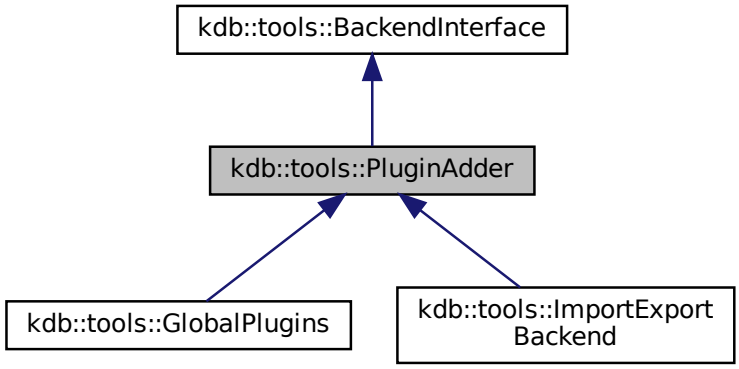
the plugin's return value for set

Implemented in org.libelektra.NativePlugin.
The documentation for this interface was generated from the following file:

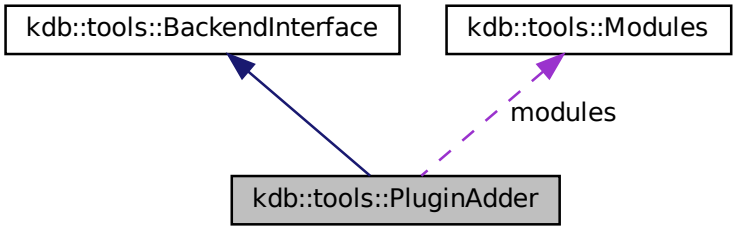
- Plugin.java

## 572.60 kdb::tools::PluginAdder Class Reference

Adds plugins in a generic map.
`#include <backend.hpp>`
Inheritance diagram for kdb::tools::PluginAdder:



Collaboration diagram for kdb::tools::PluginAdder:



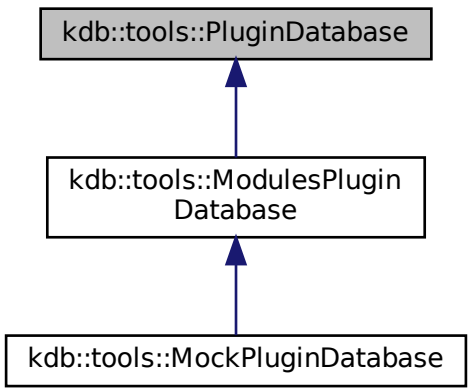

### 572.60.1 Detailed Description

Adds plugins in a generic map.
The documentation for this class was generated from the following files:

- backend.hpp
- src/backend.cpp

## 572.61 kdb::tools::PluginDatabase Class Reference

Loads all plugins and allows us to query them.
`#include <plugindatabase.hpp>`
Inheritance diagram for kdb::tools::PluginDatabase:




### Public Types

- enum Status { provides , real , missing }

### Public Member Functions

- virtual std::vector< std::string > listAllPlugins () const =0

*list all plugins*

- virtual std::string lookupInfo (PluginSpec const &whichplugin, std::string const &which) const =0

  *lookup contract clauses or dynamic information*
- virtual func_t getSymbol (PluginSpec const &whichplugin, std::string const &which) const =0

  *get exported plugin symbol*
- virtual PluginSpec lookupMetadata (std::string const &which) const =0

  *lookup which plugin handles metadata*
- virtual PluginSpec lookupProvides (std::string const &provides) const =0

  *lookup which plugin is a provider for that plugin*
- virtual std::map< int, PluginSpec > lookupAllProvidesWithStatus (std::string const &provides) const =0

  *looks up all plugins which are a suitable provider*
- virtual std::vector< PluginSpec > lookupAllProvides (std::string const &provides) const =0

  *looks up all plugins which are a suitable provider*

## Static Public Member Functions

- static int calculateStatus (std::string statusString)

### 572.61.1 Detailed Description

Loads all plugins and allows us to query them.

### 572.61.2 Member Enumeration Documentation

#### 572.61.2.1 Status

`enum kdb::tools::PluginDatabase::Status`

**Enumerator**

| | |
|---|---|
| provides | does not directly, but can be loaded via provides |
| real | exists and working as given |
| missing | does not exist or cannot be loaded |

### 572.61.3 Member Function Documentation

#### 572.61.3.1 calculateStatus()

```
int kdb::tools::PluginDatabase::calculateStatus (
            std::string statusString )  [static]
```

**Parameters**

| | |
|---|---|
| *statusString* | the string encoding the status |

**Returns**

The representing number for a given status.

#### 572.61.3.2 getSymbol()

```
virtual func_t kdb::tools::PluginDatabase::getSymbol (
            PluginSpec const & whichplugin,
            std::string const & which ) const  [pure virtual]
```

get exported plugin symbol

**Parameters**

| | |
|---|---|
| *whichplugin* | from which plugin? |
| *which* | which symbol would you like to look up? |

**Returns**

the function pointer to the exported symbol or NULL if the symbol was not found

Implemented in kdb::tools::MockPluginDatabase, and kdb::tools::ModulesPluginDatabase.

#### 572.61.3.3 listAllPlugins()

```
virtual std::vector<std::string> kdb::tools::PluginDatabase::listAllPlugins ( ) const  [pure
virtual]
```

list all plugins
If Elektra is compiled with plugins, it will search for shared libraries. In any case, if no shared libraries were found it will fallback to an internal list (plugins that were compiled together with Elektra).

**Returns**

a list of all available plugins

Implemented in kdb::tools::MockPluginDatabase, and kdb::tools::ModulesPluginDatabase.

#### 572.61.3.4 lookupAllProvides()

```
virtual std::vector<PluginSpec> kdb::tools::PluginDatabase::lookupAllProvides (
            std::string const & provides ) const  [pure virtual]
```

looks up all plugins which are a suitable provider

**Note**

in case a plugin name is provided, the plugin with the name will also be part of the result. But if there are other plugins providing the requirement, then they will also be part of the result. The ordering of the resulting vector has no special meaning.

**Parameters**

| | |
|---|---|
| *provides* | is the provider to find |

**Returns**

a vector of plugins offering the requirement or are named after it

Implemented in kdb::tools::ModulesPluginDatabase.

#### 572.61.3.5 lookupAllProvidesWithStatus()

```
virtual std::map<int, PluginSpec> kdb::tools::PluginDatabase::lookupAllProvidesWithStatus (
            std::string const & provides ) const  [pure virtual]
```

looks up all plugins which are a suitable provider

**Note**

in case a plugin name is provided, the plugin with the name will also be part of the result. But if there are other plugins providing the requirement, then they will also be part of the result.

**Parameters**

| | |
|---|---|
| *provides* | is the provider to find |

**Returns**

a map of plugins with their status offering the requirement or are named after it

Implemented in kdb::tools::ModulesPluginDatabase.

#### 572.61.3.6 lookupInfo()

```
virtual std::string kdb::tools::PluginDatabase::lookupInfo (
            PluginSpec const & whichplugin,
            std::string const & which ) const  [pure virtual]
```

lookup contract clauses or dynamic information

**Parameters**

| | |
|---|---|
| *whichplugin* | about which plugin? |
| *which* | about which clause in the contract? |

**Returns**

the clause of the contract

Implemented in kdb::tools::MockPluginDatabase, and kdb::tools::ModulesPluginDatabase.

#### 572.61.3.7 lookupMetadata()

```
virtual PluginSpec kdb::tools::PluginDatabase::lookupMetadata (
            std::string const & which ) const  [pure virtual]
```

lookup which plugin handles metadata

**Parameters**

| | |
|---|---|
| *which* | the metadata of interest |

**Returns**

the best suited plugin specification which provides it

Implemented in kdb::tools::ModulesPluginDatabase.

#### 572.61.3.8 lookupProvides()

```
virtual PluginSpec kdb::tools::PluginDatabase::lookupProvides (
            std::string const & provides ) const  [pure virtual]
```

lookup which plugin is a provider for that plugin

**Note**

will return a PluginSpec with getName() == provides if the string provides actually is a plugin name.

**Parameters**

| | |
|---|---|
| *provides* | is the provider to find |

**Exceptions**

| | |
|---|---|
| *NoPlugin* | if no plugin that provides the functionality could be found |

**Returns**

the plugin itself or the best suited plugin specification which provides it

Implemented in kdb::tools::ModulesPluginDatabase.
The documentation for this class was generated from the following files:

- plugindatabase.hpp
- plugindatabase.cpp

# 572.62 org.libelektra.PluginLoader Class Reference

This class can be used to load plugins from Elektra.

## Public Member Functions

- PluginLoader (Key errorKey)

  *Instantiates a new PluginLoader with the possibility to add a custom error key.*
- PluginLoader ()

  *Instantiates a new PluginLoader with a default error key which is empty.*
- Plugin loadElektraPlugin (String name) throws InstallationException

  *This plugin loads a Native Elektra Plugin.*

## 572.62.1 Detailed Description

This class can be used to load plugins from Elektra.

## 572.62.2 Constructor & Destructor Documentation

### 572.62.2.1 PluginLoader()

```
org.libelektra.PluginLoader.PluginLoader (
            Key errorKey ) [inline]
```

Instantiates a new PluginLoader with the possibility to add a custom error key.

**Parameters**

| | |
|---|---|
| *errorKey* | The custom error key |

### 572.62.3 Member Function Documentation

#### 572.62.3.1 loadElektraPlugin()

```
Plugin org.libelektra.PluginLoader.loadElektraPlugin (
            String name ) throws InstallationException  [inline]
```

This plugin loads a Native Elektra Plugin.

**Parameters**

| | |
|---|---|
| *name* | the plugin name |

**Returns**

the Plugin

**Exceptions**

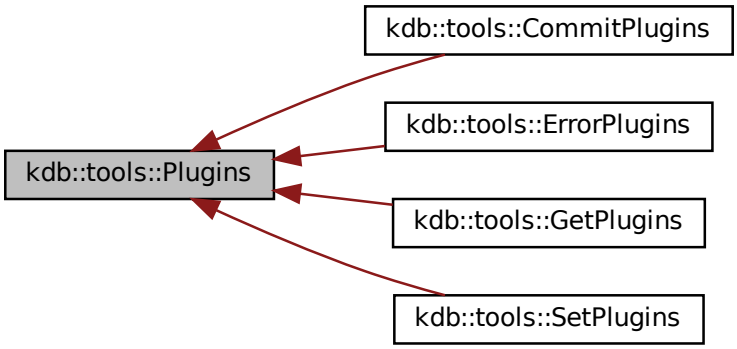
| | |
|---|---|
| *InstallationException* | if the plugin does not exist |

The documentation for this class was generated from the following file:

- PluginLoader.java

## 572.63 kdb::tools::Plugins Class Reference

A collection of plugins (either get, set or error)
`#include <plugins.hpp>`
Inheritance diagram for kdb::tools::Plugins:




### Public Member Functions

- void addInfo (Plugin &plugin)

  *Add needed, provided and recommend information.*
- bool validateProvided () const

*Validate needed, and provided information.*

- bool checkPlacement (Plugin &plugin, std::string which)

  *check if this plugin has at least one placement*
- void checkOrdering (Plugin &plugin)

  *Check ordering of plugins.*
- void checkConflicts (Plugin &plugin)

  *Check conflicts of plugins.*

### 572.63.1 Detailed Description

A collection of plugins (either get, set or error)

### 572.63.2 Member Function Documentation

#### 572.63.2.1 checkPlacement()

```
bool kdb::tools::Plugins::checkPlacement (
            Plugin & plugin,
            std::string which )
```

check if this plugin has at least one placement

**Parameters**

| | |
|---|---|
| *plugin* | the plugin to check |
| *which* | placementInfo it is |

**Return values**

| | |
|---|---|
| *true* | if it should be added |
| *false* | no placements (will not be added) |

#### 572.63.2.2 validateProvided()

```
bool kdb::tools::Plugins::validateProvided ( ) const
```

Validate needed, and provided information.
(Recommended ignored,

**See also**

getRecommendedMissing(),

getNeededMissing()

The documentation for this class was generated from the following files:

- plugins.hpp
- plugins.cpp

## 572.64 kdb::tools::PluginSpec Class Reference

Specifies a plugin by its name and configuration.
`#include <pluginspec.hpp>`

## Public Member Functions

- PluginSpec (std::string pluginName, KeySet pluginConfig=KeySet())

  *Construct a plugin spec with a given module name.*
- PluginSpec (std::string pluginName, std::string refName, KeySet pluginConfig=KeySet())

  *Construct a plugin spec with a given module and ref name.*
- PluginSpec (std::string pluginName, size_t refNumber, KeySet pluginConfig=KeySet())

  *Construct a plugin spec with a given module name and ref number.*
- std::string getFullName () const
- std::string getRefName () const
- bool isRefNumber () const

  *Checks if reference name contains only numbers.*
- std::string getName () const
- KeySet getConfig () const
- void setFullName (std::string const &name)

  *Set the full name with # or only the name.*
- void setRefName (std::string const &name)

  *Set the reference name of the plugin.*
- void setRefNumber (size_t number)

  *Set a number for automatic references during parsing.*
- void setName (std::string const &name)

  *Set the module name of the plugin.*
- void appendConfig (KeySet config)

  *Append to config.*
- void setConfig (KeySet config)

  *Set plugin config.*
- void validate (std::string const &str) const

  *Check if str starts with a-z and then only has chars a-z, 0-9 or underscore (_)*

### 572.64.1 Detailed Description

Specifies a plugin by its name and configuration.

**Invariant**

name is valid (nonempty, starts with a-z, then a-z_0-9)

refname is valid (same as above or a size_t number)

### 572.64.2 Constructor & Destructor Documentation

#### 572.64.2.1 PluginSpec() [1/3]

```
kdb::tools::PluginSpec::PluginSpec (
            std::string pluginName,
            KeySet pluginConfig = KeySet () )  [explicit]
```

Construct a plugin spec with a given module name.

**Parameters**

| | |
|---|---|
| *pluginName* | the fullname (modulename # refname) |
| *pluginConfig* | the plugins config |

**Exceptions**

| | |
|---|---|
| *BadPluginName* | if name not a-z |

**See also**

setFullName()

#### 572.64.2.2 PluginSpec() [2/3]

```
kdb::tools::PluginSpec::PluginSpec (
            std::string pluginName,
            std::string refName,
            KeySet pluginConfig = KeySet () )  [explicit]
```

Construct a plugin spec with a given module and ref name.

**Parameters**

| | |
|---|---|
| *pluginName* | the module this plugin is instantiated from |
| *refName* | for uniqueness for more plugins from one module |
| *pluginConfig* | the plugins config |

**Exceptions**

| | |
|---|---|
| *BadPluginName* | if name not a-z |

**See also**

setName()

setRefName()

#### 572.64.2.3 PluginSpec() [3/3]

```
kdb::tools::PluginSpec::PluginSpec (
            std::string pluginName,
            size_t refNumber,
            KeySet pluginConfig = KeySet () )  [explicit]
```

Construct a plugin spec with a given module name and ref number.

**Parameters**

| | |
|---|---|
| *pluginName* | the module this plugin is instantiated from |
| *refNumber* | for uniqueness for more plugins from one module |
| *pluginConfig* | the plugins config |

**Exceptions**

| | |
|---|---|
| *BadPluginName* | if name not a-z |

**See also**

setName()

setRefName()

## 572.64.3 Member Function Documentation

### 572.64.3.1 appendConfig()

```
void kdb::tools::PluginSpec::appendConfig (
            KeySet c )
```

Append to config.

**Parameters**

| | |
|---|---|
| *c* | config to append |

### 572.64.3.2 getConfig()

```
KeySet kdb::tools::PluginSpec::getConfig ( ) const
```

**Returns**

the config

### 572.64.3.3 getFullName()

```
std::string kdb::tools::PluginSpec::getFullName ( ) const
```

**Note**

if no reference name is given its equal to the module name

**Returns**

the module name, then #, and then the reference name

### 572.64.3.4 getName()

```
std::string kdb::tools::PluginSpec::getName ( ) const
```

**Returns**

the module name

### 572.64.3.5 getRefName()

```
std::string kdb::tools::PluginSpec::getRefName ( ) const
```

**Returns**

the reference name

#### 572.64.3.6 isRefNumber()

```
bool kdb::tools::PluginSpec::isRefNumber ( ) const
```

Checks if reference name contains only numbers.

**Return values**

| | |
|---|---|
| *true* | if only numbers |
| *true* | if a refname was given |

#### 572.64.3.7 setConfig()

```
void kdb::tools::PluginSpec::setConfig (
            KeySet c )
```

Set plugin config.

**Parameters**

| | |
|---|---|
| *c* | new config to be used as plugin config |

#### 572.64.3.8 setFullName()

```
void kdb::tools::PluginSpec::setFullName (
            std::string const & n )
```

Set the full name with # or only the name.

**Exceptions**

| | |
|---|---|
| *BadPluginName* | if name not a-z (no change then) |

**Parameters**

| | |
|---|---|
| *n* | the string to set |

#### 572.64.3.9 setName()

```
void kdb::tools::PluginSpec::setName (
            std::string const & n )
```

Set the module name of the plugin.

**Exceptions**

| | |
|---|---|
| *BadPluginName* | if name not a-z (no change then) |

**Parameters**

| | |
|---|---|
| *n* | the string to set |

#### 572.64.3.10 setRefName()

```
void kdb::tools::PluginSpec::setRefName (
            std::string const & n )
```

Set the reference name of the plugin.

**Exceptions**

| | |
|---|---|
| *BadPluginName* | if name not a-z (no change then) |

Makes different plugins based from the same module unique.

**Parameters**

| | |
|---|---|
| *n* | the string to set |

#### 572.64.3.11 setRefNumber()

```
void kdb::tools::PluginSpec::setRefNumber (
            size_t refnumber )
```

Set a number for automatic references during parsing.

**Parameters**

| | |
|---|---|
| *refnumber* | the number to set |

#### 572.64.3.12 validate()

```
void kdb::tools::PluginSpec::validate (
            std::string const & n ) const
```

Check if str starts with a-z and then only has chars a-z, 0-9 or underscore (_)

**Parameters**

| | |
|---|---|
| *n* | the string to check |

The documentation for this class was generated from the following files:

- pluginspec.hpp
- pluginspec.cpp

## 572.65 kdb::tools::PluginSpecHash Struct Reference

Only to be used with PluginSpecName!

```
#include <pluginspec.hpp>
```

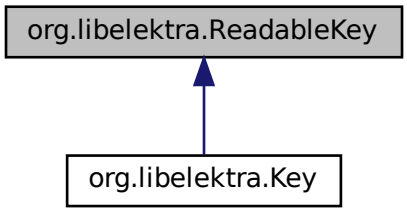
### 572.65.1 Detailed Description

Only to be used with PluginSpecName!
The documentation for this struct was generated from the following file:

- pluginspec.hpp

# 572.66 org.libelektra.ReadableKey Class Reference

Read only key representing a native Elektra key providing read access to its name and value.
Inheritance diagram for org.libelektra.ReadableKey:




## Public Member Functions

- String toString ()
- Iterator< String > keyNameIterator ()
- boolean getBoolean ()
- byte getByte ()
- short getShort ()
- int getInt ()
- long getLong ()
- float getFloat ()
- double getDouble ()
- String getString ()
- Key dup ()

  *Duplicates this ReadableKey as Key.*
- Key dup (int flags)

  *Duplicates this ReadableKey as Key.*
- int compareTo (ReadableKey other)

  *Compares this key with the.*
- boolean isBelow (ReadableKey other)

  *Checks whether this key is sub-key of the.*
- boolean isBelowOrSame (ReadableKey other)

  *Checks whether this key is the same as the.*
- boolean isDirectlyBelow (ReadableKey other)

  *Checks whether this key is direct sub-key of the.*
- boolean isBinary ()
- boolean isString ()
- String getName ()
- int getNameSize ()
- String getBaseName ()
- int getBaseNameSize ()
- int getValueSize ()
- boolean isNull ()

## Static Public Attributes

- static final int KEY_CP_NAME = 1 << 0

  *Flag for use with Key#copy(Key, int) and dup(int) for copying the key name.*
- static final int KEY_CP_STRING = 1 << 1

  *Flag for use with Key#copy(Key, int) and dup(int) for copying the key value, if it is a string.*
- static final int KEY_CP_VALUE = 1 << 2

  *Flag for use with Key#copy(Key, int) and dup(int) for copying the key value.*
- static final int KEY_CP_META = 1 << 3

  *Flag for use with Key#copy(Key, int) and dup(int) for copying the key metadata.*
- static final int KEY_CP_ALL = KEY_CP_NAME | KEY_CP_VALUE | KEY_CP_META

  *Flag for use with Key#copy(Key, int) and dup(int) for copying the key name, value and metadata.*

## Protected Member Functions

- ReadableKey (Pointer pointer)

  *Constructor associating a new ReadableKey instance with a JNA pointer.*
- ReadableKey (Pointer pointer, boolean suppressCleanUp)

  *Constructor associating a new ReadableKey instance with a JNA pointer*

  *Suppressing clean-up has been introduced for usage of this binding as JNI plug-in and should normally not be used in any other case.*
- void release ()

  *Clean-up method to release key reference by first decrementing its reference counter and then trying to free the native reference*

  *keys, will get cleaned up by garbage collection as soon as they get phantom reachable.*
- Pointer getPointer ()

## Static Protected Member Functions

- static Optional< ReadableKey > createReadOnly (@Nullable Pointer pointer)

  *Constructs a new ReadableKey instance associated with a JNA pointer.*

### 572.66.1 Detailed Description

Read only key representing a native Elektra key providing read access to its name and value.
@apiNote This abstraction is used to represent meta keys being read only by definition and cannot contain binary data

### 572.66.2 Constructor & Destructor Documentation

#### 572.66.2.1 ReadableKey() [1/2]

```
org.libelektra.ReadableKey.ReadableKey (
            Pointer pointer ) [inline], [protected]
```

Constructor associating a new ReadableKey instance with a JNA pointer.

**Parameters**

| | |
|---|---|
| *pointer* | JNA Pointer to key |

#### 572.66.2.2 ReadableKey() [2/2]

```
org.libelektra.ReadableKey.ReadableKey (
            Pointer pointer,
            boolean suppressCleanUp )  [inline], [protected]
```

Constructor associating a new ReadableKey instance with a JNA pointer

Suppressing clean-up has been introduced for usage of this binding as JNI plug-in and should normally not be used in any other case.

**Parameters**

| | |
|---|---|
| *pointer* | JNA Pointer to key |
| *suppressCleanUp* | True to suppress native reference clean-up as soon as this Key instance becomes phantom reachable, false otherwise |

### 572.66.3 Member Function Documentation

#### 572.66.3.1 compareTo()

```
int org.libelektra.ReadableKey.compareTo (
            ReadableKey other )  [inline]
```

Compares this key with the.
`other`
key by comparing the key name with string comparison

**Parameters**

| | |
|---|---|
| *other* | Other key to compare this key to |

**Returns**

- 0 if key name is equal
- -1 if this key name has lower alphabetical order than the `other` key
- 1 if this key has higher alphabetical order

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this or the `other` ReadableKey has already been released |
| *IllegalArgumentException* | if `other` is `null` |

#### 572.66.3.2 createReadOnly()

```
static Optional<ReadableKey> org.libelektra.ReadableKey.createReadOnly (
            @Nullable Pointer pointer )  [inline], [static], [protected]
```

Constructs a new ReadableKey instance associated with a JNA pointer.

**Parameters**

| | |
|---|---|
| *pointer* | Optional JNA Pointer to key |

**Returns**

New ReadableKey instance if `pointer` is non-null, {} otherwise

### 572.66.3.3 dup() [1/2]

```
Key org.libelektra.ReadableKey.dup ( )  [inline]
```

Duplicates this ReadableKey as Key.

**Returns**

New Key object containing the same information as this key

**Exceptions**

| | |
|---|---|
| *KeyException* | if copying failed |
| *IllegalStateException* | if this ReadableKey has already been released |

**See also**

dup(int)

### 572.66.3.4 dup() [2/2]

```
Key org.libelektra.ReadableKey.dup (
            int flags )  [inline]
```

Duplicates this ReadableKey as Key.

**Parameters**

| | |
|---|---|
| *flags* | Flags indicating which parts of the key to copy<br>Example:<br>KEY_CP_NAME \| KEY_CP_VALUE |

**Returns**

New Key object containing the same information as this key

**Exceptions**

| | |
|---|---|
| *KeyException* | if copying failed |
| *IllegalStateException* | if this ReadableKey has already been released |

**See also**

dup()
KEY_CP_ALL
KEY_CP_META
KEY_CP_NAME
KEY_CP_STRING
KEY_CP_VALUE

### 572.66.3.5 getBaseName()

```
String org.libelektra.ReadableKey.getBaseName ( )  [inline]
```

**Returns**

Key's base name as String

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.6 getBaseNameSize()

```
int org.libelektra.ReadableKey.getBaseNameSize ( )  [inline]
```

**Returns**

Length of key's base name

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.7 getBoolean()

```
boolean org.libelektra.ReadableKey.getBoolean ( )  [inline]
```

**See also**

Definition of Bool

**Returns**

getString() interpreted as boolean value

**Exceptions**

| | |
|---|---|
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.8 getByte()

```
byte org.libelektra.ReadableKey.getByte ( )  [inline]
```

**Returns**

getString() parsed as
`byte`

**Exceptions**

| | |
|---|---|
| *NumberFormatException* | if the getString() does not return a parsable `byte` |
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.9 getDouble()

```
double org.libelektra.ReadableKey.getDouble ( )  [inline]
```

**Returns**

getString() parsed as
`double`

**Exceptions**

| | |
|---|---|
| *NumberFormatException* | if the getString() does not return a parsable `double` |
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.10 getFloat()

```
float org.libelektra.ReadableKey.getFloat ( )  [inline]
```

**Returns**

getString() parsed as
`float`

**Exceptions**

| | |
|---|---|
| *NumberFormatException* | if the getString() does not return a parsable `float` |
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.11 getInt()

```
int org.libelektra.ReadableKey.getInt ( )  [inline]
```

**Returns**

getString() parsed as integer

**Exceptions**

| | |
|---|---|
| *NumberFormatException* | if the getString() does not return a parsable integer |
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.12 getLong()

```
long org.libelektra.ReadableKey.getLong ( )  [inline]
```

**Returns**

getString() parsed as
`long`

**Exceptions**

| | |
|---|---|
| *NumberFormatException* | if the getString() does not return a parsable `long` |
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.13 getName()

```
String org.libelektra.ReadableKey.getName ( )  [inline]
```

**Returns**

Key name (key part of "key-value" pair)

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.14 getNameSize()

```
int org.libelektra.ReadableKey.getNameSize ( )  [inline]
```

**Returns**

Length of key name

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.15 getPointer()

```
Pointer org.libelektra.ReadableKey.getPointer ( )  [inline], [protected]
```

**Returns**

JNA pointer to the native pointer for this key

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.16 getShort()

```
short org.libelektra.ReadableKey.getShort ( )  [inline]
```

**Returns**

getString() parsed as
`short`

**Exceptions**

| | |
|---|---|
| *NumberFormatException* | if the getString() does not return a parsable `short` |
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.17 getString()

```
String org.libelektra.ReadableKey.getString ( )  [inline]
```

**Returns**

This key's value as string

**Exceptions**

| | |
|---|---|
| *KeyStringValueException* | if the underlying native key is not of type string |
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.18 getValueSize()

```
int org.libelektra.ReadableKey.getValueSize ( )  [inline]
```

**Returns**

Length / size of key value in bytes

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.19 isBelow()

```
boolean org.libelektra.ReadableKey.isBelow (
            ReadableKey other )  [inline]
```

Checks whether this key is sub-key of the.
`other`
key

#### Parameters

| | |
|---|---|
| *other* | Key that is used in check as parent key |

#### Returns

Boolean if this key is (non-direct) sub-key of other-key

#### Exceptions

| | |
|---|---|
| *IllegalStateException* | if this or the `other` ReadableKey has already been released |
| *IllegalArgumentException* | if `other` is `null` |

### 572.66.3.20 isBelowOrSame()

```
boolean org.libelektra.ReadableKey.isBelowOrSame (
            ReadableKey other )  [inline]
```

Checks whether this key is the same as the.
`other`
key or a sub-key of the
`other`
key

#### Parameters

| | |
|---|---|
| *other* | Key that is used in check as parent key |

#### Returns

Boolean if this key is other key or (non-direct) sub-key of other-key

#### Exceptions

| | |
|---|---|
| *IllegalStateException* | if this or the `other` ReadableKey has already been released |
| *IllegalArgumentException* | if `other` is `null` |

### 572.66.3.21 isBinary()

```
boolean org.libelektra.ReadableKey.isBinary ( )  [inline]
```

**Returns**

True if the underlying native key's value is of type binary, false otherwise

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

### 572.66.3.22 isDirectlyBelow()

```
boolean org.libelektra.ReadableKey.isDirectlyBelow (
            ReadableKey other )  [inline]
```

Checks whether this key is direct sub-key of the.
`other`
key

**Parameters**

| | |
|---|---|
| *other* | Key that is used in check as parent key |

**Returns**

Boolean if this key is direct sub-key of other key ("child")

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this or the `other` ReadableKey has already been released |
| *IllegalArgumentException* | if `other` is `null` |

### 572.66.3.23 isNull()

```
boolean org.libelektra.ReadableKey.isNull ( )  [inline]
```

**Returns**

True, if the key has no value, false otherwise

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.24 isString()

```
boolean org.libelektra.ReadableKey.isString ( )  [inline]
```

**Returns**

True if the underlying native key's value is a valid string, false otherwise

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.25 keyNameIterator()

```
Iterator<String> org.libelektra.ReadableKey.keyNameIterator ( )  [inline]
```

**Returns**

New KeyNameIterator backed by this ReadableKey

**Exceptions**

| | |
|---|---|
| *IllegalStateException* | if this ReadableKey has already been released |

#### 572.66.3.26 toString()

```
String org.libelektra.ReadableKey.toString ( )  [inline]
```

**Returns**

Key name in string format as returned by getName()

### 572.66.4 Member Data Documentation

#### 572.66.4.1 KEY_CP_STRING

```
final int org.libelektra.ReadableKey.KEY_CP_STRING = 1 << 1  [static]
```

Flag for use with Key#copy(Key, int) and dup(int) for copying the key value, if it is a string.
@apiNote Do not use together with KEY_CP_VALUE

#### 572.66.4.2 KEY_CP_VALUE

```
final int org.libelektra.ReadableKey.KEY_CP_VALUE = 1 << 2  [static]
```

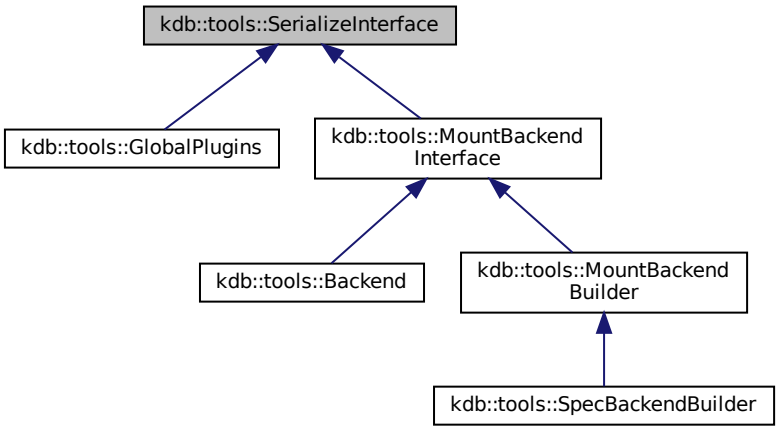
Flag for use with Key#copy(Key, int) and dup(int) for copying the key value.
@apiNote Do not use together with KEY_CP_STRING
The documentation for this class was generated from the following file:

- ReadableKey.java

## 572.67 kdb::tools::SerializeInterface Class Reference

Interface to serialize a backend.
`#include <backend.hpp>`

Inheritance diagram for kdb::tools::SerializeInterface:



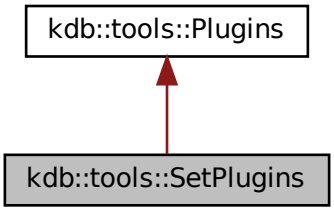

### 572.67.1 Detailed Description

Interface to serialize a backend.
The documentation for this class was generated from the following files:

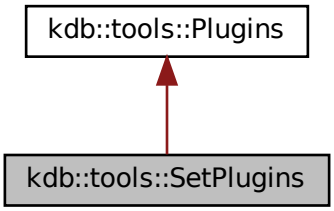
- backend.hpp
- src/backend.cpp

## 572.68 kdb::tools::SetPlugins Class Reference

Plugins to set configuration.
`#include <plugins.hpp>`
Inheritance diagram for kdb::tools::SetPlugins:



Collaboration diagram for kdb::tools::SetPlugins:




### 572.68.1 Detailed Description

Plugins to set configuration.
The documentation for this class was generated from the following files:

- plugins.hpp
- plugins.cpp

## 572.69 kdb::SetPolicyIs< Policy > Class Template Reference

Needed by the user to set one of the policies.
`#include <kdbvalue.hpp>`
Inherits kdb::DefaultPolicies.

### 572.69.1 Detailed Description

**template<typename Policy>**
**class kdb::SetPolicyIs< Policy >**

Needed by the user to set one of the policies.

**Template Parameters**

| | |
|---|---|
| *Policy* | |

The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.70 SomeloLibHandle Struct Reference

Example I/O management library data structure.

### Public Attributes

- void * data

  *Let's you access the context you supplied to the I/O management library.*

### 572.70.1 Detailed Description

Example I/O management library data structure.

The documentation for this struct was generated from the following file:

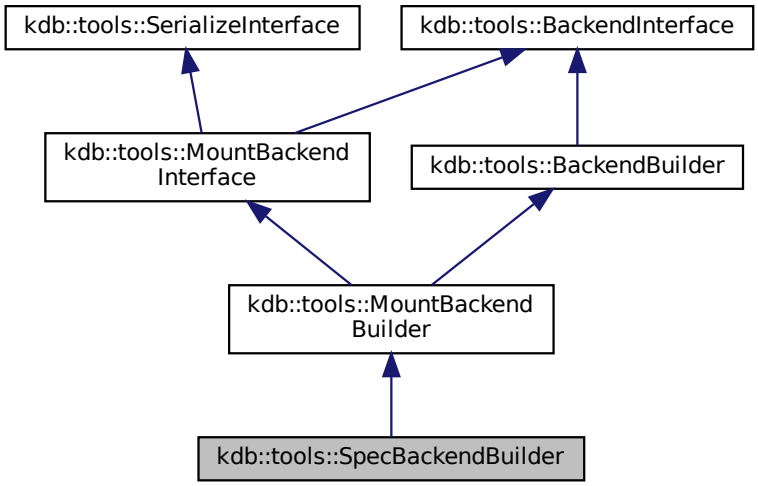
- io_doc.c

## 572.71 kdb::tools::SpecBackendBuilder Class Reference

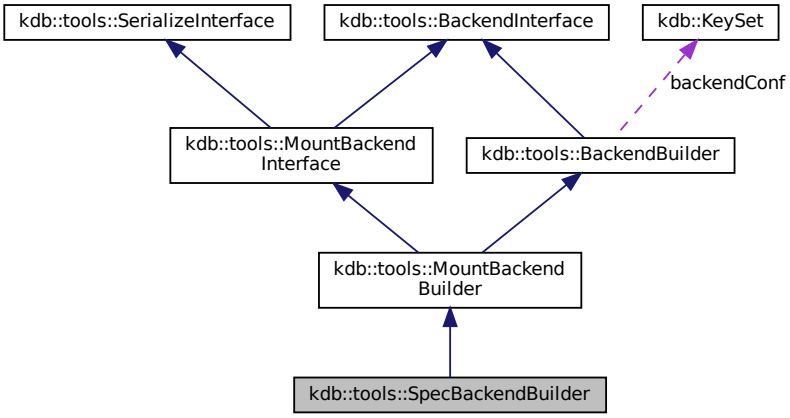
Build individual backend while reading specification.

`#include <specreader.hpp>`

Inheritance diagram for kdb::tools::SpecBackendBuilder:




Collaboration diagram for kdb::tools::SpecBackendBuilder:




### Additional Inherited Members

### 572.71.1 Detailed Description

Build individual backend while reading specification.

The documentation for this class was generated from the following files:

- specreader.hpp
- specreader.cpp

## 572.72 kdb::tools::SpecReader Class Reference

Highlevel interface to build a backend from specification.
`#include <specreader.hpp>`

### Public Member Functions

- Backends getBackends ()
- void readSpecification (KeySet const &ks)

  *Reads in a specification.*
- void checkKey (const Key key)

  *Perform some sanity-checks for keys.*

### 572.72.1 Detailed Description

Highlevel interface to build a backend from specification.

### 572.72.2 Member Function Documentation

#### 572.72.2.1 checkKey()

```
void kdb::tools::SpecReader::checkKey (
            const Key key )
```

Perform some sanity-checks for keys.

**Parameters**

| | |
|---|---|
| *key* | The key to check. |

**Exceptions**

| | |
|---|---|
| *CommandAbortException* | If a sanity-check fails. |

#### 572.72.2.2 getBackends()

```
Backends kdb::tools::SpecReader::getBackends ( )  [inline]
```

**Returns**

backends without resolved needs

**See also**

resolveNeeds()

#### 572.72.2.3 readSpecification()

```
void kdb::tools::SpecReader::readSpecification (
            KeySet const & ks )
```

Reads in a specification.
Adds plugins using BackendBuilder during that.

**Parameters**

| | |
|---|---|
| *ks* | |

The documentation for this class was generated from the following files:

- specreader.hpp
- specreader.cpp

## 572.73 kdb::ThreadSubject Class Reference

Subject from Observer pattern for ThreadContext.
`#include <kdbthread.hpp>`
Inherited by kdb::ThreadContext.

### 572.73.1 Detailed Description

Subject from Observer pattern for ThreadContext.
The documentation for this class was generated from the following file:

- kdbthread.hpp

## 572.74 kdb::tools::ToolException Struct Reference

All exceptions from the elektratools library are derived from this exception.
`#include <toolexcept.hpp>`
Inherits std::runtime_error.
Inherited by kdb::tools::BackendCheckException, kdb::tools::ParseException, kdb::tools::PluginCheckException, kdb::tools::helper::InvalidRebaseException, and kdb::tools::merging::InvalidConflictOperation.

### 572.74.1 Detailed Description

All exceptions from the elektratools library are derived from this exception.
The documentation for this struct was generated from the following file:

- toolexcept.hpp

## 572.75 kdb::VaAlloc Struct Reference

Needed to avoid constructor ambiguity.
`#include <keyset.hpp>`

### 572.75.1 Detailed Description

Needed to avoid constructor ambiguity.
when ... is same type as va_list
The documentation for this struct was generated from the following file:

- keyset.hpp

## 572.76 kdb::ValueObserver Class Reference

Base class for values to be observed.
`#include <kdbvalue.hpp>`
Inherited by kdb::Value< T, PolicySetter1, PolicySetter2, PolicySetter3, PolicySetter4, PolicySetter5, PolicySetter6 >.

### 572.76.1 Detailed Description

Base class for values to be observed.
updateContext() is called whenever a context tells a value that it should reevaluate its name and update its cache.
The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.77 kdb::Wrapped Class Reference

Everything implementing this interface can be used as layer.
`#include <kdbvalue.hpp>`
Inherited by kdb::Value< T, PolicySetter1, PolicySetter2, PolicySetter3, PolicySetter4, PolicySetter5, PolicySetter6 >.

### 572.77.1 Detailed Description

Everything implementing this interface can be used as layer.
Different from "Layer" objects they will not be constructed on activation but instead only the WrapLayer will be constructed and the wrapped object will be passed along by reference.

**Note**

the lifetime must be beyond layer deactivation!

The documentation for this class was generated from the following file:

- kdbvalue.hpp

## 572.78 kdb::WritePolicyIs< Policy > Class Template Reference

Needed by the user to set one of the policies.
`#include <kdbvalue.hpp>`
Inherits kdb::DefaultPolicies.

### 572.78.1 Detailed Description

**template<typename Policy>**
**class kdb::WritePolicyIs< Policy >**

Needed by the user to set one of the policies.

**Template Parameters**

| | |
|---|---|
| *Policy* | |

The documentation for this class was generated from the following file:

- kdbvalue.hpp

# Chapter 573

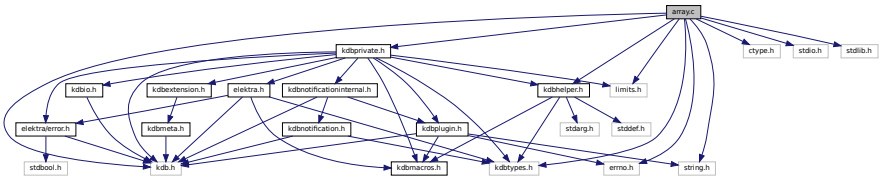
# File Documentation

## 573.1 array.c File Reference

Array methods.

```
#include <kdb.h>
#include <kdbease.h>
#include <kdbhelper.h>
#include <kdbprivate.h>
#include <kdbtypes.h>
#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for array.c:




### Functions

- int elektraArrayValidateName (const Key ∗key)

  *validate array syntax*
- int elektraArrayValidateBaseNameString (const char ∗baseName)

  *validate array syntax*
- char ∗ elektraArrayGetPrefix (Key ∗key)

  *Get the base name of the passed array.*
- int elektraArrayIncName (Key ∗key)

  *Increment the name of the key by one.*
- int elektraArrayDecName (Key ∗key)

  *Decrement the name of an array key by one.*
- KeySet ∗ elektraArrayGet (const Key ∗arrayParent, KeySet ∗keys)

  *Return all the array keys below the given array parent.*
- Key ∗ elektraArrayGetNextKey (KeySet ∗arrayKeys)

  *Return the next key in the given array.*

### 573.1.1 Detailed Description

Array methods.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.1.2 Function Documentation

#### 573.1.2.1 elektraArrayDecName()

```
int elektraArrayDecName (
            Key * key )
```

Decrement the name of an array key by one.
The alphabetical order will remain intact. For example, `user:/abc/\#_10` will be changed to `user:/abc/\#9`.

**Parameters**

| | |
|---|---|
| *This* | parameter determines the key name this function decrements. |

**Return values**

| | |
|---|---|
| *-1* | on error (e.g. new array index too small, non-valid array) |
| *0* | on success |

#### 573.1.2.2 elektraArrayGet()

```
KeySet* elektraArrayGet (
            const Key * arrayParent,
            KeySet * keys )
```

Return all the array keys below the given array parent.
The array parent itself is not returned. For example, if `user:/config/#` is an array, `user:/config` is the array parent. Only the direct array keys will be returned. This means that for example `user:/config/#1/key` will not be included, but only `user:/config/#1`.
A new keyset will be allocated for the resulting keys. This means that the caller must `ksDel` the resulting keyset.

**Parameters**

| | |
|---|---|
| *arrayParent* | the parent of the array to be returned |
| *keys* | the keyset containing the array keys |

**Returns**

a keyset containing the array keys (if any)

**Return values**

| | |
|---|---|
| *NULL* | on `NULL` pointers |

#### 573.1.2.3 elektraArrayGetNextKey()

```
Key* elektraArrayGetNextKey (
            KeySet * arrayKeys )
```

Return the next key in the given array.
The function will automatically allocate memory for a new key and name it accordingly.

**Precondition**

The supplied keyset must contain only valid array keys.

The caller has to keyDel the resulting key.

**Parameters**

| | |
|---|---|
| *arrayKeys* | the array where the new key will belong to |

**Returns**

the new array key on success

**Return values**

| | |
|---|---|
| *NULL* | if the passed array is empty |
| *NULL* | on NULL pointers or if an error occurs |

#### 573.1.2.4 elektraArrayGetPrefix()

```
char* elektraArrayGetPrefix (
            Key * key )
```

Get the base name of the passed array.
The returned value must be freed (if not null)
e.g. user:/abc/#9/foo#1 wil return user:/abc

**Parameters**

| | |
|---|---|
| *key* | |

**Return values**

| | |
|---|---|
| | |

#### 573.1.2.5 elektraArrayIncName()

```
int elektraArrayIncName (
            Key * key )
```

Increment the name of the key by one.
Alphabetical order will remain
e.g. user:/abc/#9 will be changed to user:/abc/#_10
For the start: user:/abc/# will be changed to user:/abc/#0

**Parameters**

| | |
|---|---|
| *key* | which base name will be incremented |

**Return values**

| | |
|---|---|
| *-1* | on error (e.g. array too large, non-valid array) |
| *0* | on success |

#### 573.1.2.6 elektraArrayValidateBaseNameString()

```
int elektraArrayValidateBaseNameString (
            const char * baseName )
```

validate array syntax

**Parameters**

| | |
|---|---|
| *baseName* | the supposed array element basename |

**Return values**

| | |
|---|---|
| *-1* | if no array element/syntax error/no key |
| *0* | if start |
| *offsetIndex* | otherwise, where `offsetIndex` stores the offset to the first digit of the array index of `baseName` |

#### 573.1.2.7 elektraArrayValidateName()

```
int elektraArrayValidateName (
            const Key * key )
```

validate array syntax

**Parameters**

| | |
|---|---|
| *key* | an element of an array |

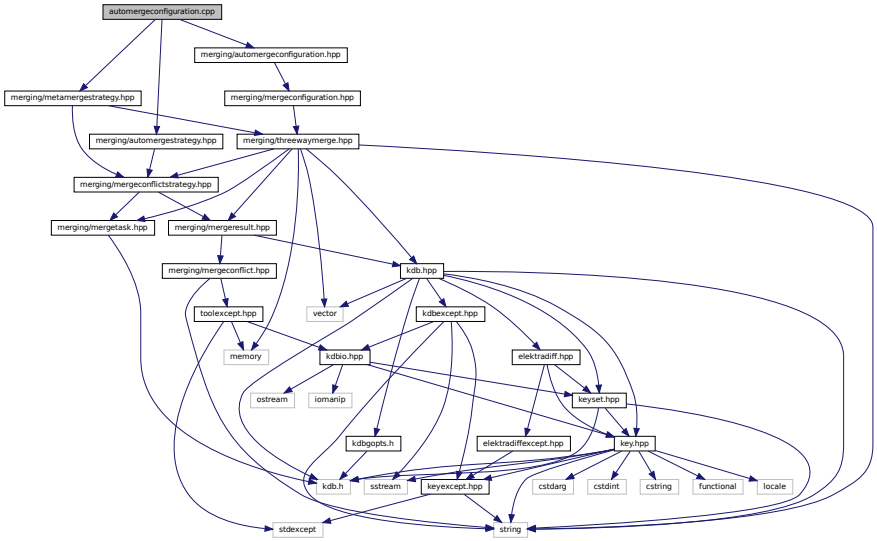
**Return values**

| | |
|---|---|
| *-1* | if no array element/syntax error/no key |
| *0* | if start |
| *1* | if array element |

## 573.2 automergeconfiguration.cpp File Reference

```
#include <merging/automergeconfiguration.hpp>
#include <merging/automergestrategy.hpp>
#include <merging/metamergestrategy.hpp>
```

Include dependency graph for automergeconfiguration.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.2.1 Detailed Description

**Copyright**

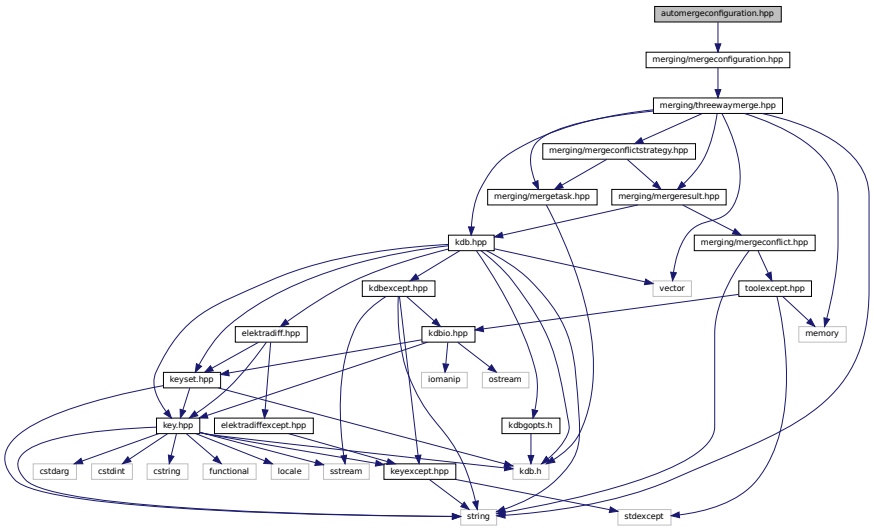
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.3 automergeconfiguration.hpp File Reference

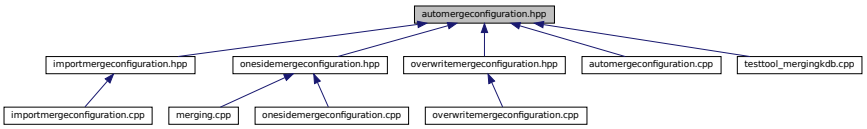
A configuration for a simple automerge.

```
#include <merging/mergeconfiguration.hpp>
```

Include dependency graph for automergeconfiguration.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.3.1 Detailed Description

A configuration for a simple automerge.

**Copyright**

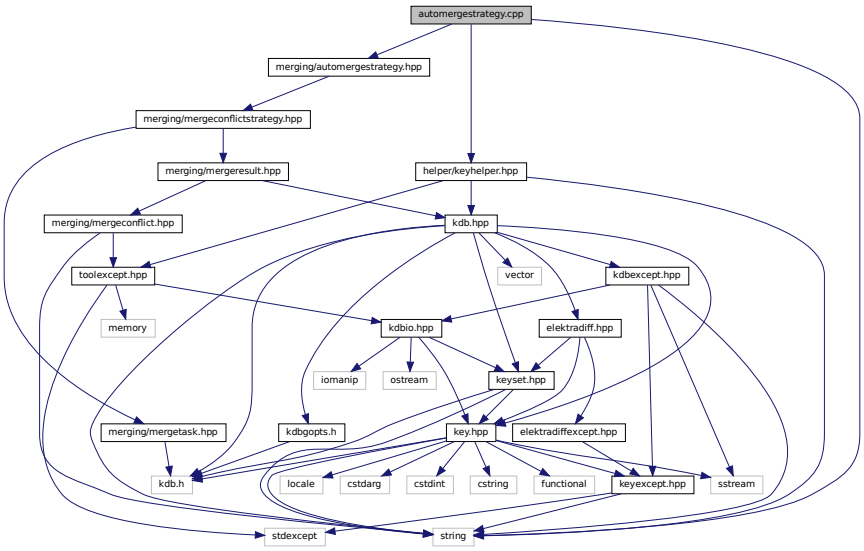
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.4 automergestrategy.cpp File Reference

Implementation of AutoMergeStrategy.

```
#include <helper/keyhelper.hpp>
#include <merging/automergestrategy.hpp>
#include <string>
```

Include dependency graph for automergestrategy.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.4.1 Detailed Description

Implementation of AutoMergeStrategy.

**Copyright**

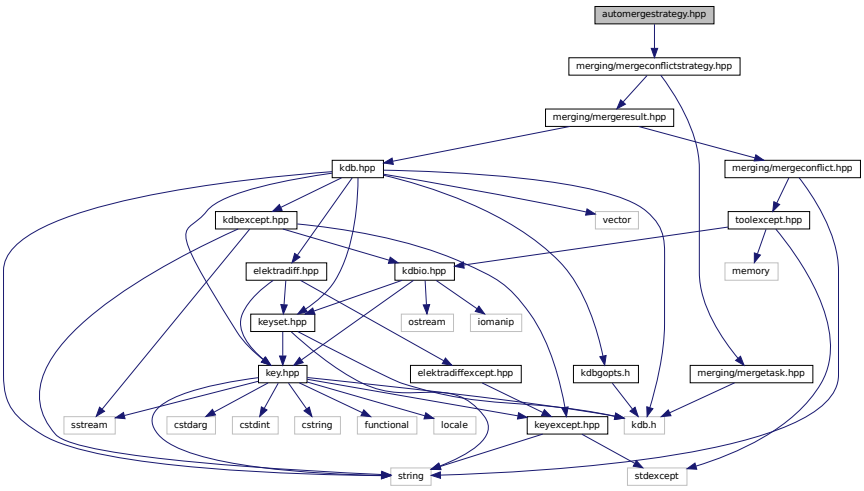
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.5 automergestrategy.hpp File Reference

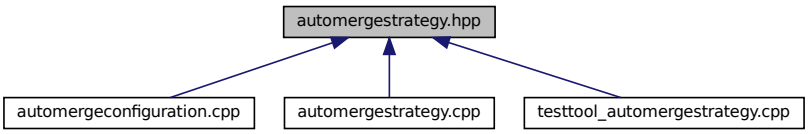
A strategy for taking the value of.

```
#include <merging/mergeconflictstrategy.hpp>
```

Include dependency graph for automergestrategy.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.5.1 Detailed Description

A strategy for taking the value of.

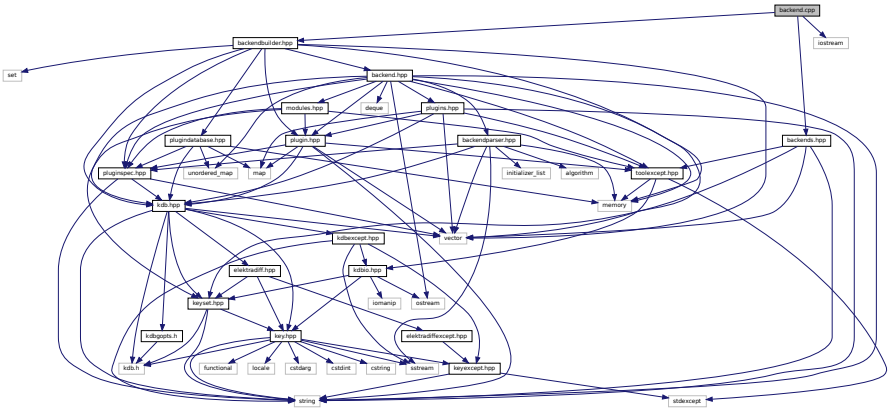
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.6 backend.cpp File Reference

```
#include <backendbuilder.hpp>
#include <backends.hpp>
#include <iostream>
```

Include dependency graph for examples/backend.cpp:




### 573.6.1 Detailed Description

**Copyright**

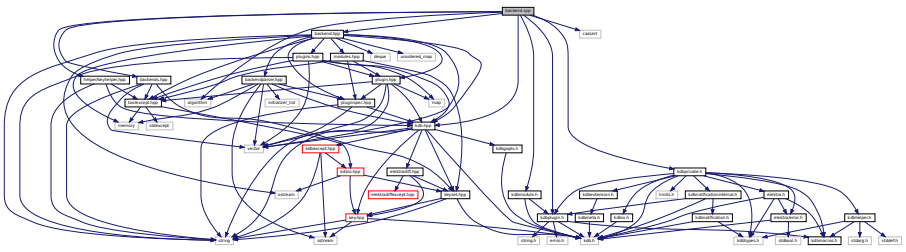
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.7 backend.cpp File Reference

Implementation of backend.

```
#include <backend.hpp>
#include <backends.hpp>
#include <helper/keyhelper.hpp>
#include <kdbmodule.h>
#include <kdbplugin.h>
#include <kdbprivate.h>
#include <kdbease.h>
#include <algorithm>
#include <cassert>
#include <kdb.hpp>
```

Include dependency graph for src/backend.cpp:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- std::ostream & kdb::tools::operator<< (std::ostream &os, Backend const &b)

  *Prints the current status.*

### 573.7.1 Detailed Description

Implementation of backend.

**Copyright**

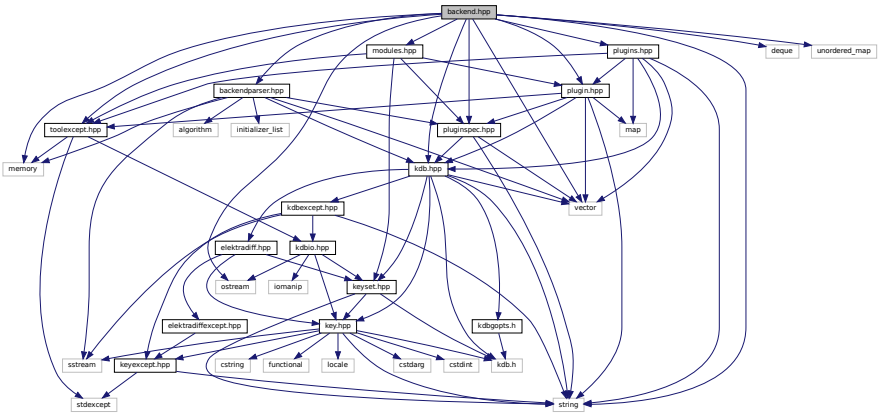
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.8 backend.hpp File Reference

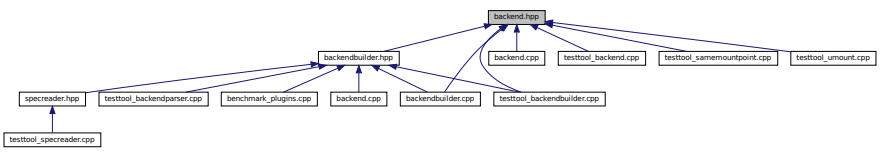
Implements a way to deal with a backend.

```
#include <backendparser.hpp>
#include <modules.hpp>
#include <plugin.hpp>
#include <plugins.hpp>
#include <pluginspec.hpp>
#include <toolexcept.hpp>
#include <deque>
#include <memory>
#include <ostream>
#include <string>
#include <unordered_map>
#include <vector>
#include <kdb.hpp>
```

Include dependency graph for backend.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- class kdb::tools::BackendInterface

  *Minimal interface to add plugins.*
- class kdb::tools::SerializeInterface

  *Interface to serialize a backend.*
- class kdb::tools::MountBackendInterface

  *Interface to work with mountpoints (backends) for factory.*
- class kdb::tools::Backend

  *A low-level representation of the backend (= set of plugins) that can be mounted.*
- class kdb::tools::BackendFactory

  *Factory for MountBackendInterface.*
- class kdb::tools::PluginAdder

  *Adds plugins in a generic map.*
- class kdb::tools::GlobalPlugins

  *Low level representation of global plugins.*
- class kdb::tools::ImportExportBackend

  *Backend for import/export functionality.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- std::ostream & kdb::tools::operator<< (std::ostream &os, Backend const &b)

  *Prints the current status.*

### 573.8.1 Detailed Description

Implements a way to deal with a backend.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

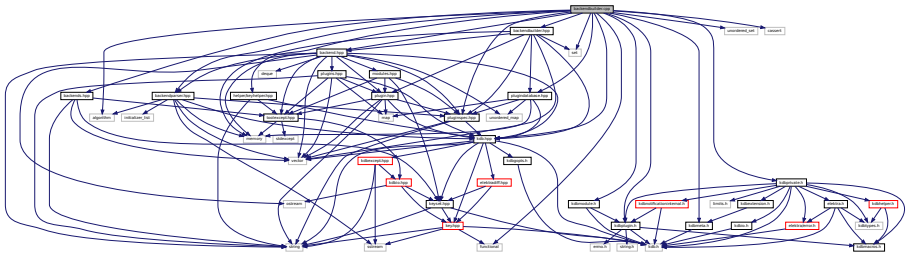
# 573.9 backendbuilder.cpp File Reference

Implementation of backend builder.

```
#include <backend.hpp>
#include <backendbuilder.hpp>
#include <backendparser.hpp>
#include <backends.hpp>
#include <plugindatabase.hpp>
#include <pluginspec.hpp>
#include <helper/keyhelper.hpp>
#include <kdbmodule.h>
#include <kdbplugin.h>
#include <kdbprivate.h>
#include <algorithm>
#include <functional>
#include <set>
```

```
#include <unordered_set>
#include <cassert>
#include <kdb.hpp>
#include <kdbmeta.h>
```

Include dependency graph for backendbuilder.cpp:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

### 573.9.1 Detailed Description

Implementation of backend builder.

**Copyright**

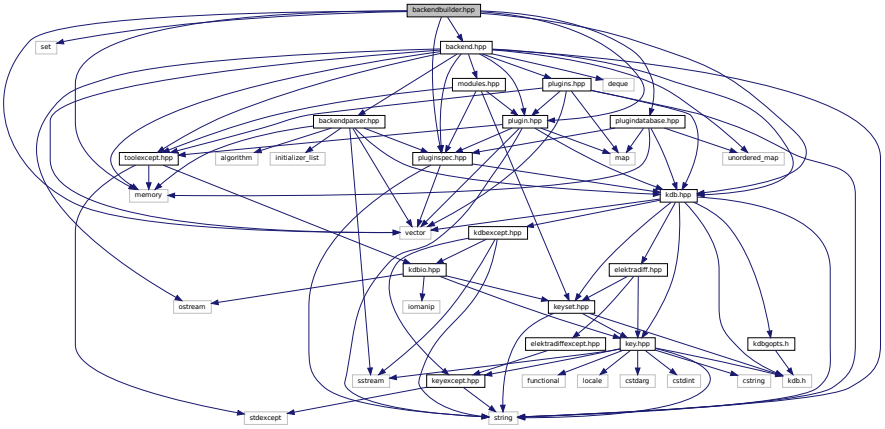
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.10 backendbuilder.hpp File Reference

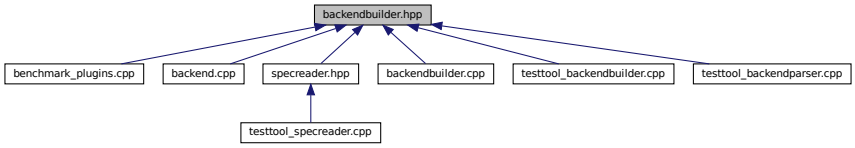
Implements a way to build backends.

```
#include <memory>
#include <set>
#include <vector>
#include <kdb.hpp>
#include <backend.hpp>
#include <plugin.hpp>
#include <plugindatabase.hpp>
#include <pluginspec.hpp>
```

Include dependency graph for backendbuilder.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::tools::BackendBuilderInit

  *Used as argument of constructor of ∗BackendBuilder.*
- class kdb::tools::BackendBuilder

  *Highlevel interface to build a backend.*
- class kdb::tools::GlobalPluginsBuilder

  *Build global plugins.*
- class kdb::tools::MountBackendBuilder

  *High-level functionality to build a mountpoint.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.10.1 Detailed Description

Implements a way to build backends.

**Copyright**

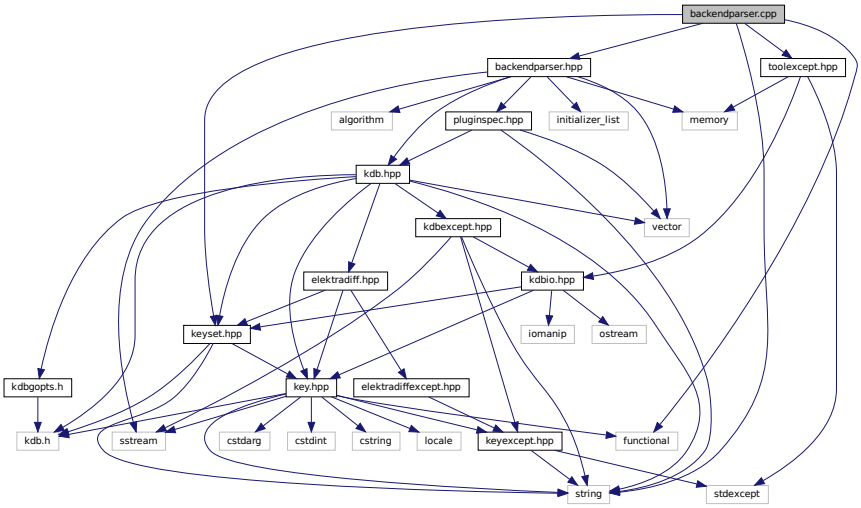
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.11 backendparser.cpp File Reference

Tests for the Backend parser class.

```
#include <backendparser.hpp>
#include <functional>
#include <string>
#include <keyset.hpp>
#include <toolexcept.hpp>
```

Include dependency graph for backendparser.cpp:




### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### Functions

- kdb::KeySet kdb::tools::parsePluginArguments (std::string const &pluginArguments, std::string const &basepath)

  *Parse a string containing information to create a KeySet.*
- PluginSpecVector kdb::tools::parseArguments (std::string const &cmdline)

  *Parse a complete commandline.*
- void kdb::tools::detail::processArgument (PluginSpecVector &arguments, size_t &counter, std::string argument)

  *Process a single argument and add it to PluginSpecVector.*
- void kdb::tools::detail::fixArguments (PluginSpecVector &arguments)

  *Fix refnames after parsing.*

### 573.11.1 Detailed Description

Tests for the Backend parser class.

**Copyright**

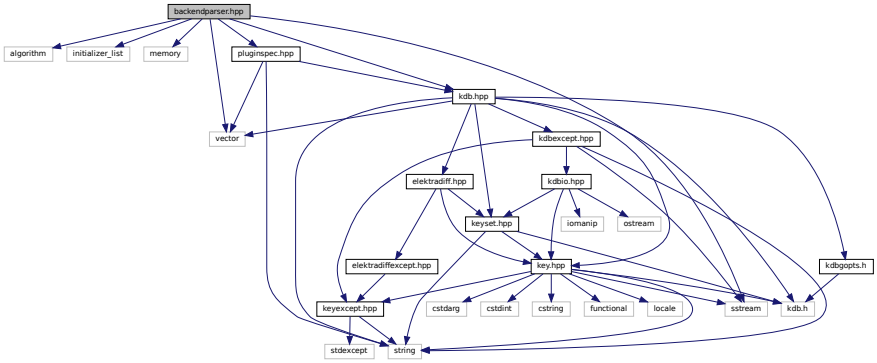
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.12 backendparser.hpp File Reference

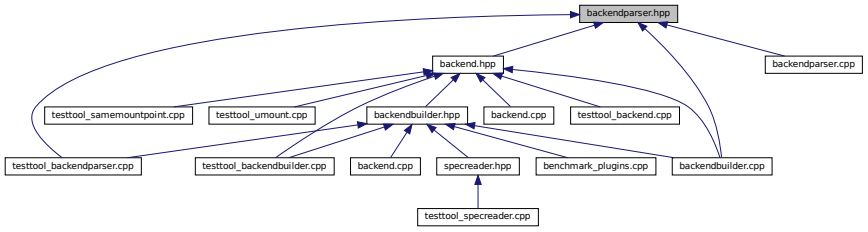
Implements ways to parse backends.

```
#include <algorithm>
#include <initializer_list>
#include <memory>
#include <sstream>
#include <vector>
#include <pluginspec.hpp>
#include <kdb.hpp>
```

Include dependency graph for backendparser.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- kdb::KeySet kdb::tools::parsePluginArguments (std::string const &pluginArguments, std::string const &basepath)

  *Parse a string containing information to create a KeySet.*
- PluginSpecVector kdb::tools::parseArguments (std::string const &cmdline)

  *Parse a complete commandline.*
- void kdb::tools::detail::processArgument (PluginSpecVector &arguments, size_t &counter, std::string argument)

  *Process a single argument and add it to PluginSpecVector.*

- void kdb::tools::detail::fixArguments (PluginSpecVector &arguments)

  *Fix refnames after parsing.*

- template<typename Iterator >
  PluginSpecVector kdb::tools::parseArguments (Iterator first, Iterator last)

  *Parse a complete commandline that is already tokenized in pluginname pluginconfig.*

### 573.12.1 Detailed Description

Implements ways to parse backends.

**Copyright**

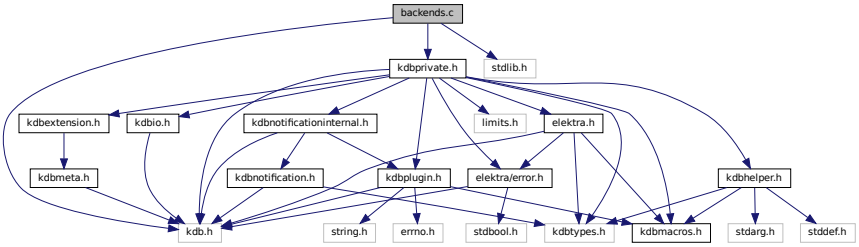
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.13 backends.c File Reference

Internal functions for handling the backends KeySet of a KDB instance.

```
#include <kdb.h>
#include <kdbprivate.h>
#include <stdlib.h>
```

Include dependency graph for backends.c:



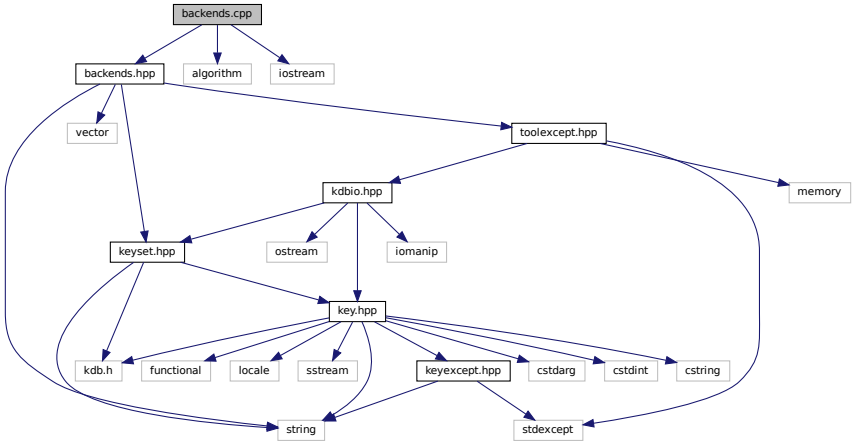

### 573.13.1 Detailed Description

Internal functions for handling the backends KeySet of a KDB instance.

# 573.14 backends.cpp File Reference

```
#include <backends.hpp>
#include <algorithm>
#include <iostream>
```

Include dependency graph for backends.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.14.1 Detailed Description

**Copyright**

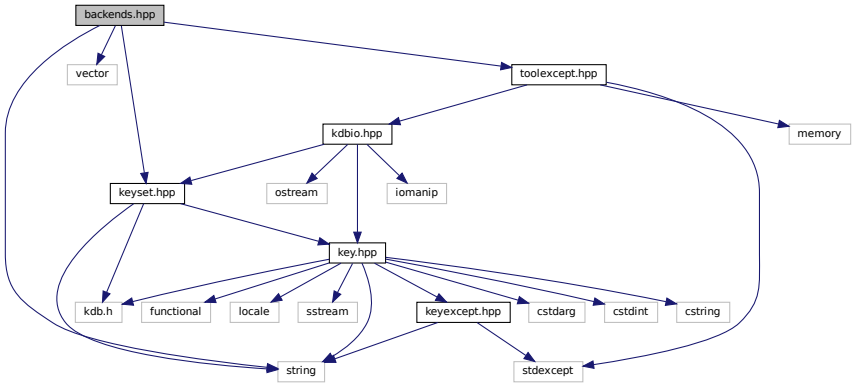


## 573.15 backends.hpp File Reference

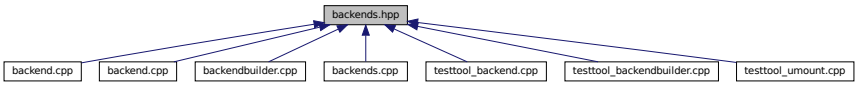
Allows one to list all available backends.

```
#include <string>
#include <vector>
#include <keyset.hpp>
#include <toolexcept.hpp>
```

Include dependency graph for backends.hpp:




This graph shows which files directly or indirectly include this file:




### Classes

- struct kdb::tools::BackendInfo

  *Info about a backend.*
- class kdb::tools::Backends

  *Allows one to list backends.*

### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.15.1 Detailed Description

Allows one to list all available backends.

**Copyright**

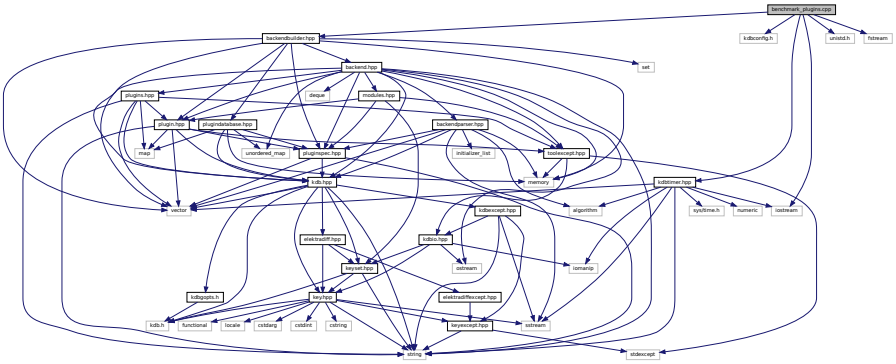
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.16 benchmark_plugins.cpp File Reference

benchmark for getenv

```
#include <backendbuilder.hpp>
#include <kdbconfig.h>
#include <kdbtimer.hpp>
#include <unistd.h>
#include <fstream>
#include <iostream>
```

Include dependency graph for benchmark_plugins.cpp:



### 573.16.1 Detailed Description

benchmark for getenv

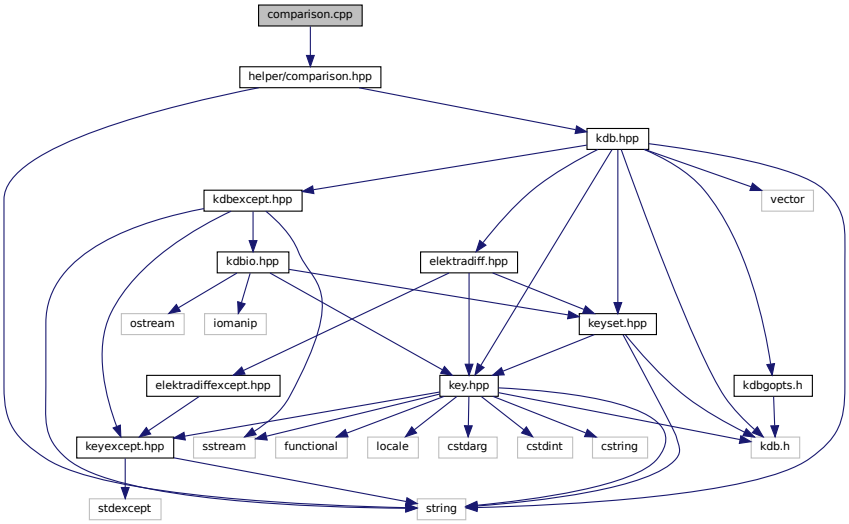
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.17 comparison.cpp File Reference

Comparison helper functions.

`#include <helper/comparison.hpp>`
Include dependency graph for comparison.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- bool kdb::tools::helper::keyDataEqual (const Key &, const Key &)

  *Determines if two keys are equal based on their string value If one of the two keys is null, false is returned.*
- bool kdb::tools::helper::keyMetaEqual (Key &, Key &)

  *Determines if two keys have equal metadata.*

## 573.17.1 Detailed Description

Comparison helper functions.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.17.2 Function Documentation

### 573.17.2.1 keyDataEqual()

```
bool kdb::tools::helper::keyDataEqual (
            const Key & k1,
            const Key & k2 )
```

Determines if two keys are equal based on their string value If one of the two keys is null, false is returned.

**Parameters**

| | |
|---|---|
| *k1* | the first key to be compared |
| *k2* | the second key to be compared |

**Returns**

true if both keys are not null and have an equal string value, false otherwise

#### 573.17.2.2 keyMetaEqual()

```
bool kdb::tools::helper::keyMetaEqual (
            Key & k1,
            Key & k2 )
```

Determines if two keys have equal metadata.
The keys are not const because their meta cursor is changed

**Parameters**

| | |
|---|---|
| *k1* | the first key whose metadata should be compared |
| *k2* | the second key whose metadata should be compared |

**Returns**

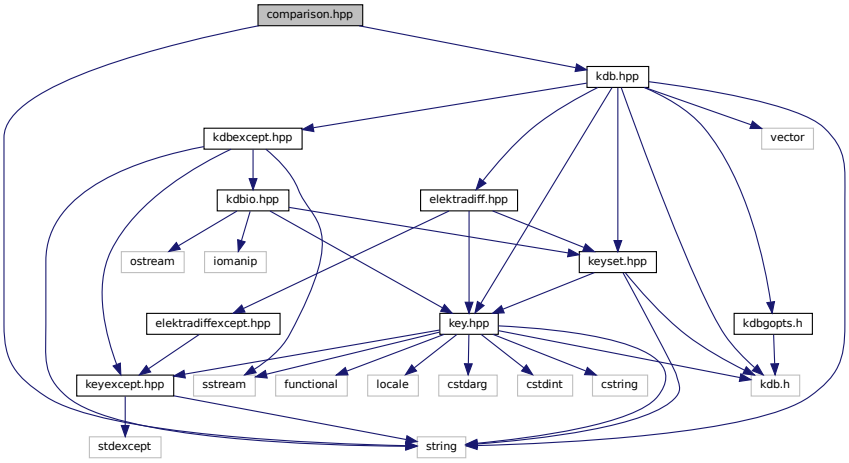
true if the keys have equal metadata, false otherwise

## 573.18 comparison.hpp File Reference

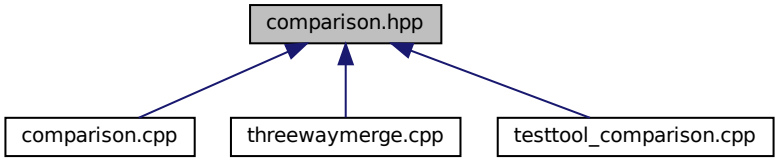
Comparison helper functions.

```
#include <kdb.hpp>
#include <string>
```

Include dependency graph for comparison.hpp:



This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- bool kdb::tools::helper::keyDataEqual (const Key &, const Key &)

  *Determines if two keys are equal based on their string value If one of the two keys is null, false is returned.*
- bool kdb::tools::helper::keyMetaEqual (Key &, Key &)

  *Determines if two keys have equal metadata.*

### 573.18.1 Detailed Description

Comparison helper functions.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.18.2 Function Documentation

#### 573.18.2.1 keyDataEqual()

```
bool kdb::tools::helper::keyDataEqual (
            const Key & k1,
            const Key & k2 )
```

Determines if two keys are equal based on their string value If one of the two keys is null, false is returned.

**Parameters**

| | |
|---|---|
| *k1* | the first key to be compared |
| *k2* | the second key to be compared |

**Returns**

true if both keys are not null and have an equal string value, false otherwise

#### 573.18.2.2 keyMetaEqual()

```
bool kdb::tools::helper::keyMetaEqual (
            Key & k1,
            Key & k2 )
```

Determines if two keys have equal metadata.
The keys are not const because their meta cursor is changed

**Parameters**

| | |
|---|---|
| *k1* | the first key whose metadata should be compared |
| *k2* | the second key whose metadata should be compared |

**Returns**

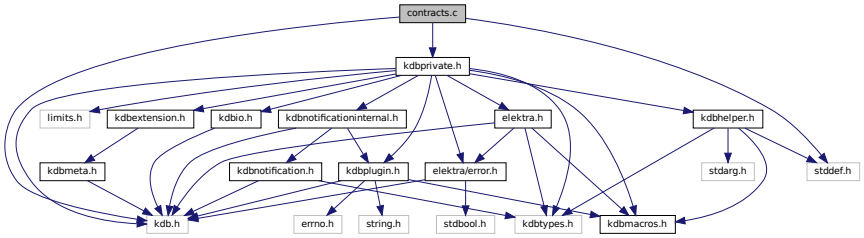
true if the keys have equal metadata, false otherwise

## 573.19 contracts.c File Reference

Contract constructors for kdbOpen()

```
#include "kdb.h"
#include "kdbprivate.h"
#include <stddef.h>
```

Include dependency graph for contracts.c:




## Functions

- int elektraGOptsContract (KeySet ∗contract, int argc, const char ∗const ∗argv, const char ∗const ∗envp, const Key ∗parentKey, KeySet ∗goptsConfig)

  *Sets up a contract for use with kdbOpen() that configures the gopts plugin.*
- int elektraGOptsContractFromStrings (KeySet ∗contract, size_t argsSize, const char ∗args, size_t envSize, const char ∗env, const Key ∗parentKey, KeySet ∗goptsConfig)

  *Sets up a contract for use with kdbOpen() that configures the gopts plugin.*

### 573.19.1 Detailed Description

Contract constructors for kdbOpen()

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.19.2 Function Documentation

#### 573.19.2.1 elektraGOptsContract()

```
int elektraGOptsContract (
            KeySet * contract,
            int argc,
            const char *const * argv,
            const char *const * envp,
            const Key * parentKey,
            KeySet * goptsConfig )
```

Sets up a contract for use with kdbOpen() that configures the gopts plugin.
You can pass 0 for `argc` **and** `NULL` for `argv` to let gopts lookup command line options internally. You can also pass `NULL` for `envp` to do the same for environment variables.

**Parameters**

| | |
|---|---|
| *contract* | The KeySet into which the contract will be written. |
| *argc* | The argc value that should be used by gopts. |
| *argv* | The argv value that should be used by gopts. IMPORTANT: The pointer and data behind must be valid until after kdbClose() is called. |
| *envp* | The envp value that should be used by gopts. IMPORTANT: The pointer and data behind must be valid until after kdbClose() is called. |
| *parentKey* | The parent key that should be used by gopts. Only the key name is copied. The key can be deleted immediately after calling this function. |
| *goptsConfig* | The config that used to mount the gopts plugin. This value can be NULL. Only keys in the user:/ namespace will be used. |

**Return values**

| | |
|---|---|
| *-1* | if `contract` or `parentKey` is NULL |
| *-1* | if `argc` is 0 and `argv` is not NULL or if `argc` is not 0 and `argv` is NULL |
| *0* | on success |

#### 573.19.2.2 elektraGOptsContractFromStrings()

```
int elektraGOptsContractFromStrings (
            KeySet * contract,
            size_t argsSize,
            const char * args,
            size_t envSize,
            const char * env,
            const Key * parentKey,
            KeySet * goptsConfig )
```

Sets up a contract for use with kdbOpen() that configures the gopts plugin.
NOTE: prefer to use elektraGOptsContract() if possible
You can pass `NULL` for `args` to let gopts lookup command line options internally. You can also pass `NULL` for `env` to do the same for environment variables.

**Parameters**

| | |
|---|---|
| *contract* | The KeySet into which the contract will be written. |
| *argsSize* | The size of the `args` data |
| *args* | Continuous buffer containing all argv arguments separated (and terminated) by zero bytes. The whole buffer is copied, so the pointer only has to be valid for this function call. |
| *envSize* | The size of the `env` data |

**Parameters**

| | |
|---|---|
| *env* | Continuous buffer containing all environment variables separated (and terminated) by zero bytes The whole buffer is copied, so the pointer only has to be valid for this function call. |
| *parentKey* | The parent key that should be used by gopts. Only the key name is copied. The key can be deleted immediately after calling this function. |
| *goptsConfig* | The config that used to mount the gopts plugin. This value can be NULL. Only keys in the user:/ namespace will be used. |

**Return values**

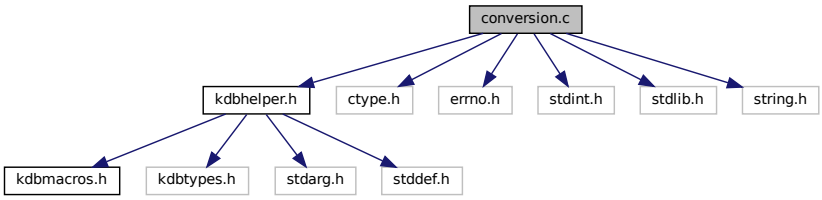
| | |
|---|---|
| *-1* | if any of `contract` or `parentKey` is NULL |
| *0* | on success |

## 573.20 conversion.c File Reference

Elektra High Level API.

```
#include "kdbease.h"
#include "kdbhelper.h"
#include <ctype.h>
#include <errno.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "macros/type_create_to_value.h"
```

Include dependency graph for conversion.c:




### Functions

- int elektraKeyToString (const Key *key, const char **variable)

  *Converts a Key to string.*
- int elektraKeyToBoolean (const Key *key, kdb_boolean_t *variable)

  *Converts a Key to boolean.*
- int elektraKeyToChar (const Key *key, kdb_char_t *variable)

  *Converts a Key to char.*
- int elektraKeyToOctet (const Key *key, kdb_octet_t *variable)

  *Converts a Key to octet.*
- int elektraKeyToShort (const Key *key, kdb_short_t *variable)

  *Converts a Key to short.*
- int elektraKeyToUnsignedShort (const Key *key, kdb_unsigned_short_t *variable)

  *Converts a Key to unsigned_short.*

- int elektraKeyToLong (const Key *key, kdb_long_t *variable)
  *Converts a Key to long.*
- int elektraKeyToUnsignedLong (const Key *key, kdb_unsigned_long_t *variable)
  *Converts a Key to unsigned_long.*
- int elektraKeyToLongLong (const Key *key, kdb_long_long_t *variable)
  *Converts a Key to long_long.*
- int elektraKeyToUnsignedLongLong (const Key *key, kdb_unsigned_long_long_t *variable)
  *Converts a Key to unsigned_long_long.*
- int elektraKeyToFloat (const Key *key, kdb_float_t *variable)
  *Converts a Key to float.*
- int elektraKeyToDouble (const Key *key, kdb_double_t *variable)
  *Converts a Key to double.*
- char * elektraBooleanToString (kdb_boolean_t value)
  *Converts a boolean to string.*
- char * elektraCharToString (kdb_char_t value)
  *Converts a char to string.*
- char * elektraOctetToString (kdb_octet_t value)
  *Converts an octet to string.*
- char * elektraShortToString (kdb_short_t value)
  *Converts a short to string.*
- char * elektraUnsignedShortToString (kdb_unsigned_short_t value)
  *Converts an unsigned short to string.*
- char * elektraLongToString (kdb_long_t value)
  *Converts a long to string.*
- char * elektraUnsignedLongToString (kdb_unsigned_long_t value)
  *Converts an unsigned long to string.*
- char * elektraLongLongToString (kdb_long_long_t value)
  *Converts a long long to string.*
- char * elektraUnsignedLongLongToString (kdb_unsigned_long_long_t value)
  *Converts an unsigned long long to string.*
- char * elektraFloatToString (kdb_float_t value)
  *Converts a float to string.*
- char * elektraDoubleToString (kdb_double_t value)
  *Converts a double to string.*

### 573.20.1 Detailed Description

Elektra High Level API.

**Copyright**

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

### 573.20.2 Function Documentation

#### 573.20.2.1 elektraBooleanToString()

```
char* elektraBooleanToString (
            kdb_boolean_t value )
```

Converts a boolean to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the boolean to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.2 elektraCharToString()

```
char* elektraCharToString (
            kdb_char_t value )
```

Converts a char to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.3 elektraDoubleToString()

```
char* elektraDoubleToString (
            kdb_double_t value )
```

Converts a double to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.4 elektraFloatToString()

```
char* elektraFloatToString (
            kdb_float_t value )
```

Converts a float to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.5 elektraKeyToBoolean()

```
int elektraKeyToBoolean (
            const Key * key,
            kdb_boolean_t * variable )
```

Converts a Key to boolean.
The value "1" is regarded as true, anything is as false.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.6 elektraKeyToChar()

```
int elektraKeyToChar (
            const Key * key,
            kdb_char_t * variable )
```

Converts a Key to char.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.7 elektraKeyToDouble()

```
int elektraKeyToDouble (
            const Key * key,
            kdb_double_t * variable )
```

Converts a Key to double.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

### 573.20.2.8 elektraKeyToFloat()

```
int elektraKeyToFloat (
            const Key * key,
            kdb_float_t * variable )
```

Converts a Key to float.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

### 573.20.2.9 elektraKeyToLong()

```
int elektraKeyToLong (
            const Key * key,
            kdb_long_t * variable )
```

Converts a Key to long.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

### 573.20.2.10 elektraKeyToLongLong()

```
int elektraKeyToLongLong (
            const Key * key,
            kdb_long_long_t * variable )
```

Converts a Key to long_long.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.11 elektraKeyToOctet()

```
int elektraKeyToOctet (
            const Key * key,
            kdb_octet_t * variable )
```

Converts a Key to octet.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.12 elektraKeyToShort()

```
int elektraKeyToShort (
            const Key * key,
            kdb_short_t * variable )
```

Converts a Key to short.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.13 elektraKeyToString()

```
int elektraKeyToString (
            const Key * key,
            const char ** variable )
```

Converts a Key to string.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.14 elektraKeyToUnsignedLong()

```
int elektraKeyToUnsignedLong (
            const Key * key,
            kdb_unsigned_long_t * variable )
```

Converts a Key to unsigned_long.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.15 elektraKeyToUnsignedLongLong()

```
int elektraKeyToUnsignedLongLong (
            const Key * key,
            kdb_unsigned_long_long_t * variable )
```

Converts a Key to unsigned_long_long.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.16 elektraKeyToUnsignedShort()

```
int elektraKeyToUnsignedShort (
            const Key * key,
            kdb_unsigned_short_t * variable )
```

Converts a Key to unsigned_short.
The variable pointed to by `variable` is unchanged, if an error occurs.

**Parameters**

| | |
|---|---|
| *key* | the key to convert |
| *variable* | pointer to the output variable |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | otherwise |

#### 573.20.2.17 elektraLongLongToString()

```
char* elektraLongLongToString (
            kdb_long_long_t value )
```

Converts a long long to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.18 elektraLongToString()

```
char* elektraLongToString (
            kdb_long_t value )
```

Converts a long to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.19 elektraOctetToString()

```
char* elektraOctetToString (
            kdb_octet_t value )
```

Converts an octet to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.20 elektraShortToString()

```
char* elektraShortToString (
            kdb_short_t value )
```

Converts a short to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.21 elektraUnsignedLongLongToString()

```
char* elektraUnsignedLongLongToString (
            kdb_unsigned_long_long_t value )
```

Converts an unsigned long long to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.22 elektraUnsignedLongToString()

```
char* elektraUnsignedLongToString (
            kdb_unsigned_long_t value )
```

Converts an unsigned long to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

#### 573.20.2.23 elektraUnsignedShortToString()

```
char* elektraUnsignedShortToString (
            kdb_unsigned_short_t value )
```

Converts an unsigned short to string.
The string is allocated with elektraMalloc() and must be disposed of with elektraFree().

**Parameters**

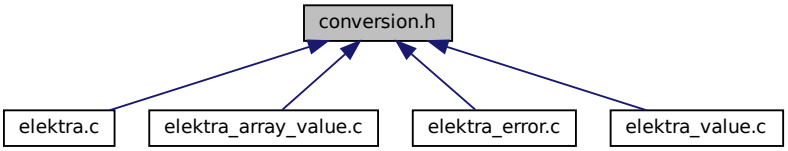
| | |
|---|---|
| *value* | the value to convert |

**Returns**

a new string allocated with elektraMalloc, or 0 on error

## 573.21 conversion.h File Reference

Elektra conversion.
`#include "kdbease.h"`
This graph shows which files directly or indirectly include this file:




### 573.21.1 Detailed Description

Elektra conversion.

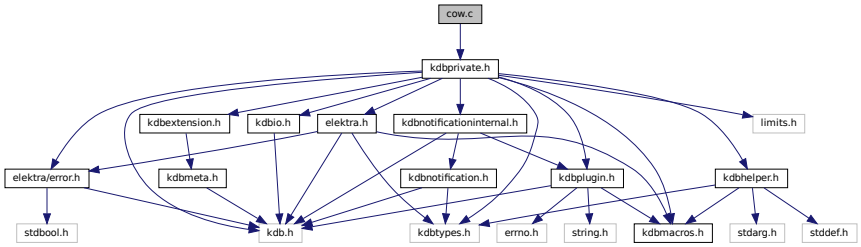
**Copyright**

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

## 573.22 cow.c File Reference

Shared methods for key and keyset copy-on-write.

```
#include <kdbprivate.h>
```
Include dependency graph for cow.c:




### 573.22.1 Detailed Description

Shared methods for key and keyset copy-on-write.

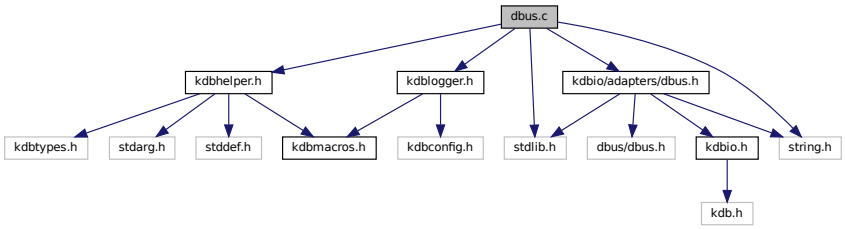
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.23 dbus.c File Reference

I/O Adapter for D-Bus.

```
#include <kdbhelper.h>
#include <kdbio/adapters/dbus.h>
#include <kdblogger.h>
#include <stdlib.h>
#include <string.h>
```
Include dependency graph for dbus.c:




### Functions

- ElektraIoAdapterDbusHandle * elektraIoAdapterDbusAttach (DBusConnection *connection, ElektraIoInterface *ioBinding)

  *Attach D-Bus connection to asynchronous I/O binding.*
- int elektraIoAdapterDbusCleanup (ElektraIoAdapterDbusHandle *priv)

  *Remove D-Bus connection from I/O binding.*

### 573.23.1 Detailed Description

I/O Adapter for D-Bus.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.23.2 Function Documentation

#### 573.23.2.1 elektraIoAdapterDbusAttach()

```
ElektraIoAdapterDbusHandle* elektraIoAdapterDbusAttach (
            DBusConnection * connection,
            ElektraIoInterface * ioBinding )
```

Attach D-Bus connection to asynchronous I/O binding.
Messages are sent and received using the I/O binding

**Parameters**

| | |
|---|---|
| *connection* | D-Bus connection |
| *ioBinding* | I/O binding |

**Returns**

handle to be used with elektraIoAdapterDbusCleanup() or NULL on error

#### 573.23.2.2 elektraIoAdapterDbusCleanup()

```
int elektraIoAdapterDbusCleanup (
            ElektraIoAdapterDbusHandle * handle )
```

Remove D-Bus connection from I/O binding.
This function frees the passed handle.
Currently it is NOT possible to revert all changes made to a connection by elektraIoAdapterDbusAttach(). It is advisable to call dbus_connection_unref() or dbus_connection_close() in case of a private bus connection afterwards.

**Parameters**

| | |
|---|---|
| *handle* | adapter handle |

**Return values**

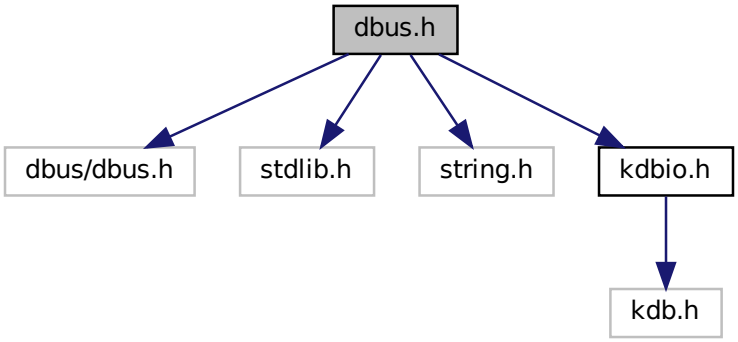
| | |
|---|---|
| *1* | on success |
| *0* | on error |

## 573.24 dbus.h File Reference

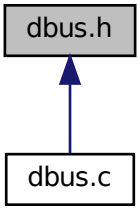
I/O Adapter for D-Bus.

```
#include <dbus/dbus.h>
#include <stdlib.h>
#include <string.h>
#include <kdbio.h>
```

Include dependency graph for dbus.h:




This graph shows which files directly or indirectly include this file:




## Typedefs

- typedef struct _ElektraIoAdapterDbusHandle ElektraIoAdapterDbusHandle

  *D-Bus Adapter Handle.*

## Functions

- ElektraIoAdapterDbusHandle ∗ elektraIoAdapterDbusAttach (DBusConnection ∗connection, ElektraIoInterface ∗ioBinding)

  *Attach D-Bus connection to asynchronous I/O binding.*
- int elektraIoAdapterDbusCleanup (ElektraIoAdapterDbusHandle ∗handle)

  *Remove D-Bus connection from I/O binding.*

### 573.24.1 Detailed Description

I/O Adapter for D-Bus.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.24.2 Typedef Documentation

#### 573.24.2.1 ElektraIoAdapterDbusHandle

```
typedef struct _ElektraIoAdapterDbusHandle ElektraIoAdapterDbusHandle
```

D-Bus Adapter Handle.
Returned by elektraIoAdapterDbusAttach().

### 573.24.3 Function Documentation

#### 573.24.3.1 elektraIoAdapterDbusAttach()

```
ElektraIoAdapterDbusHandle* elektraIoAdapterDbusAttach (
            DBusConnection * connection,
            ElektraIoInterface * ioBinding )
```

Attach D-Bus connection to asynchronous I/O binding.
Messages are sent and received using the I/O binding

**Parameters**

| | |
|---|---|
| *connection* | D-Bus connection |
| *ioBinding* | I/O binding |

**Returns**

handle to be used with elektraIoAdapterDbusCleanup() or NULL on error

#### 573.24.3.2 elektraIoAdapterDbusCleanup()

```
int elektraIoAdapterDbusCleanup (
            ElektraIoAdapterDbusHandle * handle )
```

Remove D-Bus connection from I/O binding.
This function frees the passed handle.
Currently it is NOT possible to revert all changes made to a connection by elektraIoAdapterDbusAttach(). It is advisable to call dbus_connection_unref() or dbus_connection_close() in case of a private bus connection afterwards.

**Parameters**

| | |
|---|---|
| *handle* | adapter handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

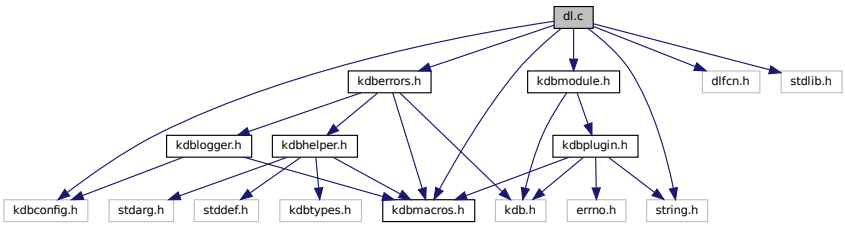
## 573.25 dl.c File Reference

Loading modules under linux.

```
#include <kdbconfig.h>
#include <kdbmacros.h>
```

```
#include <dlfcn.h>
#include <kdberrors.h>
#include <kdbmodule.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for dl.c:




### 573.25.1 Detailed Description

Loading modules under linux.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

The name of the module will be libname. A .so will be appended. This file will be loaded.
The path were the plugins are located, e.g. /usr/src/elektra need to be added with LD_LIBRARY_PATH.
The reason is that only LD_LIBRARY_PATH also loads libraries which are seen by symlink only. That feature is needed for libelektra-default.
The buildsystem makes sure that dlfcn.h exists.

## 573.26 doc.c File Reference

Loading Modules for Elektra documentation.

### Functions

- int elektraModulesInit (KeySet *modules, Key *error)

  *Initialises the module loading system.*
- elektraPluginFactory elektraModulesLoad (KeySet *modules, const char *name, Key *error)

  *Load a library with the given name.*
- int elektraModulesClose (KeySet *modules, Key *error)

  *Close all modules.*

### 573.26.1 Detailed Description

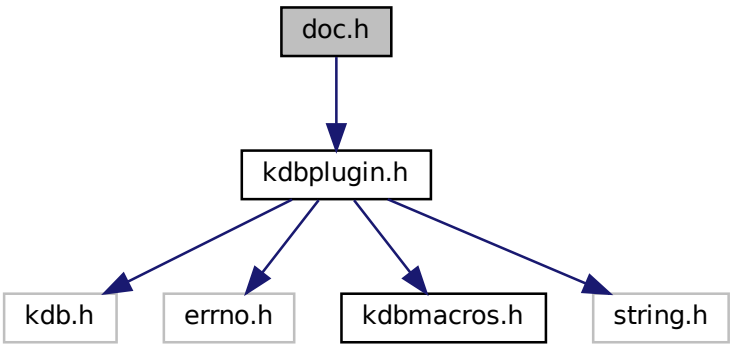
Loading Modules for Elektra documentation.

**Copyright**



## 573.27 doc.h File Reference

`#include <kdbplugin.h>`
Include dependency graph for doc.h:




### Macros

- #define ELEKTRA_SET_ERROR(number, key, text)
  *Sets the error in the keys metadata.*
- #define ELEKTRA_SET_ERRORF(number, key, formatstring, ...)
  *Sets the error in the keys metadata.*
- #define ELEKTRA_ADD_WARNINGF(number, key, formatstring, ...)
  *Adds a warning in the keys metadata.*
- #define ELEKTRA_ADD_WARNING(number, key, text)
  *Adds a warning in the keys metadata.*
- #define ELEKTRA_SET_ERROR_GET(parentKey)
  *Set error in kdbGet() when opening the file failed.*
- #define ELEKTRA_SET_ERROR_SET(parentKey)
  *Set error in kdbSet() when opening the file failed.*

### Functions

- int elektraDocOpen (Plugin *handle, Key *warningsKey)
  *Initialize data for the plugin.*
- int elektraDocClose (Plugin *handle, Key *warningsKey)
  *Finalize the plugin.*
- int elektraDocGet (Plugin *handle, KeySet *returned, Key *parentKey)
  *Get data from storage to application.*
- int elektraDocSet (Plugin *handle, KeySet *returned, Key *parentKey)
  *Set data from application to storage.*
- int elektraDocCommit (Plugin *handle, KeySet *returned, Key *parentKey)
  *Make changes to storage final.*

- int elektraDocError (Plugin *handle, KeySet *returned, Key *parentKey)

  *Rollback in case of errors.*
- int elektraDocCheckConf (Key *errorKey, KeySet *conf)

  *Validate plugin configuration at mount time.*

### 573.27.1 Detailed Description

**Copyright**

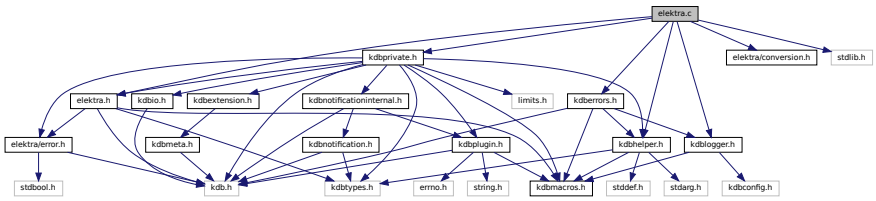
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.28 elektra.c File Reference

Elektra High Level API.

```
#include "elektra.h"
#include "elektra/conversion.h"
#include "elektra/types.h"
#include "kdberrors.h"
#include "kdbhelper.h"
#include "kdblogger.h"
#include "kdbprivate.h"
#include <stdlib.h>
```

Include dependency graph for elektra.c:




## Functions

- kdb_boolean_t checkSpec (Key *const parentKey, KeySet *contract, ElektraError **error)

  *Verify that specification is properly mounted and is equal to specification at compile time.*
- Elektra * elektraOpen (const char *application, KeySet *defaults, KeySet *contract, ElektraError **error)

  *Initializes a new Elektra instance.*
- void elektraFatalError (Elektra *elektra, ElektraError *fatalError)

  *Promote an ElektraError to fatal and call the fatal error handler.*
- Key * elektraHelpKey (Elektra *elektra)

  *This function is only intended for use with code-generation.*
- void elektraFatalErrorHandler (Elektra *elektra, ElektraErrorHandler fatalErrorHandler)

  *Sets the fatal error handler that will be called, whenever a fatal error occurs.*
- void elektraClose (Elektra *elektra)

  *Releases all resources used by the given elektra instance.*

### 573.28.1 Detailed Description

Elektra High Level API.

**Copyright**

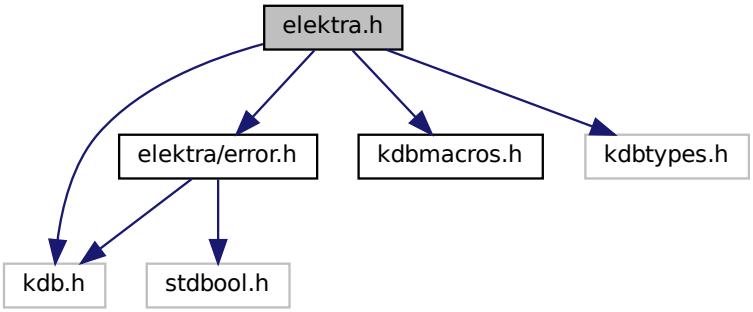
BSD License (see doc/LICENSE.md or http://www.libelektra.org)

## 573.29 elektra.h File Reference

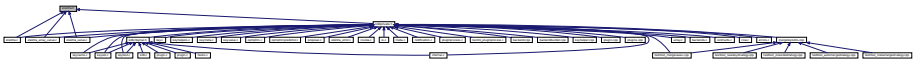
Elektra High Level API.

```
#include <elektra/error.h>
#include <elektra/types.h>
#include <kdb.h>
#include <kdbmacros.h>
#include <kdbtypes.h>
```

Include dependency graph for elektra.h:




This graph shows which files directly or indirectly include this file:



### Functions

- Elektra * elektraOpen (const char *application, KeySet *defaults, KeySet *contract, ElektraError **error)

  *Initializes a new Elektra instance.*
- void elektraClose (Elektra *elektra)

  *Releases all resources used by the given elektra instance.*
- void elektraFatalErrorHandler (Elektra *elektra, ElektraErrorHandler fatalErrorHandler)

  *Sets the fatal error handler that will be called, whenever a fatal error occurs.*
- Key * elektraFindKey (Elektra *elektra, const char *name, KDBType type)

  *Helper function for code generation.*
- Key * elektraFindArrayElementKey (Elektra *elektra, const char *name, kdb_long_long_t index, KDBType type)

  *Helper function for code generation.*
- void elektraFatalError (Elektra *elektra, ElektraError *fatalError)

  *Promote an ElektraError to fatal and call the fatal error handler.*
- const char * elektraFindReference (Elektra *elektra, const char *name)

  *Resolves the reference stored in a key.*
- const char * elektraFindReferenceArrayElement (Elektra *elektra, const char *name, kdb_long_long_t index)

  *Resolves the reference stored in a key.*
- Key * elektraHelpKey (Elektra *elektra)

  *This function is only intended for use with code-generation.*
- const char * elektraGetRawString (Elektra *elektra, const char *name)

*Get the raw string value of a key.*

- const char * elektraGetString (Elektra *elektra, const char *keyname)

  *Gets a string value.*
- kdb_boolean_t elektraGetBoolean (Elektra *elektra, const char *keyname)

  *Gets a boolean value.*
- kdb_char_t elektraGetChar (Elektra *elektra, const char *keyname)

  *Gets a char value.*
- kdb_octet_t elektraGetOctet (Elektra *elektra, const char *keyname)

  *Gets an octet value.*
- kdb_short_t elektraGetShort (Elektra *elektra, const char *keyname)

  *Gets a short value.*
- kdb_unsigned_short_t elektraGetUnsignedShort (Elektra *elektra, const char *keyname)

  *Gets a unsigned short value.*
- kdb_long_t elektraGetLong (Elektra *elektra, const char *keyname)

  *Gets a long value.*
- kdb_unsigned_long_t elektraGetUnsignedLong (Elektra *elektra, const char *keyname)

  *Gets a unsigned long value.*
- kdb_long_long_t elektraGetLongLong (Elektra *elektra, const char *keyname)

  *Gets a long long value.*
- kdb_unsigned_long_long_t elektraGetUnsignedLongLong (Elektra *elektra, const char *keyname)

  *Gets a long long value.*
- kdb_float_t elektraGetFloat (Elektra *elektra, const char *keyname)

  *Gets a float value.*
- kdb_double_t elektraGetDouble (Elektra *elektra, const char *keyname)

  *Gets a double value.*
- void elektraSetRawString (Elektra *elektra, const char *name, const char *value, KDBType type, ElektraError **error)

  *Set the raw string value of a key.*
- void elektraSetString (Elektra *elektra, const char *keyname, const char *value, ElektraError **error)

  *Sets a string value.*
- void elektraSetBoolean (Elektra *elektra, const char *keyname, kdb_boolean_t value, ElektraError **error)

  *Sets a boolean value.*
- void elektraSetChar (Elektra *elektra, const char *keyname, kdb_char_t value, ElektraError **error)

  *Sets a char value.*
- void elektraSetOctet (Elektra *elektra, const char *keyname, kdb_octet_t value, ElektraError **error)

  *Sets an octet value.*
- void elektraSetShort (Elektra *elektra, const char *keyname, kdb_short_t value, ElektraError **error)

  *Sets a short value.*
- void elektraSetUnsignedShort (Elektra *elektra, const char *keyname, kdb_unsigned_short_t value, ElektraError **error)

  *Sets a unsigned short value.*
- void elektraSetLong (Elektra *elektra, const char *keyname, kdb_long_t value, ElektraError **error)

  *Sets a long value.*
- void elektraSetUnsignedLong (Elektra *elektra, const char *keyname, kdb_unsigned_long_t value, ElektraError **error)

  *Sets a unsigned long value.*
- void elektraSetLongLong (Elektra *elektra, const char *keyname, kdb_long_long_t value, ElektraError **error)

  *Sets a long long value.*
- void elektraSetUnsignedLongLong (Elektra *elektra, const char *keyname, kdb_unsigned_long_long_t value, ElektraError **error)

*Sets a unsigned long long value.*

- void elektraSetFloat (Elektra *elektra, const char *keyname, kdb_float_t value, ElektraError **error)

  *Sets a float value.*
- void elektraSetDouble (Elektra *elektra, const char *keyname, kdb_double_t value, ElektraError **error)

  *Sets a double value.*
- kdb_long_long_t elektraArraySize (Elektra *elektra, const char *name)

  *Gets the size of an array.*
- const char * elektraGetRawStringArrayElement (Elektra *elektra, const char *name, kdb_long_long_t index)

  *Get the raw string value of an array element key.*
- const char * elektraGetStringArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a string value array element.*
- kdb_boolean_t elektraGetBooleanArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a boolean value array element.*
- kdb_char_t elektraGetCharArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a char value array element.*
- kdb_octet_t elektraGetOctetArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets an octet value array element.*
- kdb_short_t elektraGetShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a short value array element.*
- kdb_unsigned_short_t elektraGetUnsignedShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a unsigned short value array element.*
- kdb_long_t elektraGetLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a long value array element.*
- kdb_unsigned_long_t elektraGetUnsignedLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a unsigned long value array element.*
- kdb_long_long_t elektraGetLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a long long value array element.*
- kdb_unsigned_long_long_t elektraGetUnsignedLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a unsigned long long value array element.*
- kdb_float_t elektraGetFloatArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a float value array element.*
- kdb_double_t elektraGetDoubleArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a double value array element.*
- void elektraSetRawStringArrayElement (Elektra *elektra, const char *name, kdb_long_long_t index, const char *value, KDBType type, ElektraError **error)

  *Set the raw string value of an array element key.*
- void elektraSetStringArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, const char *value, ElektraError **error)

  *Sets a string value array element.*
- void elektraSetBooleanArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_boolean_t value, ElektraError **error)

  *Sets a boolean value array element.*
- void elektraSetCharArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_char_t value, ElektraError **error)

  *Sets a char value array element.*

- void elektraSetOctetArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_↩ octet_t value, ElektraError ∗∗error)

  *Sets an octet value array element.*
- void elektraSetShortArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_↩ short_t value, ElektraError ∗∗error)

  *Sets a short value array element.*
- void elektraSetUnsignedShortArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_unsigned_short_t value, ElektraError ∗∗error)

  *Sets a unsigned short value array element.*
- void elektraSetLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_long↩ _t value, ElektraError ∗∗error)

  *Sets a long value array element.*
- void elektraSetUnsignedLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_unsigned_long_t value, ElektraError ∗∗error)

  *Sets a unsigned long value array element.*
- void elektraSetLongLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb↩ _long_long_t value, ElektraError ∗∗error)

  *Sets a long long value array element.*
- void elektraSetUnsignedLongLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_unsigned_long_long_t value, ElektraError ∗∗error)

  *Sets a unsigned long long value array element.*
- void elektraSetFloatArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_float↩ _t value, ElektraError ∗∗error)

  *Sets a float value array element.*
- void elektraSetDoubleArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index, kdb_↩ double_t value, ElektraError ∗∗error)

  *Sets a double value array element.*
- KDBType elektraGetType (Elektra ∗elektra, const char ∗keyname)

  *Reads the type metadata of a given key.*
- KDBType elektraGetArrayElementType (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Reads the type metadata of a given array element.*

### 573.29.1 Detailed Description

Elektra High Level API.

**Copyright**

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

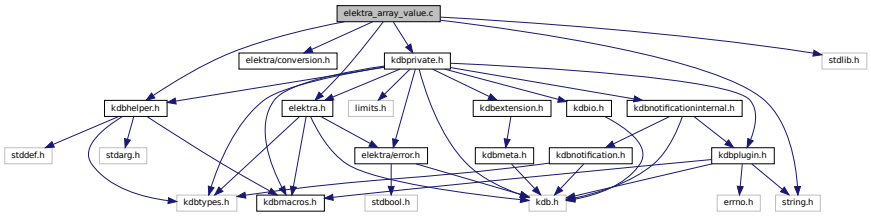
## 573.30 elektra_array_value.c File Reference

Elektra High Level API.

```
#include "elektra.h"
#include "elektra/conversion.h"
#include "kdbease.h"
#include "kdbhelper.h"
#include "kdbprivate.h"
#include <stdlib.h>
```

```
#include <string.h>
```

Include dependency graph for elektra_array_value.c:




## Functions

- kdb_long_long_t elektraArraySize (Elektra ∗elektra, const char ∗name)

  *Gets the size of an array.*
- Key ∗ elektraFindArrayElementKey (Elektra ∗elektra, const char ∗name, kdb_long_long_t index, KDBType type)

  *Helper function for code generation.*
- const char ∗ elektraFindReferenceArrayElement (Elektra ∗elektra, const char ∗name, kdb_long_long_t index)

  *Resolves the reference stored in a key.*
- KDBType elektraGetArrayElementType (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Reads the type metadata of a given array element.*
- const char ∗ elektraGetRawStringArrayElement (Elektra ∗elektra, const char ∗name, kdb_long_long_t index)

  *Get the raw string value of an array element key.*
- void elektraSetRawStringArrayElement (Elektra ∗elektra, const char ∗name, kdb_long_long_t index, const char ∗value, KDBType type, ElektraError ∗∗error)

  *Set the raw string value of an array element key.*
- const char ∗ elektraGetStringArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets a string value array element.*
- kdb_boolean_t elektraGetBooleanArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_↩ t index)

  *Gets a boolean value array element.*
- kdb_char_t elektraGetCharArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets a char value array element.*
- kdb_octet_t elektraGetOctetArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets an octet value array element.*
- kdb_short_t elektraGetShortArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets a short value array element.*
- kdb_unsigned_short_t elektraGetUnsignedShortArrayElement (Elektra ∗elektra, const char ∗keyname, kdb↩ _long_long_t index)

  *Gets a unsigned short value array element.*
- kdb_long_t elektraGetLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

  *Gets a long value array element.*
- kdb_unsigned_long_t elektraGetUnsignedLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_↩ long_long_t index)

  *Gets a unsigned long value array element.*
- kdb_long_long_t elektraGetLongLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long↩ _t index)

  *Gets a long long value array element.*
- kdb_unsigned_long_long_t elektraGetUnsignedLongLongArrayElement (Elektra ∗elektra, const char ∗keyname, kdb_long_long_t index)

*Gets a unsigned long long value array element.*

- kdb_float_t elektraGetFloatArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a float value array element.*
- kdb_double_t elektraGetDoubleArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index)

  *Gets a double value array element.*
- void elektraSetStringArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, const char *value, ElektraError **error)

  *Sets a string value array element.*
- void elektraSetBooleanArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_boolean_t value, ElektraError **error)

  *Sets a boolean value array element.*
- void elektraSetCharArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_char_t value, ElektraError **error)

  *Sets a char value array element.*
- void elektraSetOctetArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_octet_t value, ElektraError **error)

  *Sets an octet value array element.*
- void elektraSetShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_short_t value, ElektraError **error)

  *Sets a short value array element.*
- void elektraSetUnsignedShortArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_unsigned_short_t value, ElektraError **error)

  *Sets a unsigned short value array element.*
- void elektraSetLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_long_t value, ElektraError **error)

  *Sets a long value array element.*
- void elektraSetUnsignedLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_unsigned_long_t value, ElektraError **error)

  *Sets a unsigned long value array element.*
- void elektraSetLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_long_long_t value, ElektraError **error)

  *Sets a long long value array element.*
- void elektraSetUnsignedLongLongArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_unsigned_long_long_t value, ElektraError **error)

  *Sets a unsigned long long value array element.*
- void elektraSetFloatArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_float_t value, ElektraError **error)

  *Sets a float value array element.*
- void elektraSetDoubleArrayElement (Elektra *elektra, const char *keyname, kdb_long_long_t index, kdb_double_t value, ElektraError **error)

  *Sets a double value array element.*

### 573.30.1 Detailed Description

Elektra High Level API.

**Copyright**

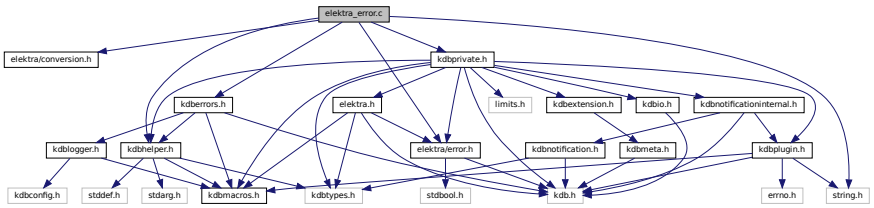
BSD License (see doc/LICENSE.md or http://www.libelektra.org)

## 573.31 elektra_error.c File Reference

The error module of the High level API.

```
#include "elektra/conversion.h"
#include "elektra/error.h"
#include "kdberrors.h"
#include "kdbhelper.h"
#include "kdbprivate.h"
#include <string.h>
```

Include dependency graph for elektra_error.c:




### Functions

- ElektraError ∗ elektraErrorCreate (const char ∗code, const char ∗description, const char ∗module, const char ∗file, kdb_long_t line)

  *Creates a new ElektraError using the provided values.*
- void elektraErrorAddWarning (ElektraError ∗error, ElektraError ∗warning)

  *Adds a warning to an existing ElektraError struct.*
- ElektraError ∗ elektraErrorFromKey (Key ∗key)

  *Extracts the error and all warnings from the given key.*
- ElektraError ∗ elektraErrorKeyNotFound (const char ∗keyname)

  *Creates a "Key not found" error.*
- ElektraError ∗ elektraErrorWrongType (const char ∗keyname, KDBType expectedType, KDBType actualType)

  *Creates a "Wrong type" error.*
- ElektraError ∗ elektraErrorNullError (const char ∗function)

  *Creates a "Null error argument" error.*
- ElektraError ∗ elektraErrorConversionToString (KDBType sourceType, const char ∗keyname)

  *Creates a "Conversion to string failed" error.*
- ElektraError ∗ elektraErrorConversionFromString (KDBType targetType, const char ∗keyname, const char ∗sourceValue)

  *Creates a "Conversion from string failed" error.*
- ElektraError ∗ elektraErrorPureWarning (void)

  *Creates a dummy ElektraError struct to store warnings in.*
- const char ∗ elektraErrorCode (const ElektraError ∗error)
- const char ∗ elektraErrorDescription (const ElektraError ∗error)
- void elektraErrorReset (ElektraError ∗∗error)

  *Frees the memory used by the error and sets the referenced error variable to NULL.*

### 573.31.1 Detailed Description

The error module of the High level API.
Can be used to create errors from scratch or from errors that were attached to keys using src/libs/elektra/errors.

**Copyright**

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

## 573.31.2 Function Documentation

### 573.31.2.1 elektraErrorAddWarning()

```
void elektraErrorAddWarning (
            ElektraError * error,
            ElektraError * warning )
```

Adds a warning to an existing ElektraError struct.
If you want to report a warning without an error, create a dummy error with elektraErrorPureWarning() and then add a warning to it.

**Parameters**

| | |
|---|---|
| *error* | The error to which `warning` shall be added. |
| *warning* | The warning to add. Once added it is owned by `error`. DO NOT call elektraErrorReset() on it afterwards. |

### 573.31.2.2 elektraErrorConversionFromString()

```
ElektraError* elektraErrorConversionFromString (
            KDBType targetType,
            const char * keyname,
            const char * sourceValue )
```

Creates a "Conversion from string failed" error.

**Parameters**

| | |
|---|---|
| *targetType* | The type into which `sourceValue` couldn't be converted. |
| *keyname* | The name of the key that couldn't be converted. |
| *sourceValue* | The value that couldn't be converted. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

### 573.31.2.3 elektraErrorConversionToString()

```
ElektraError* elektraErrorConversionToString (
            KDBType sourceType,
            const char * keyname )
```

Creates a "Conversion to string failed" error.

**Parameters**

| | |
|---|---|
| *sourceType* | The type which failed to be converted to string. |
| *keyname* | The name of the key that couldn't be converted. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.31.2.4 elektraErrorCreate()

```
ElektraError* elektraErrorCreate (
            const char * code,
            const char * description,
            const char * module,
            const char * file,
            kdb_long_t line )
```

Creates a new ElektraError using the provided values.
The returned value will be allocated with elektraCalloc().

**Parameters**

| | |
|---|---|
| *code* | The error code of the error. Will be copied and stored in the struct. |
| *description* | The description of the error. Will be copied and stored in the struct. |
| *module* | The module that raised the error. Will be copied and stored in the struct. |
| *file* | The file that raised the error. Will be copied and stored in the struct. |
| *line* | The line in which the error was raised. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.31.2.5 elektraErrorFromKey()

```
ElektraError* elektraErrorFromKey (
            Key * key )
```

Extracts the error and all warnings from the given key.
If no error exists, a pure warning error will be used.

**See also**

elektraErrorPureWarning

**Note**

Use the functions in src/libs/elektra/errors.c to add errors to a key.

**Parameters**

| | |
|---|---|
| *key* | The to extract error and warnings from. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.31.2.6 elektraErrorKeyNotFound()

```
ElektraError* elektraErrorKeyNotFound (
            const char * keyname )
```

Creates a "Key not found" error.

**Parameters**

| | |
|---|---|
| *keyname* | The name of the key that wasn't found. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.31.2.7 elektraErrorNullError()

```
ElektraError* elektraErrorNullError (
            const char * function )
```

Creates a "Null error argument" error.

**Parameters**

| | |
|---|---|
| *function* | The name of the function that was called with a null pointer error argument. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.31.2.8 elektraErrorWrongType()

```
ElektraError* elektraErrorWrongType (
            const char * keyname,
            KDBType expectedType,
            KDBType actualType )
```

Creates a "Wrong type" error.

**Parameters**

| | |
|---|---|
| *keyname* | The name of the key that had the wrong type. |
| *expectedType* | The type that was expected. |
| *actualType* | The type that was actually found. |

**Returns**

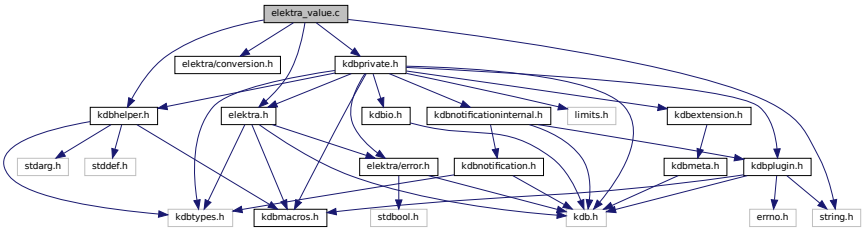
A newly allocated ElektraError (free with elektraErrorReset()).

## 573.32 elektra_value.c File Reference

Elektra High Level API.

```
#include "elektra.h"
#include "elektra/conversion.h"
#include "kdbease.h"
#include "kdbhelper.h"
#include "kdbprivate.h"
#include <string.h>
```

Include dependency graph for elektra_value.c:




## Functions

- Key * elektraFindKey (Elektra *elektra, const char *name, KDBType type)

  *Helper function for code generation.*
- const char * elektraFindReference (Elektra *elektra, const char *name)

  *Resolves the reference stored in a key.*
- KDBType elektraGetType (Elektra *elektra, const char *keyname)

  *Reads the type metadata of a given key.*
- const char * elektraGetRawString (Elektra *elektra, const char *name)

  *Get the raw string value of a key.*
- void elektraSetRawString (Elektra *elektra, const char *name, const char *value, KDBType type, ElektraError **error)

  *Set the raw string value of a key.*
- const char * elektraGetString (Elektra *elektra, const char *keyname)

  *Gets a string value.*
- kdb_boolean_t elektraGetBoolean (Elektra *elektra, const char *keyname)

  *Gets a boolean value.*
- kdb_char_t elektraGetChar (Elektra *elektra, const char *keyname)

  *Gets a char value.*
- kdb_octet_t elektraGetOctet (Elektra *elektra, const char *keyname)

  *Gets an octet value.*
- kdb_short_t elektraGetShort (Elektra *elektra, const char *keyname)

  *Gets a short value.*
- kdb_unsigned_short_t elektraGetUnsignedShort (Elektra *elektra, const char *keyname)

  *Gets a unsigned short value.*
- kdb_long_t elektraGetLong (Elektra *elektra, const char *keyname)

  *Gets a long value.*
- kdb_unsigned_long_t elektraGetUnsignedLong (Elektra *elektra, const char *keyname)

  *Gets a unsigned long value.*
- kdb_long_long_t elektraGetLongLong (Elektra *elektra, const char *keyname)

  *Gets a long long value.*
- kdb_unsigned_long_long_t elektraGetUnsignedLongLong (Elektra *elektra, const char *keyname)

  *Gets a long long value.*
- kdb_float_t elektraGetFloat (Elektra *elektra, const char *keyname)

  *Gets a float value.*
- kdb_double_t elektraGetDouble (Elektra *elektra, const char *keyname)

  *Gets a double value.*
- void elektraSetString (Elektra *elektra, const char *keyname, const char *value, ElektraError **error)

  *Sets a string value.*

- void elektraSetBoolean (Elektra *elektra, const char *keyname, kdb_boolean_t value, ElektraError **error)
  *Sets a boolean value.*
- void elektraSetChar (Elektra *elektra, const char *keyname, kdb_char_t value, ElektraError **error)
  *Sets a char value.*
- void elektraSetOctet (Elektra *elektra, const char *keyname, kdb_octet_t value, ElektraError **error)
  *Sets an octet value.*
- void elektraSetShort (Elektra *elektra, const char *keyname, kdb_short_t value, ElektraError **error)
  *Sets a short value.*
- void elektraSetUnsignedShort (Elektra *elektra, const char *keyname, kdb_unsigned_short_t value, ElektraError **error)
  *Sets a unsigned short value.*
- void elektraSetLong (Elektra *elektra, const char *keyname, kdb_long_t value, ElektraError **error)
  *Sets a long value.*
- void elektraSetUnsignedLong (Elektra *elektra, const char *keyname, kdb_unsigned_long_t value, ElektraError **error)
  *Sets a unsigned long value.*
- void elektraSetLongLong (Elektra *elektra, const char *keyname, kdb_long_long_t value, ElektraError **error)
  *Sets a long long value.*
- void elektraSetUnsignedLongLong (Elektra *elektra, const char *keyname, kdb_unsigned_long_long_t value, ElektraError **error)
  *Sets a unsigned long long value.*
- void elektraSetFloat (Elektra *elektra, const char *keyname, kdb_float_t value, ElektraError **error)
  *Sets a float value.*
- void elektraSetDouble (Elektra *elektra, const char *keyname, kdb_double_t value, ElektraError **error)
  *Sets a double value.*

### 573.32.1 Detailed Description

Elektra High Level API.

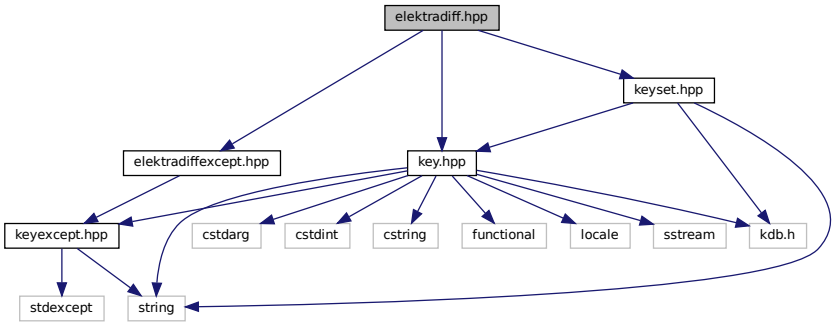
**Copyright**

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

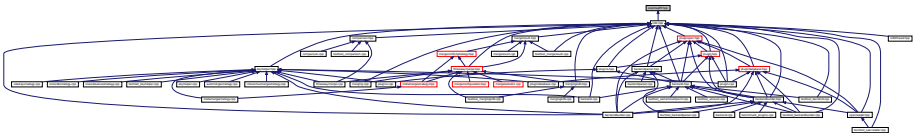
## 573.33 elektradiff.hpp File Reference

```
#include <elektradiffexcept.hpp>
#include <key.hpp>
#include <keyset.hpp>
#include <kdbdiff.h>
```

Include dependency graph for elektradiff.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class kdb::ElektraDiff

  *This class is a wrapper around the ElektraDiff C struct.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.33.1 Detailed Description

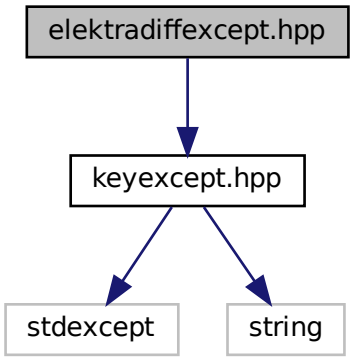
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

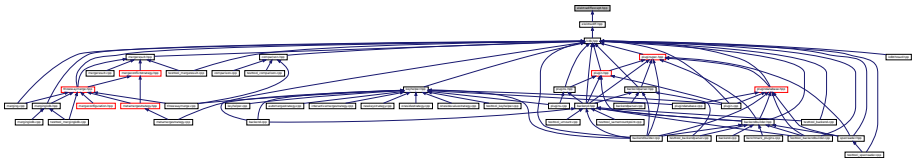
# 573.34 elektradiffexcept.hpp File Reference

```
#include <keyexcept.hpp>
```

Include dependency graph for elektradiffexcept.hpp:




This graph shows which files directly or indirectly include this file:



### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.34.1 Detailed Description

**Copyright**

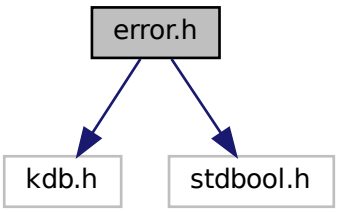
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.35 error.h File Reference

Elektra error.

```
#include "types.h"
#include <kdb.h>
#include <stdbool.h>
```

Include dependency graph for error.h:




This graph shows which files directly or indirectly include this file:

### Functions

- ElektraError * elektraErrorPureWarning (void)

  *Creates a dummy ElektraError struct to store warnings in.*
- const char * elektraErrorCode (const ElektraError *error)
- const char * elektraErrorDescription (const ElektraError *error)
- void elektraErrorReset (ElektraError **error)

  *Frees the memory used by the error and sets the referenced error variable to NULL.*
- ElektraError * elektraErrorConversionToString (KDBType sourceType, const char *keyname)

  *Creates a "Conversion to string failed" error.*
- ElektraError * elektraErrorConversionFromString (KDBType targetType, const char *keyname, const char *sourceValue)

  *Creates a "Conversion from string failed" error.*

### 573.35.1 Detailed Description

Elektra error.

**Copyright**

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

### 573.35.2 Function Documentation

#### 573.35.2.1 elektraErrorConversionFromString()

```
ElektraError* elektraErrorConversionFromString (
            KDBType targetType,
            const char * keyname,
            const char * sourceValue )
```

Creates a "Conversion from string failed" error.

**Parameters**

| | |
|---|---|
| *targetType* | The type into which `sourceValue` couldn't be converted. |
| *keyname* | The name of the key that couldn't be converted. |
| *sourceValue* | The value that couldn't be converted. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.35.2.2 elektraErrorConversionToString()

```
ElektraError* elektraErrorConversionToString (
            KDBType sourceType,
            const char * keyname )
```

Creates a "Conversion to string failed" error.

**Parameters**

| | |
|---|---|
| *sourceType* | The type which failed to be converted to string. |
| *keyname* | The name of the key that couldn't be converted. |

**Returns**

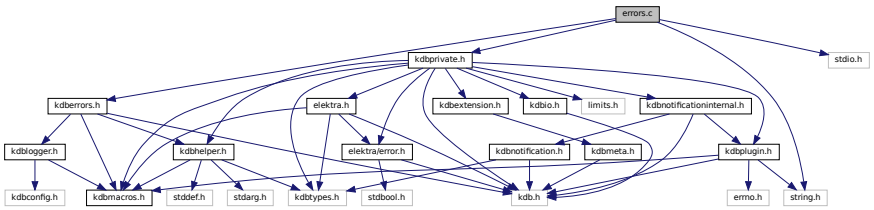
A newly allocated ElektraError (free with elektraErrorReset()).

## 573.36 errors.c File Reference

Used for writing the error/warning information into a key to be used for emitting messages to the user.

```
#include <kdberrors.h>
#include <kdbprivate.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for errors.c:




## Functions

- void elektraCopyError (Key ∗target, Key ∗source)

  *Copy the error from the source key to target key.*
- void elektraCopyWarnings (Key ∗target, Key ∗source)

  *Copy the warnings from the source key to target key.*
- void elektraCopyErrorAndWarnings (Key ∗target, Key ∗source)

  *Copies the error and warnings from the source key to the target key.*

### 573.36.1 Detailed Description

Used for writing the error/warning information into a key to be used for emitting messages to the user.

#### Copyright

BSD License (see doc/COPYING or https://www.libelektra.org)

### 573.36.2 Function Documentation

#### 573.36.2.1 elektraCopyError()

```
void elektraCopyError (
            Key * target,
            Key * source )
```

Copy the error from the source key to target key.

#### Parameters

| | |
|---|---|
| *target* | append error to this key |
| *source* | copy error from this key |

#### 573.36.2.2 elektraCopyErrorAndWarnings()

```
void elektraCopyErrorAndWarnings (
            Key * target,
            Key * source )
```

Copies the error and warnings from the source key to the target key.
Note that only 100 warnings can be had. If we exceed that, we'll override the existing warnings in the target key

#### Parameters

| | |
|---|---|
| *target* | copy error and warnings to this key |

**Parameters**

| | |
|---|---|
| *source* | copy error and warnings from this key |

#### 573.36.2.3 elektraCopyWarnings()

```
void elektraCopyWarnings (
            Key * target,
            Key * source )
```

Copy the warnings from the source key to target key.
Note that only 100 warnings can be had. If we exceed that, we'll override the existing warnings in the target key.

**Parameters**

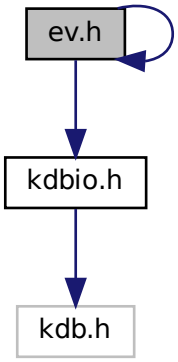
| | |
|---|---|
| *target* | append warnings to this key |
| *source* | copy warnings from this key |

## 573.37 ev.h File Reference

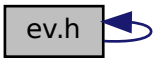
Declarations for the ev I/O binding.

```
#include <ev.h>
#include <kdbio.h>
```

Include dependency graph for ev.h:



This graph shows which files directly or indirectly include this file:




## Functions

- ElektraIoInterface * elektraIoEvNew (struct ev_loop *loop)

  *Create and initialize a new I/O binding.*

### 573.37.1 Detailed Description

Declarations for the ev I/O binding.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.37.2 Function Documentation

#### 573.37.2.1 elektraIoEvNew()

```
ElektraIoInterface* elektraIoEvNew (
            struct ev_loop * loop )
```

Create and initialize a new I/O binding.

**Parameters**

| | |
|---|---|
| *loop* | Loop to use for I/O operations |

**Returns**

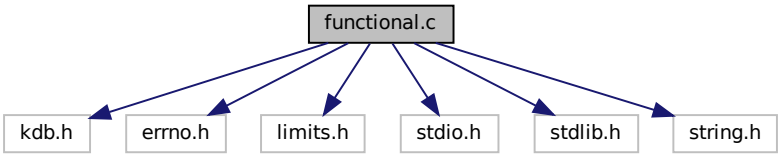
Populated I/O interface

## 573.38 functional.c File Reference

Functional helper.

```
#include <kdb.h>
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for functional.c:




## Functions

- int elektraKsFilter (KeySet *result, KeySet *input, int(*filter)(const Key *k, void *argument), void *argument)

  *return only those keys from the given keyset that pass the supplied filter function with the supplied argument*
- int elektraKsToMemArray (KeySet *ks, Key **buffer)

  *Builds an array of pointers to the keys in the supplied keyset.*

### 573.38.1 Detailed Description

Functional helper.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.38.2 Function Documentation

#### 573.38.2.1 elektraKsFilter()

```
int elektraKsFilter (
            KeySet * result,
            KeySet * input,
            int(*)(const Key *k, void *argument) filter,
            void * argument )
```

return only those keys from the given keyset that pass the supplied filter function with the supplied argument

**Parameters**

| | |
|---|---|
| *result* | the keyset that should contain the filtered keys |
| *input* | the keyset whose keys should be filtered |
| *filter* | a function pointer to a function that will be used to filter the keyset. A key will be taken if the function returns a value greater than 0. |
| *argument* | an argument that will be passed to the filter function each time it is called |

**Returns**

the number of filtered keys if the filter function always returned a positive value, -1 otherwise

**Return values**

| | |
|---|---|
| *NULL* | on NULL pointer |

#### 573.38.2.2 elektraKsToMemArray()

```
int elektraKsToMemArray (
            KeySet * ks,
            Key ** buffer )
```

Builds an array of pointers to the keys in the supplied keyset.
The keys are not copied, calling keyDel may remove them from the keyset.
The size of the buffer can be easily allocated via ksGetSize. Example:

```
KeySet *ks = somekeyset;
Key **keyArray = calloc (ksGetSize(ks), sizeof (Key *));
elektraKsToMemArray (ks, keyArray);
... work with the array ...
elektraFree (keyArray);
```

**Parameters**

| | |
|---|---|
| *ks* | the keyset object to work with |
| *buffer* | the buffer to put the result into |

**Returns**

the number of elements in the array if successful

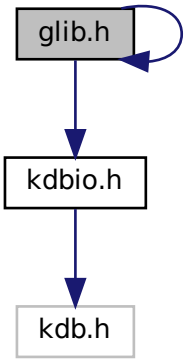
a negative number on null pointers or if an error occurred

## 573.39 glib.h File Reference

Declarations for the glib I/O binding.

```
#include <glib.h>
#include <kdbio.h>
```

Include dependency graph for glib.h:



This graph shows which files directly or indirectly include this file:


## Functions

- ElektraIoInterface * elektraIoGlibNew (GMainContext *context)

  *Create and initialize a new I/O binding.*

### 573.39.1 Detailed Description

Declarations for the glib I/O binding.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.39.2 Function Documentation

#### 573.39.2.1 elektraIoGlibNew()

```
ElektraIoInterface* elektraIoGlibNew (
            GMainContext * context )
```

Create and initialize a new I/O binding.

**Parameters**

| | |
|---|---|
| *context* | Context to use for I/O operations |

**Returns**

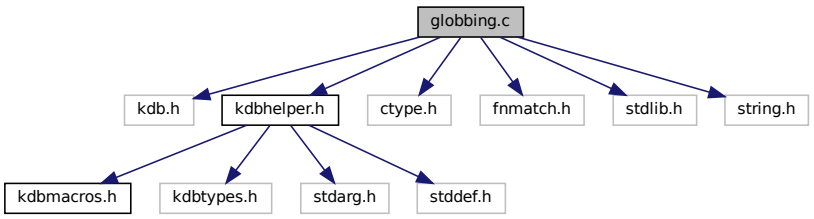
Populated I/O interface

## 573.40 globbing.c File Reference

Library for performing globbing on keynames.

```
#include <kdb.h>
#include <kdbease.h>
#include <kdbglobbing.h>
#include <kdbhelper.h>
#include <ctype.h>
#include <fnmatch.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for globbing.c:




## Functions

- int elektraKeyGlob (const Key ∗key, const char ∗pattern)
  *checks whether a given Key matches a given globbing pattern*
- int elektraKsGlob (KeySet ∗result, KeySet ∗input, const char ∗pattern)
  *filters a given KeySet by applying a globbing pattern*

### 573.40.1 Detailed Description

Library for performing globbing on keynames.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.40.2 Function Documentation

#### 573.40.2.1 elektraKeyGlob()

```
int elektraKeyGlob (
            const Key * key,
            const char * pattern )
```

checks whether a given Key matches a given globbing pattern
WARNING: this method will not work correctly, if key parts contain embedded (escaped) slashes.
The globbing patterns for this function are a superset of those from glob(7) used with the FNM_PATHNAME flag:

- '∗' matches any series of characters other than '/'
- '?' matches any single character except '/'
- '#', when used as "/#/" (or "/#" at the end of `pattern`), matches a valid array item
- '_', when used as "/_/"(or "/_" at the end of `pattern`), matches a key part that is **not** a valid array item
- everything between '[' and ']' is treated as a character class, matching exactly one of the given characters (see glob(7) for details)
- if the pattern ends with "/__", matching key names may contain arbitrary suffixes

**Note**

'∗' cannot match an empty key name part. This also means patterns like "something/∗" will not match the key "something". This is because each slash ('/') in the pattern has to correspond to a slash in the canonical key name, which neither end in a slash nor contain multiple slashes in sequence.

use "[_]", "[#]", "[∗]", "[?]" and "[[ ]" to match the literal characters '_', '#', '∗', '?' and '['. Using backslash ('\') for escaping is not supported.

**Parameters**

| | |
|---|---|
| *key* | the Key to match against the globbing pattern |
| *pattern* | the globbing pattern used |

**Return values**

| | |
|---|---|
| *0* | if `key` is not NULL, `pattern` is not NULL and `pattern` matches `key` |
| *ELEKTRA_GLOB_NOMATCH* | otherwise |

**See also**

isArrayName(), for info on valid array items

#### 573.40.2.2 elektraKsGlob()

```
int elektraKsGlob (
            KeySet * result,
            KeySet * input,
            const char * pattern )
```

filters a given KeySet by applying a globbing pattern

**Parameters**

| | |
|---|---|
| *result* | the KeySet to which the matching keys should be appended |
| *input* | the KeySet whose keys should be filtered |
| *pattern* | the globbing pattern used |

**Returns**

the number of Keys appended to result or -1, if `result`, `input` or `pattern` are NULL

**See also**

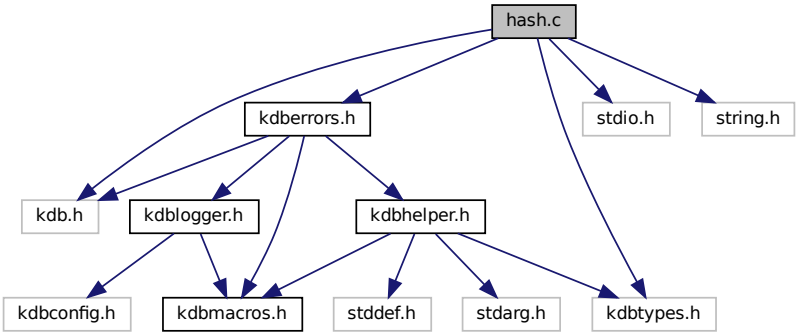
elektraKeyGlob(), for explanation of globbing pattern

## 573.41 hash.c File Reference

Provides functions to hash Elektra data structures.

```
#include <kdb.h>
#include <kdbease.h>
#include <kdberrors.h>
#include <kdbtypes.h>
#include <stdio.h>
#include <string.h>
#include "sha-256.h"
```

Include dependency graph for hash.c:




## Functions

- kdb_boolean_t calculateSpecificationToken (char hash_string[65], KeySet *ks, Key *parentKey)

  *Calculate a specification token for the KeySet of an application.*

### 573.41.1 Detailed Description

Provides functions to hash Elektra data structures.

**Copyright**

BSD License (see LICENSE.md or `https://www.libelektra.org`)

### 573.41.2 Function Documentation

#### 573.41.2.1 calculateSpecificationToken()

```
kdb_boolean_t calculateSpecificationToken (
            char hash_string[65],
            KeySet * ks,
            Key * parentKey )
```

Calculate a specification token for the KeySet of an application.
The KeySet of an application is identified as all keys below the applications root key.

**Precondition**

The parentKey must have the correct namespace. E.g. If only keys from the spec:/ should be considered for the token calculation, pass a key with KEY_NS_SPEC.

**Note**

Array parent key's (e.g., `/format/#`) are ignored for the token. See inline documentation below for rationale.

**Parameters**

| | |
|---|---|
| *hash_string* | A string. After successful execution this will contain the hash as hex-string. |
| *ks* | The KeySet for the application. |
| *parentKey* | The Key below which all the relevant keys are. Keys that are not below `parentKey` are ignored. The key's namespace is important (see preconditions) |

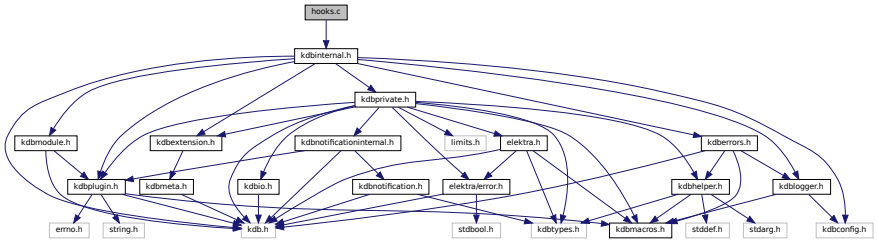
**Return values**

| | |
|---|---|
| *false* | If an error occurred. |
| *true* | If the computation was successful. |

Loop through all keys relevant for token calculation.
Ignore array parents for token calculation. Rationale: There is a bug in the spec plugin that is triggered on changing the size of an array. It leads to array parents vanishing from the spec namespace and thus a different token. See https://github.com/ElektraInitiative/libelektra/issues/4061
Include NULL teminator in this and all following key/value strings, to avoid the following bug: https://github.com/ElektraInitiative/libelektra/issues/4110

## 573.42 hooks.c File Reference

`#include <kdbinternal.h>`
Include dependency graph for hooks.c:




### Functions

- void freeHooks (KDB *kdb, Key *errorKey)

  *Uninitializes and frees all hooks in the passed KDB handle.*
- Plugin * elektraFindInternalNotificationPlugin (KDB *kdb)

  *This method looks for the hook plugin 'internalnotification'.*
- int initHooks (KDB *kdb, const KeySet *config, KeySet *modules, const KeySet *contract, Key *errorKey)

  *Initializes the hooks stored in the passed KDB handle.*

### 573.42.1 Detailed Description

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.42.2 Function Documentation

#### 573.42.2.1 elektraFindInternalNotificationPlugin()

```
Plugin* elektraFindInternalNotificationPlugin (
            KDB * kdb )
```

This method looks for the hook plugin 'internalnotification'.

**Parameters**

| | |
|---|---|
| *kdb* | the KDB instance in which to search |

**Returns**

NULL if not loaded, pointer to the plugin otherwise

#### 573.42.2.2 freeHooks()

```
void freeHooks (
            KDB * kdb,
            Key * errorKey )
```

Uninitializes and frees all hooks in the passed KDB handle.

**Parameters**

| | |
|---|---|
| *kdb* | the KDB handle where the hooks should be freed |
| *errorKey* | the key which holds errors and warnings which were issued |

#### 573.42.2.3 initHooks()

```
int initHooks (
            KDB * kdb,
            const KeySet * config,
            KeySet * modules,
            const KeySet * contract,
            Key * errorKey )
```

Initializes the hooks stored in the passed KDB handle.
If the handle already contains initialized hooks, they will be reinitialized, including unloading and loading of their plugins. Parameters `config` and `contract` will be used to determine which hooks to populate.

**Parameters**

| | |
|---|---|
| *kdb* | the KDB instance where the hooks should be initialized |
| *config* | KeySet containing the current config in `system:/elektra` namespace |
| *modules* | the current list of loaded modules |
| *contract* | the contract passed to `kdbOpen` |
| *errorKey* | the key which holds errors and warnings which were issued |

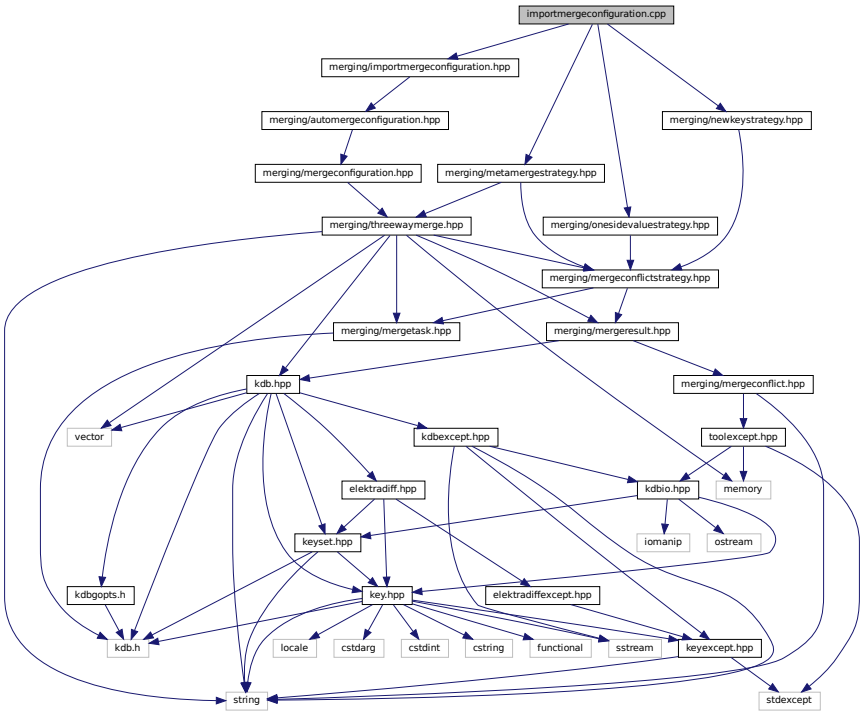
**Returns**

0 on success, -1 on failure

## 573.43 importmergeconfiguration.cpp File Reference

```
#include <merging/importmergeconfiguration.hpp>
#include <merging/metamergestrategy.hpp>
#include <merging/newkeystrategy.hpp>
#include <merging/onesidevaluestrategy.hpp>
```

Include dependency graph for importmergeconfiguration.cpp:




### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.43.1 Detailed Description

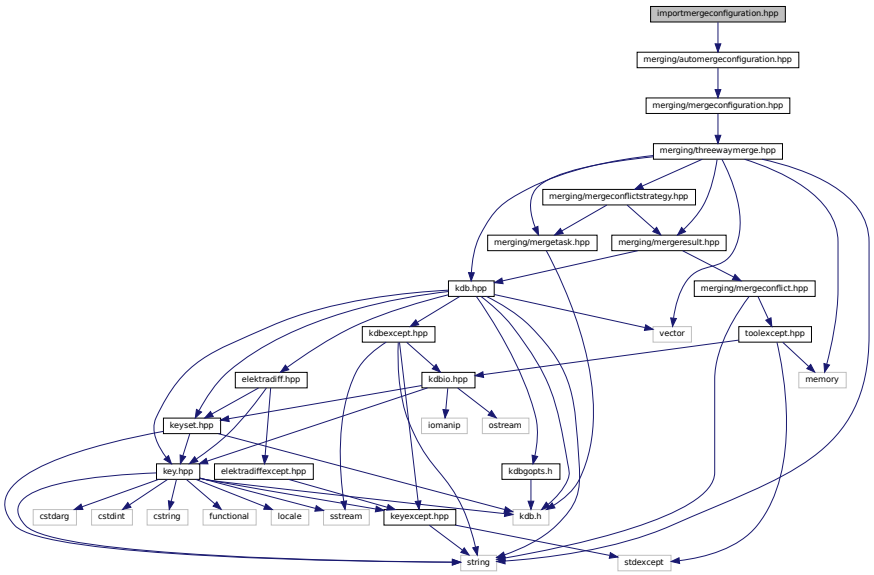
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

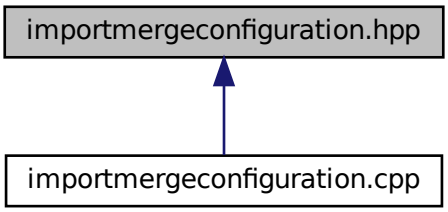
## 573.44 importmergeconfiguration.hpp File Reference

A configuration for a simple automerge and guaranteed conflict resolution by one side.

`#include <merging/automergeconfiguration.hpp>`
Include dependency graph for importmergeconfiguration.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.44.1 Detailed Description

A configuration for a simple automerge and guaranteed conflict resolution by one side.

**Copyright**

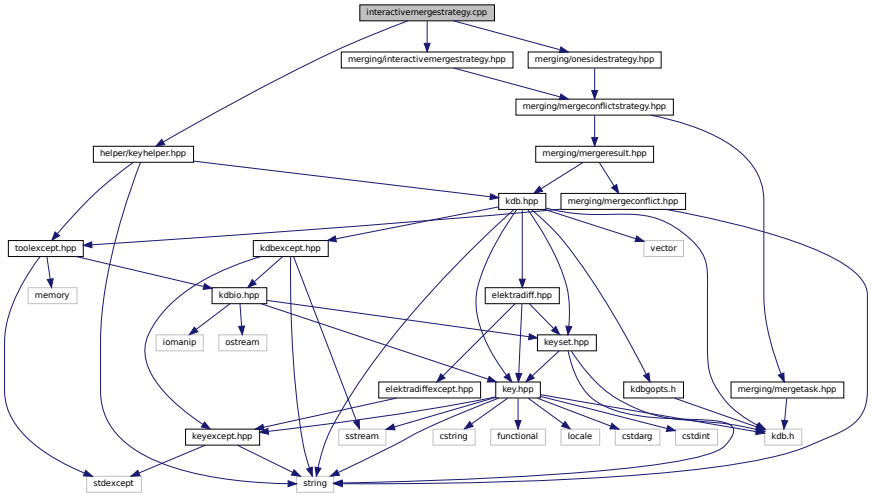
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.45 interactivemergestrategy.cpp File Reference

Implementation of InteractiveMergeStrategy.

```
#include <helper/keyhelper.hpp>
#include <merging/interactivemergestrategy.hpp>
#include <merging/onesidestrategy.hpp>
```

Include dependency graph for interactivemergestrategy.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

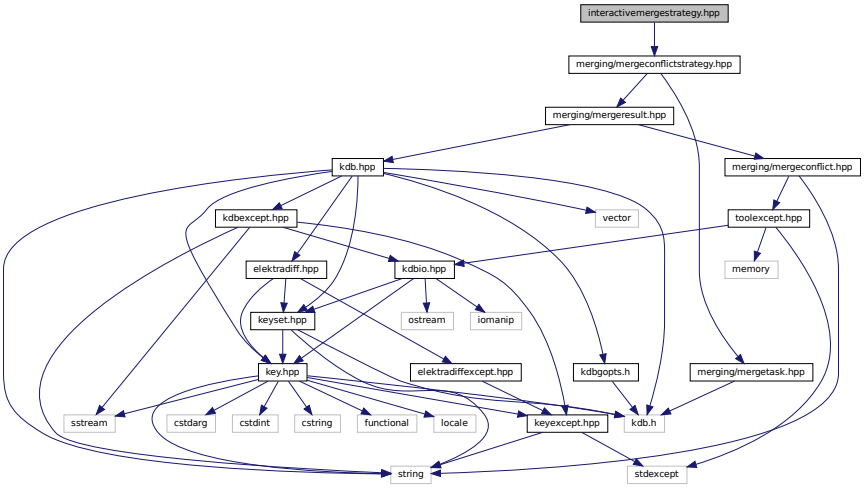
  *This namespace is for the libtool library.*

### 573.45.1 Detailed Description

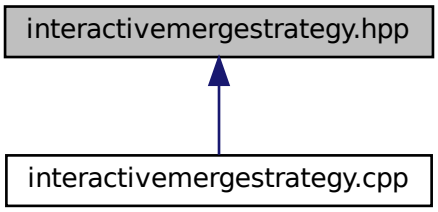
Implementation of InteractiveMergeStrategy.

# 573.46 interactivemergestrategy.hpp File Reference

Interactive merge strategy asking for user input at each step.
`#include <merging/mergeconflictstrategy.hpp>`
Include dependency graph for interactivemergestrategy.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.46.1 Detailed Description

Interactive merge strategy asking for user input at each step.

### Copyright

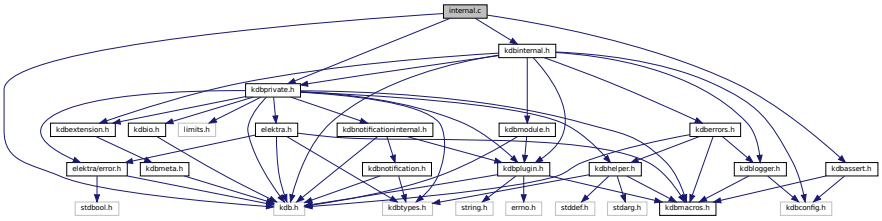


## 573.47 internal.c File Reference

Internal methods for Elektra.

```
#include "kdbprivate.h"
#include <kdb.h>
#include "kdbinternal.h"
#include <kdbassert.h>
```

Include dependency graph for internal.c:




### Functions

- ssize_t elektraMemcpy (Key **array1, Key **array2, size_t size)

  *Internal Methods for Elektra.*
- ssize_t elektraMemmove (Key **array1, Key **array2, size_t size)

  *Copies the key array2 into where array1 points.*
- int elektraStrCmp (const char *s1, const char *s2)

  *Compare Strings.*
- int elektraStrNCmp (const char *s1, const char *s2, size_t n)

  *Compare Strings up to a maximum length.*
- int elektraStrCaseCmp (const char *s1, const char *s2)

  *Compare Strings ignoring case.*
- int elektraStrNCaseCmp (const char *s1, const char *s2, size_t n)

  *Compare Strings ignoring case up to a maximum length.*
- int elektraMemCaseCmp (const char *s1, const char *s2, size_t size)

  *Compare two memory regions but make cmp chars uppercase before comparison.*
- int elektraRealloc (void **buffer, size_t size)

  *Reallocate Storage in a save way.*
- void * elektraMalloc (size_t size)

  *Allocate memory for Elektra.*
- void * elektraCalloc (size_t size)

  *Allocate memory for Elektra.*
- void elektraFree (void *ptr)

  *Free memory of Elektra or its backends.*
- char * elektraStrDup (const char *s)

  *Copy string into new allocated memory.*
- void * elektraMemDup (const void *s, size_t n)

  *Copy buffer into new allocated memory.*
- size_t elektraStrLen (const char *s)

  *Calculates the length in bytes of a string.*

- char * elektraFormat (const char *format,...)

  *Does string formatting in fresh allocated memory.*
- char * elektraVFormat (const char *format, va_list arg_list)

  *Does string formatting in fresh allocated memory.*

## 573.47.1 Detailed Description

Internal methods for Elektra.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.47.2 Function Documentation

### 573.47.2.1 elektraCalloc()

```
void* elektraCalloc (
            size_t size )
```

Allocate memory for Elektra.
Memory will be set to 0.

**Parameters**

| | |
|---|---|
| *size* | the requested size |

**See also**

elektraMalloc

### 573.47.2.2 elektraFormat()

```
char* elektraFormat (
            const char * format,
             ...  )
```

Does string formatting in fresh allocated memory.

**Parameters**

| | |
|---|---|
| *format* | as in printf() |
| ... | as in printf() |

**Returns**

new allocated memory (free with elektraFree)

### 573.47.2.3 elektraFree()

```
void elektraFree (
            void * ptr )
```

Free memory of Elektra or its backends.

**Parameters**

| | |
|---|---|
| *ptr* | the pointer to free |

If ptr is NULL, no operation is performed.

**See also**

elektraMalloc

#### 573.47.2.4 elektraMalloc()

```
void* elektraMalloc (
            size_t size )
```

Allocate memory for Elektra.

```
if ((buffer = elektraMalloc (length)) == 0) {
        // here comes the failure handler
        // no allocation happened here, so don't use buffer
#if DEBUG
        fprintf (stderr, "Allocation error");
#endif
        // return with error
}
```

**Parameters**

| | |
|---|---|
| *size* | the requested size |

This function is compatible to ANSI-C malloc

**See also**

elektraFree

elektraCalloc

#### 573.47.2.5 elektraMemCaseCmp()

```
int elektraMemCaseCmp (
            const char * s1,
            const char * s2,
            size_t size )
```

Compare two memory regions but make cmp chars uppercase before comparison.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |
| *size* | to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

### 573.47.2.6 elektraMemcpy()

```
ssize_t elektraMemcpy (
            Key ** array1,
            Key ** array2,
            size_t size )
```

Internal Methods for Elektra.
To use them:
`#include <kdbinternal.h>`
There are some areas where libraries have to reimplement some basic functions to archive support for non-standard systems, for testing purposes or to provide a little more convenience. Copies the key array2 into where array1 points. It copies size elements.
Overlapping is prohibited, use elektraMemmove() instead.

**Parameters**

| | |
|---|---|
| *array1* | the destination |
| *array2* | the source |
| *size* | how many pointer to Keys to copy |

**Return values**

| | |
|---|---|
| *-1* | on null pointers |
| *0* | if nothing was done |

**Returns**

size how many keys were copied

### 573.47.2.7 elektraMemDup()

```
void* elektraMemDup (
            const void * s,
            size_t n )
```

Copy buffer into new allocated memory.
You need to free the memory yourself.
This function also works with \0 characters in the buffer. The length is taken as given, it must be correct.

**Returns**

0 if out of memory, a pointer otherwise

**Parameters**

| | |
|---|---|
| *s* | must be an allocated buffer |
| *n* | the number of bytes to copy from s |

### 573.47.2.8 elektraMemmove()

```
ssize_t elektraMemmove (
            Key ** array1,
            Key ** array2,
            size_t size )
```

Copies the key array2 into where array1 points.
It copies size elements.
Overlapping is ok. If they do not overlap consider elektraMemcpy() instead.

#### Parameters

| | |
|---|---|
| *array1* | the destination |
| *array2* | the source |
| *size* | how many pointer to Keys to copy |

#### Return values

| | |
|---|---|
| *-1* | on null pointers |
| *0* | if nothing was done |

#### Returns

size how many keys were copied

### 573.47.2.9 elektraRealloc()

```
int elektraRealloc (
            void ** buffer,
            size_t size )
```

Reallocate Storage in a save way.

```
if (elektraRealloc ((void **) & buffer, new_length) < 0) {
        // here comes the failure handler
        // you can still use the old buffer
#if DEBUG
        fprintf (stderr, "Reallocation error\n");
#endif
        elektraFree (buffer);
        buffer = 0;
        // return with error
}
```

#### Parameters

| | |
|---|---|
| *buffer* | is a pointer to a elektraMalloc |
| *size* | is the new size for the memory |

#### Return values

| | |
|---|---|
| *-1* | on failure |
| *0* | on success |

### 573.47.2.10 elektraStrCaseCmp()

```
int elektraStrCaseCmp (
```

```
            const char * s1,
            const char * s2 )
```

Compare Strings ignoring case.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.47.2.11 elektraStrCmp()

```
int elektraStrCmp (
            const char * s1,
            const char * s2 )
```

Compare Strings.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.47.2.12 elektraStrDup()

```
char* elektraStrDup (
            const char * s )
```

Copy string into new allocated memory.
You need to free the memory yourself.

**Note**

that size is determined at runtime. So if you have a size information, don't use that function.

**Parameters**

| | |
|---|---|
| *s* | the null-terminated string to duplicate |

**Returns**

0 if out of memory, a pointer otherwise

**Precondition**

s must be a c-string.

**See also**

elektraFree

elektraStrLen

elektraMemDup

### 573.47.2.13 elektraStrLen()

```
size_t elektraStrLen (
            const char * s )
```

Calculates the length in bytes of a string.
This function differs from strlen() because it is Unicode and multibyte chars safe. While strlen() counts characters and ignores the final NULL, elektraStrLen() count bytes including the ending NULL.
It must not be used to search for / in the name, because it does not consider escaping. Instead use the unescaped name.

**See also**

keyUnescapedName()

**Parameters**

| | |
|---|---|
| *s* | the string to get the length from |

**Returns**

number of bytes used by the string, including the final NULL.

### 573.47.2.14 elektraStrNCaseCmp()

```
int elektraStrNCaseCmp (
            const char * s1,
            const char * s2,
            size_t n )
```

Compare Strings ignoring case up to a maximum length.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |

**Parameters**

| | |
|---|---|
| *s2* | The second string to be compared |
| *n* | The maximum length to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

### 573.47.2.15 elektraStrNCmp()

```
int elektraStrNCmp (
            const char * s1,
            const char * s2,
            size_t n )
```

Compare Strings up to a maximum length.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |
| *n* | The maximum length to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

### 573.47.2.16 elektraVFormat()

```
char* elektraVFormat (
            const char * format,
            va_list arg_list )
```

Does string formatting in fresh allocated memory.

**Parameters**

| | |
|---|---|
| *format* | as in vprintf() |
| *arg_list* | as in vprintf() |

**Returns**

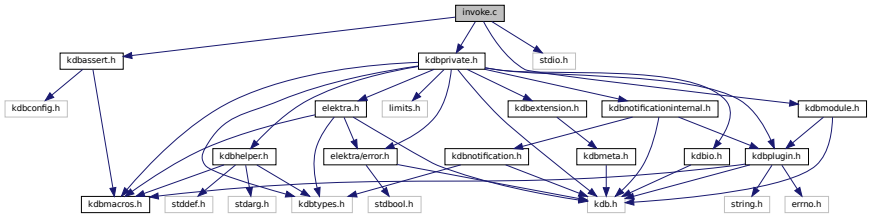
new allocated memory (free with elektraFree)

## 573.48 invoke.c File Reference

Library for invoking exported plugin functions.

```
#include <kdbassert.h>
#include <kdbinvoke.h>
#include <kdbmodule.h>
#include <kdbprivate.h>
#include <stdio.h>
```

Include dependency graph for invoke.c:




### Functions

- ElektraInvokeHandle * elektraInvokeInitialize (const char *elektraPluginName)
- ElektraInvokeHandle * elektraInvokeOpen (const char *elektraPluginName, KeySet *config, Key *errorKey)

  *Opens a new handle to invoke functions for a plugin.*
- const void * elektraInvokeGetFunction (ElektraInvokeHandle *handle, const char *elektraPluginFunction↩ Name)

  *Get a function pointer.*
- KeySet * elektraInvokeGetPluginConfig (ElektraInvokeHandle *handle)

  *Get the configuration the plugin uses.*
- const char * elektraInvokeGetPluginName (ElektraInvokeHandle *handle)

  *Get the name of the plugin.*
- void * elektraInvokeGetPluginData (ElektraInvokeHandle *handle)

  *Get the data of the plugin.*
- KeySet * elektraInvokeGetModules (ElektraInvokeHandle *handle)

  *Get the modules used for invoking.*
- KeySet * elektraInvokeGetExports (ElektraInvokeHandle *handle)

  *Get the exports from the plugin.*
- int elektraInvoke2Args (ElektraInvokeHandle *handle, const char *elektraPluginFunctionName, KeySet *ks, Key *k)

  *A convenience function to call a function with two arguments.*
- void elektraInvokeClose (ElektraInvokeHandle *handle, Key *errorKey)

  *Closes all affairs with the handle.*
- int elektraInvokeCallDeferable (ElektraInvokeHandle *handle, const char *elektraPluginFunctionName, Key↩ Set *parameters)

*Invokes a deferable function on an invoke handle.*

- void elektraInvokeExecuteDeferredCalls (ElektraInvokeHandle *handle, ElektraDeferredCallList *list)

  *Execute deferred calls from list on given invoke handle.*
- int elektraDeferredCall (Plugin *handle, const char *elektraPluginFunctionName, KeySet *parameters)

  *Call a deferrable function on a plugin handle.*
- int elektraDeferredCallAdd (ElektraDeferredCallList *list, const char *name, KeySet *parameters)

  *Add a new deferred call to the deferred call list.*
- ElektraDeferredCallList * elektraDeferredCallCreateList (void)

  *Create new deferred call list.*
- void elektraDeferredCallDeleteList (ElektraDeferredCallList *list)

  *Delete deferred call list.*
- void elektraDeferredCallsExecute (Plugin *plugin, ElektraDeferredCallList *list)

  *Execute deferred calls on given plugin.*

### 573.48.1 Detailed Description

Library for invoking exported plugin functions.

**Copyright**

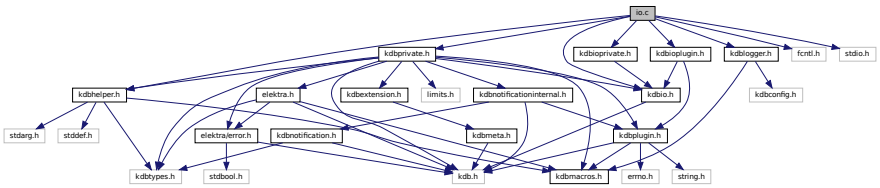
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.49 io.c File Reference

Implementation of I/O functions as defined in kdbio.h.

```
#include <kdbhelper.h>
#include <kdbinvoke.h>
#include <kdbio.h>
#include <kdbioplugin.h>
#include <kdbioprivate.h>
#include <kdblogger.h>
#include <kdbprivate.h>
#include <fcntl.h>
#include <stdio.h>
```

Include dependency graph for io.c:




### Functions

- int elektraIoContract (KeySet *contract, ElektraIoInterface *ioBinding)

  *Creates a contract for use with kdbOpen() that sets up an I/O binding.*
- ElektraIoInterface * elektraIoGetBinding (KDB *kdb)

  *Get I/O binding for asynchronous I/O operations for KDB instance.*
- ElektraIoInterface * elektraIoNewBinding (ElektraIoBindingAddFd *addFd, ElektraIoBindingUpdateFd *updateFd, ElektraIoBindingRemoveFd *removeFd, ElektraIoBindingAddTimer *addTimer, ElektraIoBindingUpdateTimer *updateTimer, ElektraIoBindingRemoveTimer *removeTimer, ElektraIoBindingAddIdle *addIdle, ElektraIoBindingUpdateIdle *updateIdle, ElektraIoBindingRemoveIdle *removeIdle, ElektraIoBindingCleanup *cleanup)

*Create a new I/O binding.*

- int elektraIoBindingAddFd (ElektraIoInterface *binding, ElektraIoFdOperation *fdOp)

  *Add file descriptor to be watched by I/O binding.*
- int elektraIoBindingUpdateFd (ElektraIoFdOperation *fdOp)

  *Notify I/O binding about changes to file descriptor watch operation.*
- int elektraIoBindingRemoveFd (ElektraIoFdOperation *fdOp)

  *Remove file descriptor from I/O binding.*
- int elektraIoBindingAddTimer (ElektraIoInterface *binding, ElektraIoTimerOperation *timerOp)

  *Add timer to I/O binding.*
- int elektraIoBindingUpdateTimer (ElektraIoTimerOperation *timerOp)

  *Notifiy I/O binding about changes to timer structure.*
- int elektraIoBindingRemoveTimer (ElektraIoTimerOperation *timerOp)

  *Remove timer from I/O binding.*
- int elektraIoBindingAddIdle (ElektraIoInterface *binding, ElektraIoIdleOperation *idleOp)

  *Add idle to I/O binding.*
- int elektraIoBindingUpdateIdle (ElektraIoIdleOperation *idleOp)

  *Notifiy I/O binding about changes to idle structure.*
- int elektraIoBindingRemoveIdle (ElektraIoIdleOperation *idleOp)

  *Remove idle from I/O binding.*
- int elektraIoBindingCleanup (ElektraIoInterface *binding)

  *Free memory used by I/O binding.*
- void * elektraIoBindingGetData (ElektraIoInterface *binding)

  *Get private data from I/O Binding.*
- int elektraIoBindingSetData (ElektraIoInterface *binding, void *data)

  *Set private data from I/O Binding.*
- ElektraIoFdOperation * elektraIoNewFdOperation (int fd, int flags, int enabled, ElektraIoFdCallback callback, void *privateData)

  *Create a new file descriptor watch operation.*
- ElektraIoTimerOperation * elektraIoNewTimerOperation (unsigned int interval, int enabled, ElektraIoTimerCallback callback, void *privateData)

  *Create a new timer operation.*
- ElektraIoIdleOperation * elektraIoNewIdleOperation (int enabled, ElektraIoIdleCallback callback, void *privateData)

  *Create a new idle operation.*
- int elektraIoFdSetEnabled (ElektraIoFdOperation *fdOp, int enabled)

  *Enable or disable file descriptor watch operation.*
- int elektraIoFdSetFlags (ElektraIoFdOperation *fdOp, int flags)

  *Update flag bitmask of file descriptor watch operation.*
- int elektraIoTimerSetEnabled (ElektraIoTimerOperation *timerOp, int enabled)

  *Enable or disable timer operation.*
- int elektraIoTimerSetInterval (ElektraIoTimerOperation *timerOp, unsigned int interval)

  *Update interval of timer operation.*
- int elektraIoIdleSetEnabled (ElektraIoIdleOperation *idleOp, int enabled)

  *Enable or disable idle operation.*
- int elektraIoFdGetFd (ElektraIoFdOperation *fdOp)

  *Get file descriptor number from operation.*
- void * elektraIoFdGetData (ElektraIoFdOperation *fdOp)

  *Get private data from operation.*
- int elektraIoFdSetBindingData (ElektraIoFdOperation *fdOp, void *data)

  *Set private binding data for operation.*
- void * elektraIoFdGetBindingData (ElektraIoFdOperation *fdOp)

*Get private binding data from operation.*

- ElektraIoInterface * elektraIoFdGetBinding (ElektraIoFdOperation *fdOp)

  *Get binding from operation.*
- int elektraIoFdIsEnabled (ElektraIoFdOperation *fdOp)

  *Check if file descriptor watch operation is enabled or disabled.*
- int elektraIoFdGetFlags (ElektraIoFdOperation *fdOp)

  *Get flag bitmask of file descriptor watch operation.*
- ElektraIoFdCallback elektraIoFdGetCallback (ElektraIoFdOperation *fdOp)

  *Get callback of file descriptor watch operation.*
- int elektraIoTimerSetBindingData (ElektraIoTimerOperation *timerOp, void *data)

  *Set private binding data for operation.*
- void * elektraIoTimerGetBindingData (ElektraIoTimerOperation *timerOp)

  *Get private binding data from operation.*
- ElektraIoInterface * elektraIoTimerGetBinding (ElektraIoTimerOperation *timerOp)

  *Get binding from operation.*
- void * elektraIoTimerGetData (ElektraIoTimerOperation *timerOp)

  *Get private data from operation.*
- int elektraIoTimerIsEnabled (ElektraIoTimerOperation *timerOp)

  *Check if timer operation is enabled or disabled.*
- unsigned int elektraIoTimerGetInterval (ElektraIoTimerOperation *timerOp)

  *Get interval of timer operation.*
- ElektraIoTimerCallback elektraIoTimerGetCallback (ElektraIoTimerOperation *timerOp)

  *Get callback of timer operation.*
- int elektraIoIdleSetBindingData (ElektraIoIdleOperation *idleOp, void *data)

  *Set private binding data for operation.*
- void * elektraIoIdleGetBindingData (ElektraIoIdleOperation *idleOp)

  *Get private binding data from operation.*
- ElektraIoInterface * elektraIoIdleGetBinding (ElektraIoIdleOperation *idleOp)

  *Get binding from operation.*
- void * elektraIoIdleGetData (ElektraIoIdleOperation *idleOp)

  *Get private data from operation.*
- int elektraIoIdleIsEnabled (ElektraIoIdleOperation *idleOp)

  *Check if idle operation is enabled or disabled.*
- ElektraIoIdleCallback elektraIoIdleGetCallback (ElektraIoIdleOperation *idleOp)

  *Get callback of idle operation.*

## 573.49.1 Detailed Description

Implementation of I/O functions as defined in kdbio.h.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.49.2 Function Documentation

### 573.49.2.1 elektraIoBindingAddFd()

```
int elektraIoBindingAddFd (
            ElektraIoInterface * binding,
            ElektraIoFdOperation * fdOp )
```

Add file descriptor to be watched by I/O binding.
An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.2 elektraIoBindingAddIdle()

```
int elektraIoBindingAddIdle (
            ElektraIoInterface * binding,
            ElektraIoIdleOperation * idleOp )
```

Add idle to I/O binding.
Idle callbacks are executed without negative effects on other IO sources or the application (e.g. next event loop iteration) An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.3 elektraIoBindingAddTimer()

```
int elektraIoBindingAddTimer (
            ElektraIoInterface * binding,
            ElektraIoTimerOperation * timerOp )
```

Add timer to I/O binding.
Timeouts callbacks are executed after the initial interval has elapsed and then repeatedly after the interval has elapsed. An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.4 elektraIoBindingCleanup()

```
int elektraIoBindingCleanup (
            ElektraIoInterface * binding )
```

Free memory used by I/O binding.
All added operations have to be removed before calling this function.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.5 elektraIoBindingGetData()

```
void* elektraIoBindingGetData (
            ElektraIoInterface * binding )
```

Get private data from I/O Binding.
To be used by I/O binding implementations only.

**Parameters**

| | |
|---|---|
| *binding* | I/O-Binding handle |

**Returns**

pointer to data or NULL on error

#### 573.49.2.6 elektraIoBindingRemoveFd()

```
int elektraIoBindingRemoveFd (
            ElektraIoFdOperation * fdOp )
```

Remove file descriptor from I/O binding.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.7 elektraIoBindingRemoveIdle()

```
int elektraIoBindingRemoveIdle (
            ElektraIoIdleOperation * idleOp )
```

Remove idle from I/O binding.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.8 elektraIoBindingRemoveTimer()

```
int elektraIoBindingRemoveTimer (
            ElektraIoTimerOperation * timerOp )
```

Remove timer from I/O binding.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.9 elektraIoBindingSetData()

```
int elektraIoBindingSetData (
            ElektraIoInterface * binding,
            void * data )
```

Set private data from I/O Binding.
To be used by I/O binding implementations only.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *data* | private data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.49.2.10 elektraIoBindingUpdateFd()

```
int elektraIoBindingUpdateFd (
            ElektraIoFdOperation * fdOp )
```

Notify I/O binding about changes to file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.11 elektraIoBindingUpdateIdle()

```
int elektraIoBindingUpdateIdle (
            ElektraIoIdleOperation * idleOp )
```

Notifiy I/O binding about changes to idle structure.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.12 elektraIoBindingUpdateTimer()

```
int elektraIoBindingUpdateTimer (
            ElektraIoTimerOperation * timerOp )
```

Notifiy I/O binding about changes to timer structure.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.13 elektraIoContract()

```
int elektraIoContract (
            KeySet * contract,
            ElektraIoInterface * ioBinding )
```

Creates a contract for use with kdbOpen() that sets up an I/O binding.
When you call kdbOpen() with this contract, the KDB instance will use `ioBinding` as its I/O binding.

**Parameters**

| | |
|---|---|
| *contract* | The keyset into which the contract is written. |
| *ioBinding* | The ioBinding to use. |

**Return values**

| | |
|---|---|
| *-1* | if `contract` or `ioBinding` are NULL |
| *0* | on success |

### 573.49.2.14 elektraIoFdGetBinding()

```
ElektraIoInterface* elektraIoFdGetBinding (
            ElektraIoFdOperation * fdOp )
```

Get binding from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | fd operation handle |

**Returns**

pointer to binding or NULL on error

### 573.49.2.15 elektraIoFdGetBindingData()

```
void* elektraIoFdGetBindingData (
            ElektraIoFdOperation * fdOp )
```

Get private binding data from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

pointer to data or NULL on error

### 573.49.2.16 elektraIoFdGetCallback()

```
ElektraIoFdCallback elektraIoFdGetCallback (
            ElektraIoFdOperation * fdOp )
```

Get callback of file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

callback

### 573.49.2.17 elektraIoFdGetData()

```
void* elektraIoFdGetData (
            ElektraIoFdOperation * fdOp )
```

Get private data from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

pointer to data or NULL on error

### 573.49.2.18 elektraIoFdGetFd()

```
int elektraIoFdGetFd (
            ElektraIoFdOperation * fdOp )
```

Get file descriptor number from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

file descriptor number or 0 on error

### 573.49.2.19 elektraIoFdGetFlags()

```
int elektraIoFdGetFlags (
            ElektraIoFdOperation * fdOp )
```

Get flag bitmask of file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

watch flag bitmask (see ElektraIoFdFlags).

### 573.49.2.20 elektraIoFdIsEnabled()

```
int elektraIoFdIsEnabled (
            ElektraIoFdOperation * fdOp )
```

Check if file descriptor watch operation is enabled or disabled.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *0* | if disabled |
| *1* | if enabled |

### 573.49.2.21 elektraIoFdSetBindingData()

```
int elektraIoFdSetBindingData (
            ElektraIoFdOperation * fdOp,
            void * data )
```

Set private binding data for operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |
| *data* | pointer to data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.22 elektraIoFdSetEnabled()

```
int elektraIoFdSetEnabled (
            ElektraIoFdOperation * fdOp,
            int enabled )
```

Enable or disable file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |
| *enabled* | 0 to disabled, any other value for enabled |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.23 elektraIoFdSetFlags()

```
int elektraIoFdSetFlags (
            ElektraIoFdOperation * fdOp,
            int flags )
```

Update flag bitmask of file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |
| *flags* | watch flag bitmask (see ElektraIoFdFlags). |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.24 elektraIoGetBinding()

```
ElektraIoInterface* elektraIoGetBinding (
            KDB * kdb )
```

Get I/O binding for asynchronous I/O operations for KDB instance.
Returns NULL if no I/O binding was set.

**Parameters**

| | |
|---|---|
| *kdb* | KDB instance |

**Returns**

I/O binding or NULL

### 573.49.2.25 elektraIoIdleGetBinding()

```
ElektraIoInterface* elektraIoIdleGetBinding (
            ElektraIoIdleOperation * idleOp )
```

Get binding from operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

pointer to binding or NULL on error

### 573.49.2.26 elektraIoIdleGetBindingData()

```
void* elektraIoIdleGetBindingData (
            ElektraIoIdleOperation * idleOp )
```

Get private binding data from operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

pointer to data or NULL on error

#### 573.49.2.27 elektraIoIdleGetCallback()

```
ElektraIoIdleCallback elektraIoIdleGetCallback (
            ElektraIoIdleOperation * idleOp )
```

Get callback of idle operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

callback

#### 573.49.2.28 elektraIoIdleGetData()

```
void* elektraIoIdleGetData (
            ElektraIoIdleOperation * idleOp )
```

Get private data from operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

pointer to data or NULL on error

#### 573.49.2.29 elektraIoIdleIsEnabled()

```
int elektraIoIdleIsEnabled (
            ElektraIoIdleOperation * idleOp )
```

Check if idle operation is enabled or disabled.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *0* | if disabled |
| *1* | if enabled |

#### 573.49.2.30 elektraIoIdleSetBindingData()

```
int elektraIoIdleSetBindingData (
```

```
            ElektraIoIdleOperation * idleOp,
            void * data )
```

Set private binding data for operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |
| *data* | pointer to data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.31 elektraIoIdleSetEnabled()

```
int elektraIoIdleSetEnabled (
            ElektraIoIdleOperation * idleOp,
            int enabled )
```

Enable or disable idle operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |
| *enabled* | 0 to disabled, any other value for enabled |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.32 elektraIoNewBinding()

```
ElektraIoInterface* elektraIoNewBinding (
            ElektraIoBindingAddFd * addFd,
            ElektraIoBindingUpdateFd * updateFd,
            ElektraIoBindingRemoveFd * removeFd,
            ElektraIoBindingAddTimer * addTimer,
            ElektraIoBindingUpdateTimer * updateTimer,
            ElektraIoBindingRemoveTimer * removeTimer,
            ElektraIoBindingAddIdle * addIdle,
            ElektraIoBindingUpdateIdle * updateIdle,
            ElektraIoBindingRemoveIdle * removeIdle,
            ElektraIoBindingCleanup * cleanup )
```

Create a new I/O binding.
Make sure to free returned data in ElektraIoBindingCleanup.

**Parameters**

| | |
|---|---|
| *addFd* | function for adding a file descriptor watch operation |
| *updateFd* | function for updating a file descriptor operation |

**Parameters**

| | |
|---|---|
| *removeFd* | function for removing a file descriptor operation |
| *addTimer* | function for adding a timer operation |
| *updateTimer* | function for updateing a timer operation |
| *removeTimer* | function for removing a timer operation |
| *addIdle* | function for adding an idle operation |
| *updateIdle* | function for updating an idle operation |
| *removeIdle* | function for removing an idle operation |
| *cleanup* | function for cleaning up binding data |

**Returns**

newly created binding

### 573.49.2.33 elektraIoNewFdOperation()

```
ElektraIoFdOperation* elektraIoNewFdOperation (
            int fd,
            int flags,
            int enabled,
            ElektraIoFdCallback callback,
            void * data )
```

Create a new file descriptor watch operation.
Free returned data after use.

**Parameters**

| | |
|---|---|
| *fd* | file descriptor number |
| *flags* | watch flag bitmask (see ElektraIoFdFlags). Select on which file descriptor state changes the callback should be invoked |
| *enabled* | 0 to disabled, any other value for enabled |
| *callback* | Called when file descriptor state has changes |
| *data* | Custom private data |

**Returns**

file descriptor operation handle

### 573.49.2.34 elektraIoNewIdleOperation()

```
ElektraIoIdleOperation* elektraIoNewIdleOperation (
            int enabled,
            ElektraIoIdleCallback callback,
            void * data )
```

Create a new idle operation.
Free returned data after use.

**Parameters**

| | |
|---|---|
| *enabled* | 0 to disable, any other value for enabled |
| *callback* | Called when file descriptor state has changes |
| *data* | Custom private data |

**Returns**

idle operation handle

### 573.49.2.35 elektraIoNewTimerOperation()

```
ElektraIoTimerOperation* elektraIoNewTimerOperation (
            unsigned int interval,
            int enabled,
            ElektraIoTimerCallback callback,
            void * data )
```

Create a new timer operation.
Free returned data after use.

**Parameters**

| | |
|---|---|
| *interval* | timer interval in milliseconds |
| *enabled* | 0 to disable, any other value for enabled |
| *callback* | Called when file descriptor state has changes |
| *data* | Custom private data |

**Returns**

timer operation handle

### 573.49.2.36 elektraIoTimerGetBinding()

```
ElektraIoInterface* elektraIoTimerGetBinding (
            ElektraIoTimerOperation * timerOp )
```

Get binding from operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

pointer to binding or NULL on error

### 573.49.2.37 elektraIoTimerGetBindingData()

```
void* elektraIoTimerGetBindingData (
            ElektraIoTimerOperation * timerOp )
```

Get private binding data from operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

pointer to data or NULL on error

#### 573.49.2.38 elektraIoTimerGetCallback()

```
ElektraIoTimerCallback elektraIoTimerGetCallback (
            ElektraIoTimerOperation * timerOp )
```

Get callback of timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

callback

#### 573.49.2.39 elektraIoTimerGetData()

```
void* elektraIoTimerGetData (
            ElektraIoTimerOperation * timerOp )
```

Get private data from operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

pointer to data or NULL on error

#### 573.49.2.40 elektraIoTimerGetInterval()

```
unsigned int elektraIoTimerGetInterval (
            ElektraIoTimerOperation * timerOp )
```

Get interval of timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

timer interval in milliseconds, 0 on error

#### 573.49.2.41 elektraIoTimerIsEnabled()

```
int elektraIoTimerIsEnabled (
            ElektraIoTimerOperation * timerOp )
```

Check if timer operation is enabled or disabled.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *0* | if disabled |
| *1* | if enabled |

### 573.49.2.42 elektraIoTimerSetBindingData()

```
int elektraIoTimerSetBindingData (
            ElektraIoTimerOperation * timerOp,
            void * data )
```

Set private binding data for operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |
| *data* | pointer to data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.43 elektraIoTimerSetEnabled()

```
int elektraIoTimerSetEnabled (
            ElektraIoTimerOperation * timerOp,
            int enabled )
```

Enable or disable timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |
| *enabled* | 0 to disabled, any other value for enabled |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.49.2.44 elektraIoTimerSetInterval()

```
int elektraIoTimerSetInterval (
            ElektraIoTimerOperation * timerOp,
            unsigned int interval )
```

Update interval of timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Parameters**

| | |
|---|---|
| *interval* | timer interval in milliseconds |

**Return values**

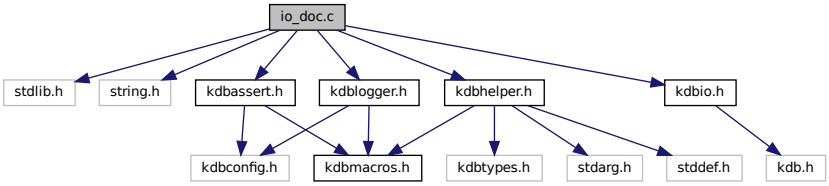
| | |
|---|---|
| *1* | on success |
| *0* | on error |

## 573.50 io_doc.c File Reference

I/O example binding.

```
#include <stdlib.h>
#include <string.h>
#include <kdbassert.h>
#include <kdbhelper.h>
#include <kdbio.h>
#include <kdblogger.h>
```

Include dependency graph for io_doc.c:




### Classes

- struct SomeIoLibHandle

  *Example I/O management library data structure.*
- struct DocOperationData

  *[kdbio operation data]*
- struct DocBindingData

  *[kdbio operation data]*

### Typedefs

- typedef struct SomeIoLibHandle SomeIoLibHandle

  *Example I/O management library data structure.*
- typedef struct DocOperationData DocOperationData

  *[kdbio operation data]*
- typedef struct DocBindingData DocBindingData

  *[kdbio operation data]*

### Enumerations

- enum SomeIoLibFlags { SOME_IOLIB_READABLE = 1 << 0 , SOME_IOLIB_WRITABLE = 1 << 1 }

  *Example I/O mangement library bitmask flags.*

## Functions

- DocOperationData * newOperationData (void)

  *[kdbio binding data]*
- int someBitMaskToElektraIoFlags (int bitmask)

  *Convert your I/O library bit mask to Elektra's I/O flags.*
- void ioDocBindingFdCallback (SomeIoLibHandle *handle, int bitmask)

  *Calls the associated operation callback.*
- void ioDocBindingTimerCallback (SomeIoLibHandle *handle)

  *[kdbio operation callback]*
- void ioDocBindingIdleCallback (SomeIoLibHandle *handle)

  *Calls the associated operation callback.*
- int ioDocBindingUpdateFd (ElektraIoFdOperation *fdOp)

  *Update information about a file descriptor watched by I/O binding.*
- int ioDocBindingAddFd (ElektraIoInterface *binding, ElektraIoFdOperation *fdOp)

  *Add file descriptor to I/O binding.*
- int ioDocBindingRemoveFd (ElektraIoFdOperation *fdOp)

  *Remove file descriptor from I/O binding.*
- int ioDocBindingUpdateTimer (ElektraIoTimerOperation *timerOp)

  *Update timer in I/O binding.*
- int ioDocBindingAddTimer (ElektraIoInterface *binding, ElektraIoTimerOperation *timerOp)

  *Add timer for I/O binding.*
- int ioDocBindingRemoveTimer (ElektraIoTimerOperation *timerOp)

  *Remove timer from I/O binding.*
- int ioDocBindingUpdateIdle (ElektraIoIdleOperation *idleOp)

  *Update idle operation in I/O binding.*
- int ioDocBindingAddIdle (ElektraIoInterface *binding, ElektraIoIdleOperation *idleOp)

  *Add idle operation to I/O binding.*
- int ioDocBindingRemoveIdle (ElektraIoIdleOperation *idleOp)

  *Remove idle operation from I/O binding.*
- int ioDocBindingCleanup (ElektraIoInterface *binding)

  *Cleanup.*
- ElektraIoInterface * elektraIoDocNew (char *foo)

  *Create and initialize a new doc I/O binding.*

### 573.50.1 Detailed Description

I/O example binding.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

All functions except the I/O binding entry point (elektraIoDocNew()) would normally be static but for documentation purposes they are not.

### 573.50.2 Typedef Documentation

#### 573.50.2.1 DocBindingData

```
typedef struct DocBindingData DocBindingData
```

[kdbio operation data]
[kdbio binding data] Container for additional information for an I/O binding.

#### 573.50.2.2 DocOperationData

```
typedef struct DocOperationData DocOperationData
```
[kdbio operation data]
Container for additional information for I/O binding operations.
It is helpful to create a data structure for your binding to store additional data

### 573.50.3 Enumeration Type Documentation

#### 573.50.3.1 SomeIoLibFlags

```
enum SomeIoLibFlags
```
Example I/O mangement library bitmask flags.

**Enumerator**

| | |
|---|---|
| SOME_IOLIB_READABLE | indicates that the file descriptor is readable |
| SOME_IOLIB_WRITABLE | indicates that the file descriptor is readable |

### 573.50.4 Function Documentation

#### 573.50.4.1 elektraIoDocNew()

```
ElektraIoInterface* elektraIoDocNew (
            char * foo )
```
Create and initialize a new doc I/O binding.

**Parameters**

| | |
|---|---|
| *foo* | Some data from I/O management library (e.g. a handle) |

**Returns**

Populated I/O interface

[kdbio binding create]
[kdbio binding create] [kdbio binding setdata]
[kdbio binding setdata]

#### 573.50.4.2 ioDocBindingAddFd()

```
int ioDocBindingAddFd (
            ElektraIoInterface * binding,
            ElektraIoFdOperation * fdOp )
```
Add file descriptor to I/O binding.

**See also**

kdbio.h ElektraIoBindingAddFd

[kdbio binding addfd]
[kdbio binding addfd]

#### 573.50.4.3 ioDocBindingAddIdle()

```
int ioDocBindingAddIdle (
            ElektraIoInterface * binding,
            ElektraIoIdleOperation * idleOp )
```

Add idle operation to I/O binding.

**See also**

kdbio.h ElektraIoBindingAddIdle

#### 573.50.4.4 ioDocBindingAddTimer()

```
int ioDocBindingAddTimer (
            ElektraIoInterface * binding,
            ElektraIoTimerOperation * timerOp )
```

Add timer for I/O binding.

**See also**

kdbio.h ElektraIoBindingAddTimer

#### 573.50.4.5 ioDocBindingCleanup()

```
int ioDocBindingCleanup (
            ElektraIoInterface * binding )
```

Cleanup.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding |

**See also**

kdbio.h ElektraIoBindingCleanup

#### 573.50.4.6 ioDocBindingFdCallback()

```
void ioDocBindingFdCallback (
            SomeIoLibHandle * handle,
            int bitmask )
```

Calls the associated operation callback.
Called by your I/O management library whenever a file descriptor status has changed.

**Parameters**

| | |
|---|---|
| *handle* | some I/O management handle |
| *flags* | flags bit mask [kdbio operation callback] |

#### 573.50.4.7 ioDocBindingIdleCallback()

```
void ioDocBindingIdleCallback (
            SomeIoLibHandle * handle )
```

Calls the associated operation callback.
Called by your I/O management library whenever an idle operation can run without interfering with other operations.

**Parameters**

| | |
|---|---|
| *handle* | some I/O management handle |

#### 573.50.4.8 ioDocBindingRemoveFd()

```
int ioDocBindingRemoveFd (
            ElektraIoFdOperation * fdOp )
```

Remove file descriptor from I/O binding.

**See also**

kdbio.h ElektraIoBindingRemoveFd

[kdbio operation getdata]
[kdbio operation getdata]

#### 573.50.4.9 ioDocBindingRemoveIdle()

```
int ioDocBindingRemoveIdle (
            ElektraIoIdleOperation * idleOp )
```

Remove idle operation from I/O binding.

**See also**

kdbio.h ElektraIoBindingRemoveIdle

#### 573.50.4.10 ioDocBindingRemoveTimer()

```
int ioDocBindingRemoveTimer (
            ElektraIoTimerOperation * timerOp )
```

Remove timer from I/O binding.

**See also**

kdbio.h ElektraIoBindingRemoveTimer

#### 573.50.4.11 ioDocBindingTimerCallback()

```
void ioDocBindingTimerCallback (
            SomeIoLibHandle * handle )
```

[kdbio operation callback]
Calls the associated operation callback. Called by your I/O management library whenever a timer interval has passed.

**Parameters**

| | |
|---|---|
| *handle* | some I/O management handle |

#### 573.50.4.12 ioDocBindingUpdateFd()

```
int ioDocBindingUpdateFd (
            ElektraIoFdOperation * fdOp )
```

Update information about a file descriptor watched by I/O binding.

**See also**

kdbio.h ElektraIoBindingUpdateFd

#### 573.50.4.13 ioDocBindingUpdateIdle()

```
int ioDocBindingUpdateIdle (
            ElektraIoIdleOperation * idleOp )
```

Update idle operation in I/O binding.

**See also**

kdbio.h ElektraIoBindingUpdateIdle

#### 573.50.4.14 ioDocBindingUpdateTimer()

```
int ioDocBindingUpdateTimer (
            ElektraIoTimerOperation * timerOp )
```

Update timer in I/O binding.

**See also**

kdbio.h ElektraIoBindingUpdateTimer

#### 573.50.4.15 newOperationData()

```
DocOperationData* newOperationData (
            void  )
```

[kdbio binding data]
A little helper function for allocating our custom operation data structure

**Returns**

New data structure, remember to call elektraFree()

#### 573.50.4.16 someBitMaskToElektraIoFlags()

```
int someBitMaskToElektraIoFlags (
            int bitmask )
```

Convert your I/O library bit mask to Elektra's I/O flags.

**Parameters**

| | |
|---|---|
| *bitmask* | bit mask from I/O management library |

**Returns**

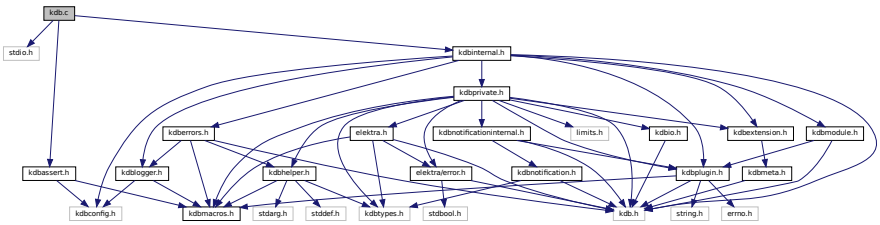
bit mask with Elektra's I/O flags (ElektraIoFdFlags)

## 573.51 kdb.c File Reference

Low level functions for access the Key Database.

```
#include <stdio.h>
#include <kdbassert.h>
#include <kdbchangetracking.h>
#include <kdbinternal.h>
```

Include dependency graph for kdb.c:




### Functions

- KeySet * ksRenameKeys (KeySet *config, const char *name)

  *Takes the first key and cuts off this common part for all other keys, instead name will be prepended.*
- KDB * kdbOpen (const KeySet *contract, Key *errorKey)

  *Opens the session with the Key database.*
- int kdbClose (KDB *handle, Key *errorKey)

  *Closes the session with the Key database.*
- int kdbGet (KDB *handle, KeySet *ks, Key *parentKey)

  *Retrieve Keys from the Key database in an atomic and universal way.*
- int kdbSet (KDB *handle, KeySet *ks, Key *parentKey)

  *Set Keys to the Key database in an atomic and universal way.*

### 573.51.1 Detailed Description

Low level functions for access the Key Database.

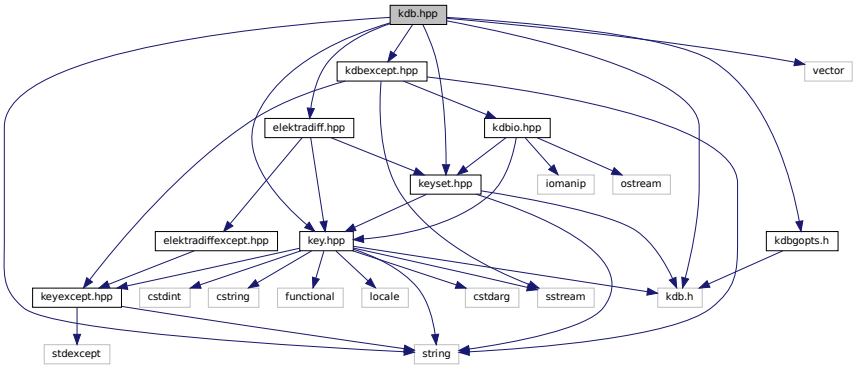
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

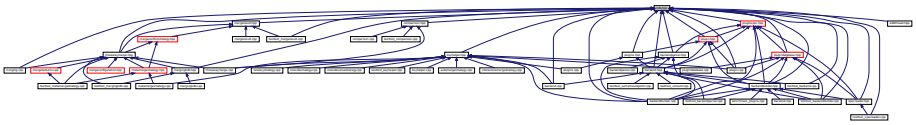
## 573.52 kdb.hpp File Reference

```
#include <string>
#include <vector>
#include <elektradiff.hpp>
#include <kdbexcept.hpp>
#include <key.hpp>
#include <keyset.hpp>
#include <kdb.h>
#include <kdbchangetracking.h>
```

`#include <kdbgopts.h>`
Include dependency graph for kdb.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- class kdb::KDB

  *Constructs a class KDB.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

## Functions

- int kdb::goptsContract (kdb::KeySet &contract, int argc, const char ∗const ∗argv, const char ∗const ∗envp, const kdb::Key &parentKey, kdb::KeySet &goptsConfig)
- int kdb::goptsContract (kdb::KeySet &contract, const std::string &argsString, const std::string &envString, const kdb::Key &parentKey, kdb::KeySet &goptsConfig)

  *Prefer to use goptsContract with argc, argv and envp if possible (especially when you are calling this in your main function)*
- int kdb::goptsContract (kdb::KeySet &contract, const std::vector< std::string > &args, const std::vector< std::string > &env, const kdb::Key &parentKey, kdb::KeySet &goptsConfig)

  *Prefer to use goptsContract with argc, argv and envp if possible (especially when you are calling this in your main function)*

### 573.52.1 Detailed Description

**Copyright**

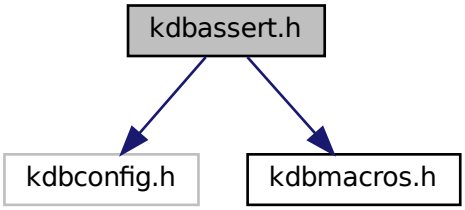
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.53 kdbassert.h File Reference

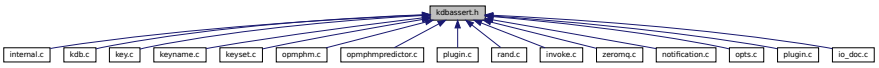
Assertions macros.

```
#include <kdbconfig.h>
#include <kdbmacros.h>
```

Include dependency graph for kdbassert.h:




This graph shows which files directly or indirectly include this file:




### 573.53.1 Detailed Description

Assertions macros.

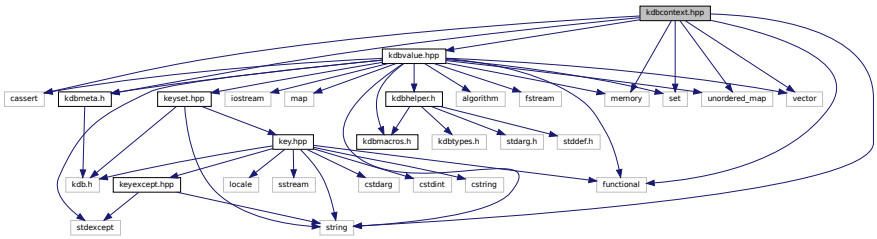
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

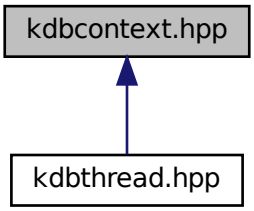
## 573.54 kdbcontext.hpp File Reference

```
#include <kdbmeta.h>
#include <kdbvalue.hpp>
#include <cassert>
#include <functional>
#include <memory>
#include <set>
#include <string>
#include <unordered_map>
#include <vector>
```

Include dependency graph for kdbcontext.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::Context

  *Provides a context for configuration.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.54.1 Detailed Description

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.55 kdbenum.c File Reference

dummy file to document the enums which have no name in the header file.

## Macros

- #define KDB_VERSION "x.y.z"

  *The version information in x.y.z format as string.*
- #define KDB_VERSION_MAJOR x

  *The version information of the major version as number.*
- #define KDB_VERSION_MINOR y

*The version information of the minor version as number.*

- #define KDB_VERSION_PATCH z

  *The version information of the patch version as number.*

- #define KDB_PATH_SEPARATOR '/'

  */ is used to separate key names.*

- #define KDB_PATH_ESCAPE '\\'

  *\ is used as escape character in the key name.*

- #define ELEKTRA_ENABLE_OPTIMIZATIONS

  *If optimizations are enabled in Elektra.*

- #define KS_END ((Key *) 0)

  *End of a list of keys.*

### Enumerations

- enum elektraKeyFlags {
  KEY_VALUE = 1 << 1 , KEY_FLAGS = 3 , KEY_BINARY = 1 << 4 , KEY_SIZE = 1 << 11 ,
  KEY_META = 1 << 15 , KEY_NULL = 1 << 16 , KEY_END = 0 }

  *Allows keyNew() to determine which information comes next.*

- enum elektraCopyFlags {
  KEY_CP_NAME = 1 << 0 , KEY_CP_STRING = 1 << 1 , KEY_CP_VALUE = 1 << 2 , KEY_CP_META = 1 << 3 ,
  KEY_CP_ALL = KEY_CP_NAME | KEY_CP_VALUE | KEY_CP_META }

  *Copy options.*

- enum elektraLockFlags { KEY_LOCK_NAME = 1 << 17 , KEY_LOCK_VALUE = 1 << 18 , KEY_LOCK_META = 1 << 19 }

  *Lock options.*

- enum elektraNamespace {
  KEY_NS_NONE = 0 , KEY_NS_CASCADING = 1 , KEY_NS_META = 2 , KEY_NS_SPEC = 3 ,
  KEY_NS_PROC = 4 , KEY_NS_DIR = 5 , KEY_NS_USER = 6 , KEY_NS_SYSTEM = 7 ,
  KEY_NS_DEFAULT = 8 }

  *Elektra currently supported Key namespaces.*

- enum elektraLookupFlags { KDB_O_NONE = 0 , KDB_O_DEL = 1 , KDB_O_POP = 1 << 1 }

  *Options to change the default behavior of ksLookup() functions.*

#### 573.55.1 Detailed Description

dummy file to document the enums which have no name in the header file.
They are duplicated here to document them.

**Copyright**



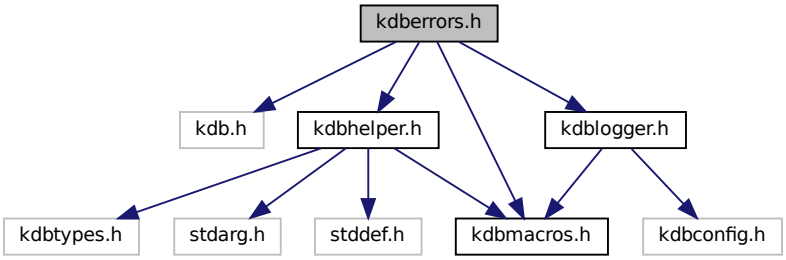
## 573.56 kdberrors.h File Reference

Provides all macros and definitions which are used for emitting error or warnings.

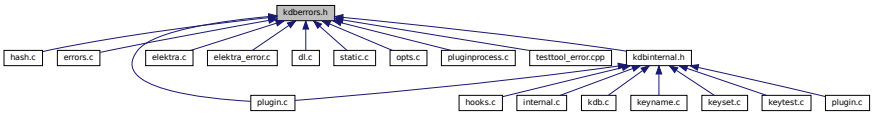
```
#include <kdb.h>
#include <kdbhelper.h>
#include <kdblogger.h>
```

```
#include <kdbmacros.h>
```

Include dependency graph for kdberrors.h:




This graph shows which files directly or indirectly include this file:




## Functions

- void elektraCopyError (Key ∗target, Key ∗source)

  *Copy the error from the source key to target key.*
- void elektraCopyWarnings (Key ∗target, Key ∗source)

  *Copy the warnings from the source key to target key.*
- void elektraCopyErrorAndWarnings (Key ∗target, Key ∗source)

  *Copies the error and warnings from the source key to the target key.*

### 573.56.1 Detailed Description

Provides all macros and definitions which are used for emitting error or warnings.

#### Copyright

BSD License (see doc/LICENSE.md or http://www.libelektra.org)

### 573.56.2 Function Documentation

#### 573.56.2.1 elektraCopyError()

```
void elektraCopyError (
            Key * target,
            Key * source )
```

Copy the error from the source key to target key.

#### Parameters

| | |
|---|---|
| *target* | append error to this key |
| *source* | copy error from this key |

#### 573.56.2.2 elektraCopyErrorAndWarnings()

```
void elektraCopyErrorAndWarnings (
            Key * target,
            Key * source )
```

Copies the error and warnings from the source key to the target key.
Note that only 100 warnings can be had. If we exceed that, we'll override the existing warnings in the target key

**Parameters**

| | |
|---|---|
| *target* | copy error and warnings to this key |
| *source* | copy error and warnings from this key |

#### 573.56.2.3 elektraCopyWarnings()

```
void elektraCopyWarnings (
            Key * target,
            Key * source )
```

Copy the warnings from the source key to target key.
Note that only 100 warnings can be had. If we exceed that, we'll override the existing warnings in the target key.

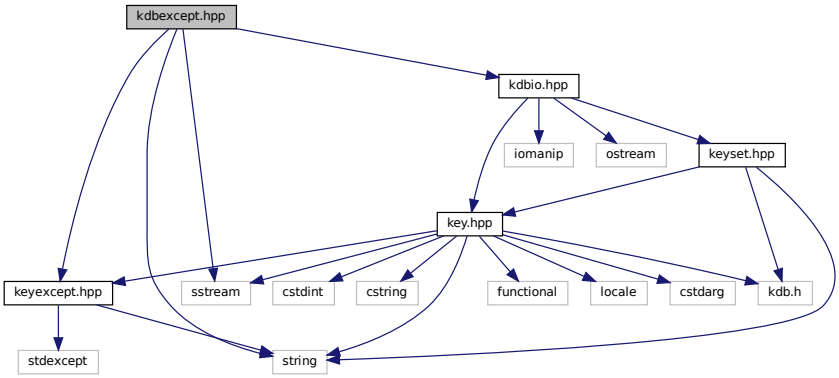
**Parameters**

| | |
|---|---|
| *target* | append warnings to this key |
| *source* | copy warnings from this key |

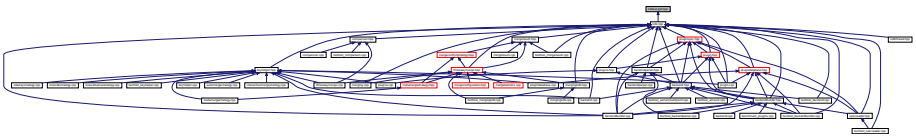
## 573.57 kdbexcept.hpp File Reference

```
#include <keyexcept.hpp>
#include <sstream>
#include <string>
#include <kdbio.hpp>
```

Include dependency graph for kdbexcept.hpp:



This graph shows which files directly or indirectly include this file:



### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.57.1 Detailed Description

**Copyright**

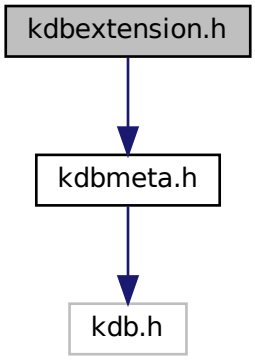
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.58 kdbextension.h File Reference

Extension functionality.

```
#include <kdbease.h>
#include <kdbmeta.h>
```

Include dependency graph for kdbextension.h:




This graph shows which files directly or indirectly include this file:

### 573.58.1 Detailed Description

Extension functionality.

**Copyright**

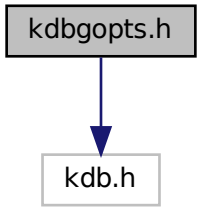
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.59 kdbgopts.h File Reference

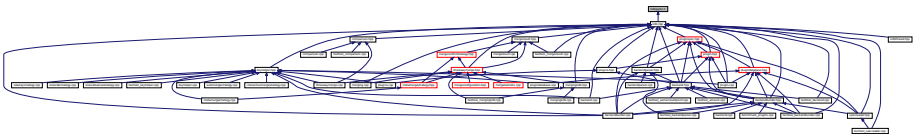
Gopts contract.

```
#include <kdb.h>
```

Include dependency graph for kdbgopts.h:




This graph shows which files directly or indirectly include this file:



### Functions

- int elektraGOptsContract (KeySet ∗contract, int argc, const char ∗const ∗argv, const char ∗const ∗envp, const Key ∗parentKey, KeySet ∗goptsConfig)

  *Sets up a contract for use with kdbOpen() that configures the gopts plugin.*
- int elektraGOptsContractFromStrings (KeySet ∗contract, size_t argsSize, const char ∗args, size_t envSize, const char ∗env, const Key ∗parentKey, KeySet ∗goptsConfig)

  *Sets up a contract for use with kdbOpen() that configures the gopts plugin.*

### 573.59.1 Detailed Description

Gopts contract.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.59.2 Function Documentation

#### 573.59.2.1 elektraGOptsContract()

```
int elektraGOptsContract (
            KeySet * contract,
```

```
            int argc,
            const char *const * argv,
            const char *const * envp,
            const Key * parentKey,
            KeySet * goptsConfig )
```

Sets up a contract for use with kdbOpen() that configures the gopts plugin.
You can pass 0 for `argc` **and** `NULL` for `argv` to let gopts lookup command line options internally. You can also pass `NULL` for `envp` to do the same for environment variables.

**Parameters**

| | |
|---|---|
| *contract* | The KeySet into which the contract will be written. |
| *argc* | The argc value that should be used by gopts. |
| *argv* | The argv value that should be used by gopts. IMPORTANT: The pointer and data behind must be valid until after kdbClose() is called. |
| *envp* | The envp value that should be used by gopts. IMPORTANT: The pointer and data behind must be valid until after kdbClose() is called. |
| *parentKey* | The parent key that should be used by gopts. Only the key name is copied. The key can be deleted immediately after calling this function. |
| *goptsConfig* | The config that used to mount the gopts plugin. This value can be NULL. Only keys in the user:/ namespace will be used. |

**Return values**

| | |
|---|---|
| *-1* | if `contract` or `parentKey` is NULL |
| *-1* | if `argc` is 0 and `argv` is not NULL or if `argc` is not 0 and `argv` is NULL |
| *0* | on success |

#### 573.59.2.2 elektraGOptsContractFromStrings()

```
int elektraGOptsContractFromStrings (
            KeySet * contract,
            size_t argsSize,
            const char * args,
            size_t envSize,
            const char * env,
            const Key * parentKey,
            KeySet * goptsConfig )
```

Sets up a contract for use with kdbOpen() that configures the gopts plugin.
NOTE: prefer to use elektraGOptsContract() if possible
You can pass `NULL` for `args` to let gopts lookup command line options internally. You can also pass `NULL` for `env` to do the same for environment variables.

**Parameters**

| | |
|---|---|
| *contract* | The KeySet into which the contract will be written. |
| *argsSize* | The size of the `args` data |
| *args* | Continuous buffer containing all argv arguments separated (and terminated) by zero bytes. The whole buffer is copied, so the pointer only has to be valid for this function call. |
| *envSize* | The size of the `env` data |
| *env* | Continuous buffer containing all environment variables separated (and terminated) by zero bytes The whole buffer is copied, so the pointer only has to be valid for this function call. |

**Parameters**

| | |
|---|---|
| *parentKey* | The parent key that should be used by gopts. Only the key name is copied. The key can be deleted immediately after calling this function. |
| *goptsConfig* | The config that used to mount the gopts plugin. This value can be NULL. Only keys in the user:/ namespace will be used. |

**Return values**

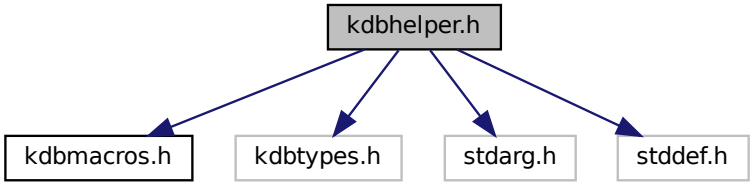
| | |
|---|---|
| *-1* | if any of `contract` or `parentKey` is NULL |
| *0* | on success |

## 573.60 kdbhelper.h File Reference

Helper for memory management.

```
#include <kdbmacros.h>
#include <kdbtypes.h>
#include <stdarg.h>
#include <stddef.h>
```

Include dependency graph for kdbhelper.h:




This graph shows which files directly or indirectly include this file:

### Enumerations

- enum elektraLookupOptions {
  KDB_O_SPEC = 1 << 15 , KDB_O_CREATE = 1 << 16 , KDB_O_NOCASCADING = 1 << 17 ,
  KDB_O_NOSPEC = 1 << 18 ,
  KDB_O_NODEFAULT = 1 << 19 , KDB_O_CALLBACK = 1 << 20 , KDB_O_OPMPHM = 1 << 21 ,
  KDB_O_BINSEARCH = 1 << 22 }

  *More lookup options.*

### Functions

- void * elektraMalloc (size_t size)

  *Allocate memory for Elektra.*

- void * elektraCalloc (size_t size)

  *Allocate memory for Elektra.*
- void elektraFree (void *ptr)

  *Free memory of Elektra or its backends.*
- int elektraRealloc (void **buffer, size_t size)

  *Reallocate Storage in a save way.*
- char * elektraStrDup (const char *s)

  *Copy string into new allocated memory.*
- void * elektraMemDup (const void *s, size_t l)

  *Copy buffer into new allocated memory.*
- char char * elektraVFormat (const char *format, va_list arg_list)

  *Does string formatting in fresh allocated memory.*
- int elektraStrCmp (const char *s1, const char *s2)

  *Compare Strings.*
- int elektraStrNCmp (const char *s1, const char *s2, size_t n)

  *Compare Strings up to a maximum length.*
- int elektraStrCaseCmp (const char *s1, const char *s2)

  *Compare Strings ignoring case.*
- int elektraStrNCaseCmp (const char *s1, const char *s2, size_t n)

  *Compare Strings ignoring case up to a maximum length.*
- int elektraMemCaseCmp (const char *s1, const char *s2, size_t size)

  *Compare two memory regions but make cmp chars uppercase before comparison.*
- size_t elektraStrLen (const char *s)

  *Calculates the length in bytes of a string.*

## 573.60.1 Detailed Description

Helper for memory management.
Should be always preferred. Can be used for profiling and tracing.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.60.2 Enumeration Type Documentation

### 573.60.2.1 elektraLookupOptions

enum elektraLookupOptions
More lookup options.

**Enumerator**

| | |
|---|---|
| KDB_O_SPEC | Use the passed key as specification, instead of looking up the specification first. |
| KDB_O_CREATE | Create the key if it was not found. |
| KDB_O_NOCASCADING | Disable cascading search for keys starting with /. |
| KDB_O_NOSPEC | Do not use specification for cascading keys (internal) |
| KDB_O_NODEFAULT | Do not honor the default spec (internal) |
| KDB_O_CALLBACK | For spec:/ lookups that traverse deeper into hierarchy (callback in ksLookup()) |
| KDB_O_OPMPHM | Overrule ksLookup search predictor to use OPMPHM, make sure to set ENABLE_OPTIMIZATIONS=ON at cmake. |
| KDB_O_BINSEARCH | Overrule ksLookup search predictor to use Binary search for lookup. |

### 573.60.3 Function Documentation

#### 573.60.3.1 elektraCalloc()

```
void* elektraCalloc (
            size_t size )
```

Allocate memory for Elektra.
Memory will be set to 0.

**Parameters**

| | |
|---|---|
| *size* | the requested size |

**See also**

elektraMalloc

#### 573.60.3.2 elektraFree()

```
void elektraFree (
            void * ptr )
```

Free memory of Elektra or its backends.

**Parameters**

| | |
|---|---|
| *ptr* | the pointer to free |

If ptr is NULL, no operation is performed.

**See also**

elektraMalloc

#### 573.60.3.3 elektraMalloc()

```
void* elektraMalloc (
            size_t size )
```

Allocate memory for Elektra.

```
if ((buffer = elektraMalloc (length)) == 0) {
        // here comes the failure handler
        // no allocation happened here, so don't use buffer
#if DEBUG
        fprintf (stderr, "Allocation error");
#endif
        // return with error
}
```

**Parameters**

| | |
|---|---|
| *size* | the requested size |

This function is compatible to ANSI-C malloc

**See also**

elektraFree

elektraCalloc

#### 573.60.3.4 elektraMemCaseCmp()

```
int elektraMemCaseCmp (
            const char * s1,
            const char * s2,
            size_t size )
```

Compare two memory regions but make cmp chars uppercase before comparison.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |
| *size* | to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.60.3.5 elektraMemDup()

```
void* elektraMemDup (
            const void * s,
            size_t n )
```

Copy buffer into new allocated memory.
You need to free the memory yourself.
This function also works with \0 characters in the buffer. The length is taken as given, it must be correct.

**Returns**

0 if out of memory, a pointer otherwise

**Parameters**

| | |
|---|---|
| *s* | must be an allocated buffer |
| *n* | the number of bytes to copy from s |

#### 573.60.3.6 elektraRealloc()

```
int elektraRealloc (
            void ** buffer,
            size_t size )
```

Reallocate Storage in a save way.

```
if (elektraRealloc ((void **) & buffer, new_length) < 0) {
```

```
        // here comes the failure handler
        // you can still use the old buffer
#if DEBUG
        fprintf (stderr, "Reallocation error\n");
#endif
        elektraFree (buffer);
        buffer = 0;
        // return with error
}
```

**Parameters**

| | |
|---|---|
| *buffer* | is a pointer to a elektraMalloc |
| *size* | is the new size for the memory |

**Return values**

| | |
|---|---|
| *-1* | on failure |
| *0* | on success |

#### 573.60.3.7 elektraStrCaseCmp()

```
int elektraStrCaseCmp (
            const char * s1,
            const char * s2 )
```

Compare Strings ignoring case.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.60.3.8 elektraStrCmp()

```
int elektraStrCmp (
            const char * s1,
            const char * s2 )
```

Compare Strings.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.60.3.9 elektraStrDup()

```
char* elektraStrDup (
            const char * s )
```

Copy string into new allocated memory.
You need to free the memory yourself.

**Note**

that size is determined at runtime. So if you have a size information, don't use that function.

**Parameters**

| | |
|---|---|
| *s* | the null-terminated string to duplicate |

**Returns**

0 if out of memory, a pointer otherwise

**Precondition**

s must be a c-string.

**See also**

elektraFree

elektraStrLen

elektraMemDup

#### 573.60.3.10 elektraStrLen()

```
size_t elektraStrLen (
            const char * s )
```

Calculates the length in bytes of a string.
This function differs from strlen() because it is Unicode and multibyte chars safe. While strlen() counts characters and ignores the final NULL, elektraStrLen() count bytes including the ending NULL.
It must not be used to search for / in the name, because it does not consider escaping. Instead use the unescaped name.

**See also**

keyUnescapedName()

**Parameters**

| | |
|---|---|
| *s* | the string to get the length from |

**Returns**

number of bytes used by the string, including the final NULL.

#### 573.60.3.11 elektraStrNCaseCmp()

```
int elektraStrNCaseCmp (
            const char * s1,
            const char * s2,
            size_t n )
```

Compare Strings ignoring case up to a maximum length.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |
| *n* | The maximum length to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.60.3.12 elektraStrNCmp()

```
int elektraStrNCmp (
            const char * s1,
            const char * s2,
            size_t n )
```

Compare Strings up to a maximum length.

**Parameters**

| | |
|---|---|
| *s1* | The first string to be compared |
| *s2* | The second string to be compared |
| *n* | The maximum length to be compared |

**Returns**

a negative number if s1 is less than s2

**Return values**

| | |
|---|---|
| *0* | if s1 matches s2 |

**Returns**

a positive number if s1 is greater than s2

#### 573.60.3.13 elektraVFormat()

```
char char* elektraVFormat (
            const char * format,
            va_list arg_list )
```

Does string formatting in fresh allocated memory.

**Parameters**

| | |
|---|---|
| *format* | as in vprintf() |
| *arg_list* | as in vprintf() |

**Returns**

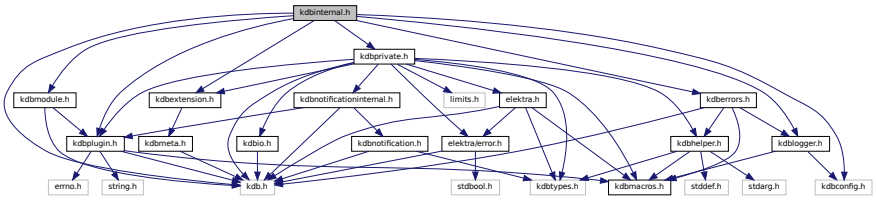
new allocated memory (free with elektraFree)

## 573.61 kdbinternal.h File Reference

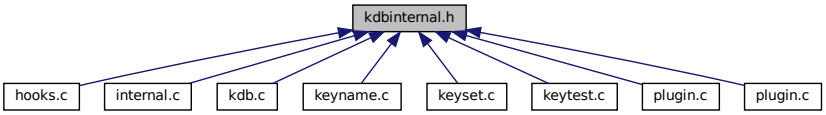
Includes most internal header files.

```
#include <kdb.h>
#include <kdbconfig.h>
#include <kdberrors.h>
#include <kdbextension.h>
#include <kdblogger.h>
#include <kdbmodule.h>
#include <kdbplugin.h>
#include <kdbprivate.h>
```

Include dependency graph for kdbinternal.h:



This graph shows which files directly or indirectly include this file:




### 573.61.1 Detailed Description

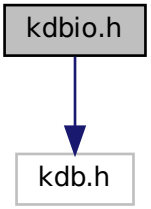
Includes most internal header files.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.62 kdbio.h File Reference

Elektra-I/O structures for I/O bindings, plugins and applications.
`#include "kdb.h"`
Include dependency graph for kdbio.h:




This graph shows which files directly or indirectly include this file:

### Typedefs

- typedef struct _ElektraIoInterface ElektraIoInterface
  *I/O binding handle.*
- typedef struct _ElektraIoFdOperation ElektraIoFdOperation
  *file descriptor watch operation handle*
- typedef struct _ElektraIoTimerOperation ElektraIoTimerOperation
  *timer operation handle*
- typedef struct _ElektraIoIdleOperation ElektraIoIdleOperation
  *idle operation handle*
- typedef void(∗ ElektraIoFdCallback) (ElektraIoFdOperation ∗fdOp, int flags)

*Callback for file descriptor watch operations.*

- typedef void(* ElektraIoIdleCallback) (ElektraIoIdleOperation *idleOp)

  *Callback for idle operations.*

- typedef void(* ElektraIoTimerCallback) (ElektraIoTimerOperation *timerOp)

  *Callback for timer operations.*

- typedef int ElektraIoBindingAddFd(ElektraIoInterface *binding, ElektraIoFdOperation *fdOp)

  *Add file descriptor to be watched by I/O binding.*

- typedef int ElektraIoBindingUpdateFd(ElektraIoFdOperation *fdOp)

  *Notify I/O binding about changes to file descriptor watch operation.*

- typedef int ElektraIoBindingRemoveFd(ElektraIoFdOperation *fdOp)

  *Remove file descriptor from I/O binding.*

- typedef int ElektraIoBindingAddTimer(ElektraIoInterface *binding, ElektraIoTimerOperation *timerOp)

  *Add timer to I/O binding.*

- typedef int ElektraIoBindingUpdateTimer(ElektraIoTimerOperation *timerOp)

  *Notifiy I/O binding about changes to timer structure.*

- typedef int ElektraIoBindingRemoveTimer(ElektraIoTimerOperation *timerOp)

  *Remove timer from I/O binding.*

- typedef int ElektraIoBindingAddIdle(ElektraIoInterface *binding, ElektraIoIdleOperation *idleOp)

  *Add idle to I/O binding.*

- typedef int ElektraIoBindingUpdateIdle(ElektraIoIdleOperation *idleOp)

  *Notifiy I/O binding about changes to idle structure.*

- typedef int ElektraIoBindingRemoveIdle(ElektraIoIdleOperation *idleOp)

  *Remove idle from I/O binding.*

- typedef int ElektraIoBindingCleanup(ElektraIoInterface *binding)

  *Free memory used by I/O binding.*

## Enumerations

- enum ElektraIoFdFlags { ELEKTRA_IO_READABLE = 1 << 0 , ELEKTRA_IO_WRITABLE = 1 << 1 }

  *Available flags for file descriptors operation bitmask.*

## Functions

- int elektraIoBindingAddFd (ElektraIoInterface *binding, ElektraIoFdOperation *fdOp)

  *Add file descriptor to be watched by I/O binding.*

- int elektraIoBindingUpdateFd (ElektraIoFdOperation *fdOp)

  *Notify I/O binding about changes to file descriptor watch operation.*

- int elektraIoBindingRemoveFd (ElektraIoFdOperation *fdOp)

  *Remove file descriptor from I/O binding.*

- int elektraIoBindingAddTimer (ElektraIoInterface *binding, ElektraIoTimerOperation *timerOp)

  *Add timer to I/O binding.*

- int elektraIoBindingUpdateTimer (ElektraIoTimerOperation *timerOp)

  *Notifiy I/O binding about changes to timer structure.*

- int elektraIoBindingRemoveTimer (ElektraIoTimerOperation *timerOp)

  *Remove timer from I/O binding.*

- int elektraIoBindingAddIdle (ElektraIoInterface *binding, ElektraIoIdleOperation *idleOp)

  *Add idle to I/O binding.*

- int elektraIoBindingUpdateIdle (ElektraIoIdleOperation *idleOp)

  *Notifiy I/O binding about changes to idle structure.*

- int elektraIoBindingRemoveIdle (ElektraIoIdleOperation *idleOp)

  *Remove idle from I/O binding.*

- int elektraIoBindingCleanup (ElektraIoInterface *binding)

*Free memory used by I/O binding.*

- ElektraIoInterface * elektraIoNewBinding (ElektraIoBindingAddFd *addFd, ElektraIoBindingUpdateFd *updateFd, ElektraIoBindingRemoveFd *removeFd, ElektraIoBindingAddTimer *addTimer, ElektraIoBindingUpdateTimer *updateTimer, ElektraIoBindingRemoveTimer *removeTimer, ElektraIoBindingAddIdle *addIdle, ElektraIoBindingUpdateIdle *updateIdle, ElektraIoBindingRemoveIdle *removeIdle, ElektraIoBindingCleanup *cleanup)

  *Create a new I/O binding.*
- void * elektraIoBindingGetData (ElektraIoInterface *binding)

  *Get private data from I/O Binding.*
- int elektraIoBindingSetData (ElektraIoInterface *binding, void *data)

  *Set private data from I/O Binding.*
- ElektraIoFdOperation * elektraIoNewFdOperation (int fd, int flags, int enabled, ElektraIoFdCallback callback, void *data)

  *Create a new file descriptor watch operation.*
- int elektraIoFdSetEnabled (ElektraIoFdOperation *fdOp, int enabled)

  *Enable or disable file descriptor watch operation.*
- int elektraIoFdSetFlags (ElektraIoFdOperation *fdOp, int flags)

  *Update flag bitmask of file descriptor watch operation.*
- int elektraIoFdGetFd (ElektraIoFdOperation *fdOp)

  *Get file descriptor number from operation.*
- void * elektraIoFdGetData (ElektraIoFdOperation *fdOp)

  *Get private data from operation.*
- int elektraIoFdSetBindingData (ElektraIoFdOperation *fdOp, void *data)

  *Set private binding data for operation.*
- void * elektraIoFdGetBindingData (ElektraIoFdOperation *fdOp)

  *Get private binding data from operation.*
- ElektraIoInterface * elektraIoFdGetBinding (ElektraIoFdOperation *fdOp)

  *Get binding from operation.*
- int elektraIoFdIsEnabled (ElektraIoFdOperation *fdOp)

  *Check if file descriptor watch operation is enabled or disabled.*
- int elektraIoFdGetFlags (ElektraIoFdOperation *fdOp)

  *Get flag bitmask of file descriptor watch operation.*
- ElektraIoFdCallback elektraIoFdGetCallback (ElektraIoFdOperation *fdOp)

  *Get callback of file descriptor watch operation.*
- ElektraIoTimerOperation * elektraIoNewTimerOperation (unsigned int interval, int enabled, ElektraIoTimerCallback callback, void *data)

  *Create a new timer operation.*
- int elektraIoTimerSetEnabled (ElektraIoTimerOperation *timerOp, int enabled)

  *Enable or disable timer operation.*
- int elektraIoTimerIsEnabled (ElektraIoTimerOperation *timerOp)

  *Check if timer operation is enabled or disabled.*
- int elektraIoTimerSetInterval (ElektraIoTimerOperation *timerOp, unsigned int interval)

  *Update interval of timer operation.*
- unsigned int elektraIoTimerGetInterval (ElektraIoTimerOperation *timerOp)

  *Get interval of timer operation.*
- int elektraIoTimerSetBindingData (ElektraIoTimerOperation *timerOp, void *data)

  *Set private binding data for operation.*
- void * elektraIoTimerGetBindingData (ElektraIoTimerOperation *timerOp)

  *Get private binding data from operation.*
- ElektraIoInterface * elektraIoTimerGetBinding (ElektraIoTimerOperation *timerOp)

  *Get binding from operation.*
- void * elektraIoTimerGetData (ElektraIoTimerOperation *timerOp)

*Get private data from operation.*

- ElektraIoTimerCallback elektraIoTimerGetCallback (ElektraIoTimerOperation *timerOp)

  *Get callback of timer operation.*
- ElektraIoIdleOperation * elektraIoNewIdleOperation (int enabled, ElektraIoIdleCallback callback, void *data)

  *Create a new idle operation.*
- int elektraIoIdleSetEnabled (ElektraIoIdleOperation *idleOp, int enabled)

  *Enable or disable idle operation.*
- int elektraIoIdleIsEnabled (ElektraIoIdleOperation *idleOp)

  *Check if idle operation is enabled or disabled.*
- int elektraIoIdleSetBindingData (ElektraIoIdleOperation *idleOp, void *data)

  *Set private binding data for operation.*
- void * elektraIoIdleGetBindingData (ElektraIoIdleOperation *idleOp)

  *Get private binding data from operation.*
- ElektraIoInterface * elektraIoIdleGetBinding (ElektraIoIdleOperation *idleOp)

  *Get binding from operation.*
- void * elektraIoIdleGetData (ElektraIoIdleOperation *idleOp)

  *Get private data from operation.*
- ElektraIoIdleCallback elektraIoIdleGetCallback (ElektraIoIdleOperation *idleOp)

  *Get callback of idle operation.*
- int elektraIoContract (KeySet *contract, ElektraIoInterface *ioBinding)

  *Creates a contract for use with kdbOpen() that sets up an I/O binding.*
- ElektraIoInterface * elektraIoGetBinding (KDB *kdb)

  *Get I/O binding for asynchronous I/O operations for KDB instance.*

## 573.62.1 Detailed Description

Elektra-I/O structures for I/O bindings, plugins and applications.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.62.2 Typedef Documentation

### 573.62.2.1 ElektraIoBindingAddFd

```
typedef int ElektraIoBindingAddFd(ElektraIoInterface *binding, ElektraIoFdOperation *fdOp)
```

Add file descriptor to be watched by I/O binding.
An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.2 ElektraIoBindingAddIdle

```
typedef int ElektraIoBindingAddIdle(ElektraIoInterface *binding, ElektraIoIdleOperation *idle↩
Op)
```

Add idle to I/O binding.

Idle callbacks are executed without negative effects on other IO sources or the application (e.g. next event loop iteration) An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.3 ElektraIoBindingAddTimer

```
typedef int ElektraIoBindingAddTimer(ElektraIoInterface *binding, ElektraIoTimerOperation
*timerOp)
```

Add timer to I/O binding.

Timeouts callbacks are executed after the initial interval has elapsed and then repeatedly after the interval has elapsed. An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.4 ElektraIoBindingCleanup

```
typedef int ElektraIoBindingCleanup(ElektraIoInterface *binding)
```

Free memory used by I/O binding.

All added operations have to be removed before calling this function.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.5 ElektraIoBindingRemoveFd

```
typedef int ElektraIoBindingRemoveFd(ElektraIoFdOperation *fdOp)
```

Remove file descriptor from I/O binding.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.6 ElektraIoBindingRemoveIdle

```
typedef int ElektraIoBindingRemoveIdle(ElektraIoIdleOperation *idleOp)
```

Remove idle from I/O binding.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.7 ElektraIoBindingRemoveTimer

```
typedef int ElektraIoBindingRemoveTimer(ElektraIoTimerOperation *timerOp)
```

Remove timer from I/O binding.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.2.8 ElektraIoBindingUpdateFd

```
typedef int ElektraIoBindingUpdateFd(ElektraIoFdOperation *fdOp)
```

Notify I/O binding about changes to file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.62.2.9 ElektraIoBindingUpdateIdle

```
typedef int ElektraIoBindingUpdateIdle(ElektraIoIdleOperation *idleOp)
```

Notifiy I/O binding about changes to idle structure.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.62.2.10 ElektraIoBindingUpdateTimer

```
typedef int ElektraIoBindingUpdateTimer(ElektraIoTimerOperation *timerOp)
```

Notifiy I/O binding about changes to timer structure.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.62.2.11 ElektraIoFdCallback

```
typedef void(* ElektraIoFdCallback) (ElektraIoFdOperation *fdOp, int flags)
```

Callback for file descriptor watch operations.
Called whenever a file descriptor status changes.

**Parameters**

| | |
|---|---|
| *fdOp* | operation handle |
| *flags* | current file descriptor flag bitmask |

#### 573.62.2.12 ElektraIoIdleCallback

```
typedef void(* ElektraIoIdleCallback) (ElektraIoIdleOperation *idleOp)
```

Callback for idle operations.
Called whenever compute intensive tasks can be execute without interfering with other operations.

**Parameters**

| | |
|---|---|
| *idleOp* | operation handle |

#### 573.62.2.13 ElektraIoTimerCallback

```
typedef void(* ElektraIoTimerCallback) (ElektraIoTimerOperation *timerOp)
```

Callback for timer operations.
Called after the timer interval has elapsed.

**Parameters**

| | |
|---|---|
| *timerOp* | operation handle |

### 573.62.3 Enumeration Type Documentation

#### 573.62.3.1 ElektraIoFdFlags

```
enum ElektraIoFdFlags
```

Available flags for file descriptors operation bitmask.

**Enumerator**

| | |
|---|---|
| ELEKTRA_IO_READABLE | file descriptor is readable |
| ELEKTRA_IO_WRITABLE | file descriptor is writable |

### 573.62.4 Function Documentation

#### 573.62.4.1 elektraIoBindingAddFd()

```
int elektraIoBindingAddFd (
            ElektraIoInterface * binding,
            ElektraIoFdOperation * fdOp )
```

Add file descriptor to be watched by I/O binding.
An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.2 elektraIoBindingAddIdle()

```
int elektraIoBindingAddIdle (
            ElektraIoInterface * binding,
            ElektraIoIdleOperation * idleOp )
```

Add idle to I/O binding.
Idle callbacks are executed without negative effects on other IO sources or the application (e.g. next event loop iteration) An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.3 elektraIoBindingAddTimer()

```
int elektraIoBindingAddTimer (
            ElektraIoInterface * binding,
            ElektraIoTimerOperation * timerOp )
```

Add timer to I/O binding.
Timeouts callbacks are executed after the initial interval has elapsed and then repeatedly after the interval has elapsed. An operation may only be added to one binding.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.4 elektraIoBindingCleanup()

```
int elektraIoBindingCleanup (
            ElektraIoInterface * binding )
```

Free memory used by I/O binding.
All added operations have to be removed before calling this function.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.5 elektraIoBindingGetData()

```
void* elektraIoBindingGetData (
            ElektraIoInterface * binding )
```

Get private data from I/O Binding.
To be used by I/O binding implementations only.

**Parameters**

| | |
|---|---|
| *binding* | I/O-Binding handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.6 elektraIoBindingRemoveFd()

```
int elektraIoBindingRemoveFd (
            ElektraIoFdOperation * fdOp )
```

Remove file descriptor from I/O binding.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.7 elektraIoBindingRemoveIdle()

```
int elektraIoBindingRemoveIdle (
            ElektraIoIdleOperation * idleOp )
```

Remove idle from I/O binding.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.8 elektraIoBindingRemoveTimer()

```
int elektraIoBindingRemoveTimer (
            ElektraIoTimerOperation * timerOp )
```

Remove timer from I/O binding.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.9 elektraIoBindingSetData()

```
int elektraIoBindingSetData (
            ElektraIoInterface * binding,
            void * data )
```

Set private data from I/O Binding.
To be used by I/O binding implementations only.

**Parameters**

| | |
|---|---|
| *binding* | I/O binding handle |
| *data* | private data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.10 elektraIoBindingUpdateFd()

```
int elektraIoBindingUpdateFd (
            ElektraIoFdOperation * fdOp )
```

Notify I/O binding about changes to file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.11 elektraIoBindingUpdateIdle()

```
int elektraIoBindingUpdateIdle (
            ElektraIoIdleOperation * idleOp )
```

Notifiy I/O binding about changes to idle structure.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.12 elektraIoBindingUpdateTimer()

```
int elektraIoBindingUpdateTimer (
            ElektraIoTimerOperation * timerOp )
```

Notifiy I/O binding about changes to timer structure.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.13 elektraIoContract()

```
int elektraIoContract (
            KeySet * contract,
            ElektraIoInterface * ioBinding )
```

Creates a contract for use with kdbOpen() that sets up an I/O binding.
When you call kdbOpen() with this contract, the KDB instance will use `ioBinding` as its I/O binding.

**Parameters**

| | |
|---|---|
| *contract* | The keyset into which the contract is written. |
| *ioBinding* | The ioBinding to use. |

**Return values**

| | |
|---|---|
| *-1* | if `contract` or `ioBinding` are NULL |
| *0* | on success |

#### 573.62.4.14 elektraIoFdGetBinding()

```
ElektraIoInterface* elektraIoFdGetBinding (
            ElektraIoFdOperation * fdOp )
```

Get binding from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | fd operation handle |

**Returns**

pointer to binding or NULL on error

#### 573.62.4.15 elektraIoFdGetBindingData()

```
void* elektraIoFdGetBindingData (
            ElektraIoFdOperation * fdOp )
```

Get private binding data from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.16 elektraIoFdGetCallback()

```
ElektraIoFdCallback elektraIoFdGetCallback (
            ElektraIoFdOperation * fdOp )
```

Get callback of file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

callback

#### 573.62.4.17 elektraIoFdGetData()

```
void* elektraIoFdGetData (
            ElektraIoFdOperation * fdOp )
```

Get private data from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.18 elektraIoFdGetFd()

```
int elektraIoFdGetFd (
            ElektraIoFdOperation * fdOp )
```

Get file descriptor number from operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

file descriptor number or 0 on error

#### 573.62.4.19 elektraIoFdGetFlags()

```
int elektraIoFdGetFlags (
            ElektraIoFdOperation * fdOp )
```

Get flag bitmask of file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Returns**

watch flag bitmask (see ElektraIoFdFlags).

#### 573.62.4.20 elektraIoFdIsEnabled()

```
int elektraIoFdIsEnabled (
            ElektraIoFdOperation * fdOp )
```

Check if file descriptor watch operation is enabled or disabled.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |

**Return values**

| | |
|---|---|
| *0* | if disabled |
| *1* | if enabled |

#### 573.62.4.21 elektraIoFdSetBindingData()

```
int elektraIoFdSetBindingData (
            ElektraIoFdOperation * fdOp,
            void * data )
```

Set private binding data for operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |
| *data* | pointer to data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.22 elektraIoFdSetEnabled()

```
int elektraIoFdSetEnabled (
            ElektraIoFdOperation * fdOp,
            int enabled )
```

Enable or disable file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |
| *enabled* | 0 to disabled, any other value for enabled |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.23 elektraIoFdSetFlags()

```
int elektraIoFdSetFlags (
            ElektraIoFdOperation * fdOp,
            int flags )
```

Update flag bitmask of file descriptor watch operation.

**Parameters**

| | |
|---|---|
| *fdOp* | file descriptor operation handle |
| *flags* | watch flag bitmask (see ElektraIoFdFlags). |

**Return values**

| | |
|---|---|
| *1* | on success |

**Return values**

| | |
|---|---|
| *0* | on error |

#### 573.62.4.24 elektraIoGetBinding()

```
ElektraIoInterface* elektraIoGetBinding (
            KDB * kdb )
```

Get I/O binding for asynchronous I/O operations for KDB instance.
Returns NULL if no I/O binding was set.

**Parameters**

| | |
|---|---|
| *kdb* | KDB instance |

**Returns**

I/O binding or NULL

#### 573.62.4.25 elektraIoIdleGetBinding()

```
ElektraIoInterface* elektraIoIdleGetBinding (
            ElektraIoIdleOperation * idleOp )
```

Get binding from operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

pointer to binding or NULL on error

#### 573.62.4.26 elektraIoIdleGetBindingData()

```
void* elektraIoIdleGetBindingData (
            ElektraIoIdleOperation * idleOp )
```

Get private binding data from operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.27 elektraIoIdleGetCallback()

```
ElektraIoIdleCallback elektraIoIdleGetCallback (
            ElektraIoIdleOperation * idleOp )
```

Get callback of idle operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

callback

#### 573.62.4.28 elektraIoIdleGetData()

```
void* elektraIoIdleGetData (
            ElektraIoIdleOperation * idleOp )
```

Get private data from operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.29 elektraIoIdleIsEnabled()

```
int elektraIoIdleIsEnabled (
            ElektraIoIdleOperation * idleOp )
```

Check if idle operation is enabled or disabled.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |

**Return values**

| | |
|---|---|
| *0* | if disabled |
| *1* | if enabled |

#### 573.62.4.30 elektraIoIdleSetBindingData()

```
int elektraIoIdleSetBindingData (
            ElektraIoIdleOperation * idleOp,
            void * data )
```

Set private binding data for operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |
| *data* | pointer to data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.31 elektraIoIdleSetEnabled()

```
int elektraIoIdleSetEnabled (
            ElektraIoIdleOperation * idleOp,
            int enabled )
```

Enable or disable idle operation.

**Parameters**

| | |
|---|---|
| *idleOp* | idle operation handle |
| *enabled* | 0 to disabled, any other value for enabled |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.62.4.32 elektraIoNewBinding()

```
ElektraIoInterface* elektraIoNewBinding (
            ElektraIoBindingAddFd * addFd,
            ElektraIoBindingUpdateFd * updateFd,
            ElektraIoBindingRemoveFd * removeFd,
            ElektraIoBindingAddTimer * addTimer,
            ElektraIoBindingUpdateTimer * updateTimer,
            ElektraIoBindingRemoveTimer * removeTimer,
            ElektraIoBindingAddIdle * addIdle,
            ElektraIoBindingUpdateIdle * updateIdle,
            ElektraIoBindingRemoveIdle * removeIdle,
            ElektraIoBindingCleanup * cleanup )
```

Create a new I/O binding.
Make sure to free returned data in ElektraIoBindingCleanup.

**Parameters**

| | |
|---|---|
| *addFd* | function for adding a file descriptor watch operation |
| *updateFd* | function for updating a file descriptor operation |
| *removeFd* | function for removing a file descriptor operation |
| *addTimer* | function for adding a timer operation |
| *updateTimer* | function for updateing a timer operation |
| *removeTimer* | function for removing a timer operation |
| *addIdle* | function for adding an idle operation |
| *updateIdle* | function for updating an idle operation |
| *removeIdle* | function for removing an idle operation |
| *cleanup* | function for cleaning up binding data |

**Returns**

newly created binding

#### 573.62.4.33 elektraIoNewFdOperation()

```
ElektraIoFdOperation* elektraIoNewFdOperation (
            int fd,
            int flags,
            int enabled,
            ElektraIoFdCallback callback,
            void * data )
```

Create a new file descriptor watch operation.
Free returned data after use.

**Parameters**

| | |
|---|---|
| *fd* | file descriptor number |
| *flags* | watch flag bitmask (see ElektraIoFdFlags). Select on which file descriptor state changes the callback should be invoked |
| *enabled* | 0 to disabled, any other value for enabled |
| *callback* | Called when file descriptor state has changes |
| *data* | Custom private data |

**Returns**

file descriptor operation handle

#### 573.62.4.34 elektraIoNewIdleOperation()

```
ElektraIoIdleOperation* elektraIoNewIdleOperation (
            int enabled,
            ElektraIoIdleCallback callback,
            void * data )
```

Create a new idle operation.
Free returned data after use.

**Parameters**

| | |
|---|---|
| *enabled* | 0 to disable, any other value for enabled |
| *callback* | Called when file descriptor state has changes |
| *data* | Custom private data |

**Returns**

idle operation handle

#### 573.62.4.35 elektraIoNewTimerOperation()

```
ElektraIoTimerOperation* elektraIoNewTimerOperation (
            unsigned int interval,
            int enabled,
```

```
            ElektraIoTimerCallback callback,
            void * data )
```

Create a new timer operation.
Free returned data after use.

**Parameters**

| | |
|---|---|
| *interval* | timer interval in milliseconds |
| *enabled* | 0 to disable, any other value for enabled |
| *callback* | Called when file descriptor state has changes |
| *data* | Custom private data |

**Returns**

timer operation handle

#### 573.62.4.36 elektraIoTimerGetBinding()

```
ElektraIoInterface* elektraIoTimerGetBinding (
            ElektraIoTimerOperation * timerOp )
```

Get binding from operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

pointer to binding or NULL on error

#### 573.62.4.37 elektraIoTimerGetBindingData()

```
void* elektraIoTimerGetBindingData (
            ElektraIoTimerOperation * timerOp )
```

Get private binding data from operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.38 elektraIoTimerGetCallback()

```
ElektraIoTimerCallback elektraIoTimerGetCallback (
            ElektraIoTimerOperation * timerOp )
```

Get callback of timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

callback

#### 573.62.4.39 elektraIoTimerGetData()

```
void* elektraIoTimerGetData (
            ElektraIoTimerOperation * timerOp )
```

Get private data from operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

pointer to data or NULL on error

#### 573.62.4.40 elektraIoTimerGetInterval()

```
unsigned int elektraIoTimerGetInterval (
            ElektraIoTimerOperation * timerOp )
```

Get interval of timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Returns**

timer interval in milliseconds, 0 on error

#### 573.62.4.41 elektraIoTimerIsEnabled()

```
int elektraIoTimerIsEnabled (
            ElektraIoTimerOperation * timerOp )
```

Check if timer operation is enabled or disabled.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |

**Return values**

| | |
|---|---|
| *0* | if disabled |
| *1* | if enabled |

#### 573.62.4.42 elektraIoTimerSetBindingData()

```
int elektraIoTimerSetBindingData (
```

```
ElektraIoTimerOperation * timerOp,
void * data )
```

Set private binding data for operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |
| *data* | pointer to data |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.62.4.43 elektraIoTimerSetEnabled()

```
int elektraIoTimerSetEnabled (
ElektraIoTimerOperation * timerOp,
int enabled )
```

Enable or disable timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |
| *enabled* | 0 to disabled, any other value for enabled |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.62.4.44 elektraIoTimerSetInterval()

```
int elektraIoTimerSetInterval (
ElektraIoTimerOperation * timerOp,
unsigned int interval )
```

Update interval of timer operation.

**Parameters**

| | |
|---|---|
| *timerOp* | timer operation handle |
| *interval* | timer interval in milliseconds |

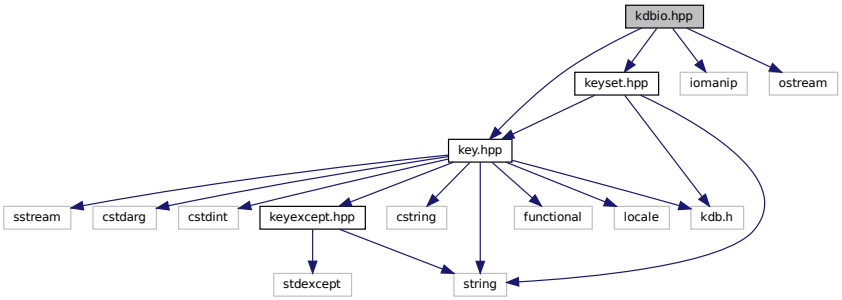
**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

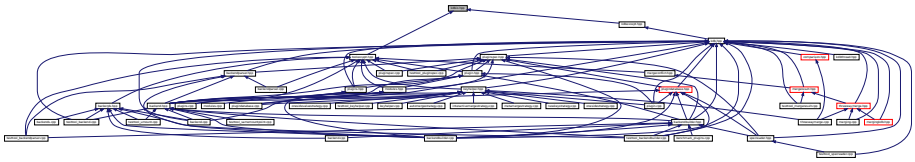
# 573.63 kdbio.hpp File Reference

```
#include <key.hpp>
#include <keyset.hpp>
#include <iomanip>
#include <ostream>
```

Include dependency graph for kdbio.hpp:




This graph shows which files directly or indirectly include this file:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

## 573.63.1 Detailed Description

**Copyright**

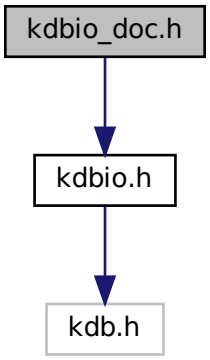
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.64 kdbio_doc.h File Reference

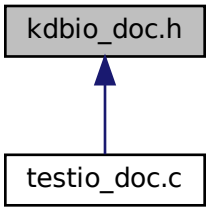
Declarations for the doc I/O binding.

```
#include <kdbio.h>
```

Include dependency graph for kdbio_doc.h:




This graph shows which files directly or indirectly include this file:




### Functions

- ElektraIoInterface * elektraIoDocNew (char *foo)

  *Create and initialize a new doc I/O binding.*

### 573.64.1 Detailed Description

Declarations for the doc I/O binding.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.64.2 Function Documentation

#### 573.64.2.1 elektraIoDocNew()

```
ElektraIoInterface* elektraIoDocNew (
            char * foo )
```

Create and initialize a new doc I/O binding.

**Parameters**

| | |
|---|---|
| *foo* | Some data from I/O management library (e.g. a handle) |

**Returns**

Populated I/O interface

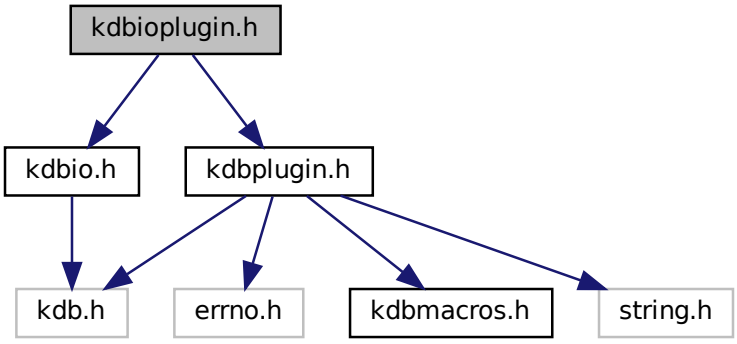
[kdbio binding create]
[kdbio binding create] [kdbio binding setdata]
[kdbio binding setdata]

## 573.65 kdbioplugin.h File Reference

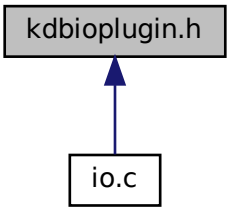
Elektra-I/O functions and declarations for the I/O binding test suite.

```
#include "kdbio.h"
#include "kdbplugin.h"
```

Include dependency graph for kdbioplugin.h:



This graph shows which files directly or indirectly include this file:




## Typedefs

- typedef void(∗ ElektraIoPluginSetBinding) (Plugin ∗plugin, KeySet ∗parameters)

  *Set I/O binding for asynchronous I/O operations for plugin.*

### 573.65.1 Detailed Description

Elektra-I/O functions and declarations for the I/O binding test suite.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.65.2 Typedef Documentation

#### 573.65.2.1 ElektraIoPluginSetBinding

```
typedef void(* ElektraIoPluginSetBinding) (Plugin *plugin, KeySet *parameters)
```

Set I/O binding for asynchronous I/O operations for plugin.
Implemented by plugins to set the I/O binding.

**Parameters**

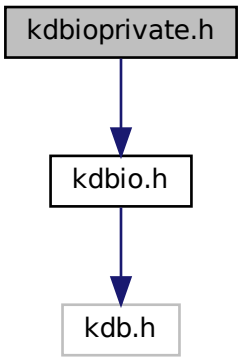
| | |
|---|---|
| *plugin* | plugin handle |
| *parameters* | contains the binary key "/ioBinding" with a pointer to ElektraIoInterface |

## 573.66 kdbioprivate.h File Reference

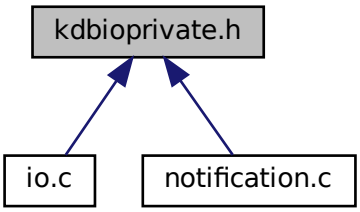
Private Elektra-IO structures for I/O bindings, plugins and applications.

```
#include "kdbio.h"
```

Include dependency graph for kdbioprivate.h:




This graph shows which files directly or indirectly include this file:




### 573.66.1 Detailed Description

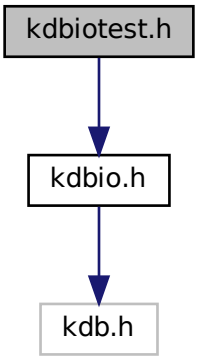
Private Elektra-IO structures for I/O bindings, plugins and applications.

## 573.67 kdbiotest.h File Reference

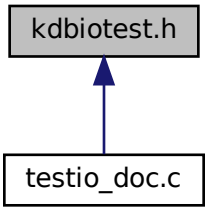
Elektra-I/O functions and declarations for the I/O binding test suite.

```
#include "kdbio.h"
```

Include dependency graph for kdbiotest.h:




This graph shows which files directly or indirectly include this file:




### Typedefs

- typedef ElektraIoInterface *(* ElektraIoTestSuiteCreateBinding) (void)

  *Create and initialize I/O binding.*
- typedef void(* ElektraIoTestSuiteStart) (void)

  *Start I/O processing (for example event loop).*
- typedef void(* ElektraIoTestSuiteStop) (void)

  *Stop I/O processing (for example event loop).*

## Functions

- void elektraIoTestSuite (ElektraIoTestSuiteCreateBinding createBinding, ElektraIoTestSuiteStart start, ElektraIoTestSuiteStop stop)

  *Test-Suite for I/O Bindings.*

### 573.67.1 Detailed Description

Elektra-I/O functions and declarations for the I/O binding test suite.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.67.2 Typedef Documentation

#### 573.67.2.1 ElektraIoTestSuiteCreateBinding

```
typedef ElektraIoInterface*(* ElektraIoTestSuiteCreateBinding) (void)
```

Create and initialize I/O binding.
Used by elektraIoTestSuite between tests to get a fresh binding instance.

**Returns**

initialized I/O binding

#### 573.67.2.2 ElektraIoTestSuiteStart

```
typedef void(* ElektraIoTestSuiteStart) (void)
```

Start I/O processing (for example event loop).
Used by elektraIoTestSuite.
Should not return until processing is stopped (e.g. by calling ElektraIoTestSuiteStop)

#### 573.67.2.3 ElektraIoTestSuiteStop

```
typedef void(* ElektraIoTestSuiteStop) (void)
```

Stop I/O processing (for example event loop).
Used by elektraIoTestSuite

### 573.67.3 Function Documentation

#### 573.67.3.1 elektraIoTestSuite()

```
void elektraIoTestSuite (
            ElektraIoTestSuiteCreateBinding createBinding,
            ElektraIoTestSuiteStart start,
            ElektraIoTestSuiteStop stop )
```

Test-Suite for I/O Bindings.

**Parameters**

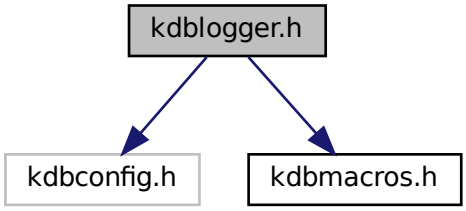
| | |
|---|---|
| *createBinding* | Create and initialize a new I/O binding instance |
| *start* | Pointer to the start function |
| *stop* | Pointer to the stop function |

# 573.68 kdblogger.h File Reference

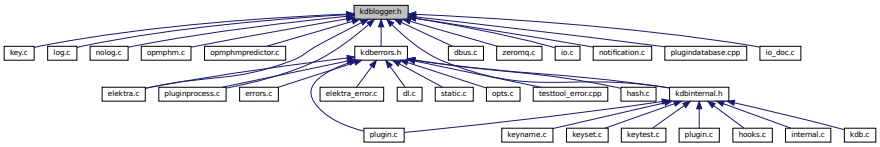
Logger Interface.

```
#include "kdbconfig.h"
#include "kdbmacros.h"
```

Include dependency graph for kdblogger.h:




This graph shows which files directly or indirectly include this file:




## Enumerations

- enum ElektraLogLevel {
  ELEKTRA_LOG_LEVEL_ERROR = 16 , ELEKTRA_LOG_LEVEL_WARNING = 8 , ELEKTRA_LOG_LEVEL_NOTICE = 4 , ELEKTRA_LOG_LEVEL_INFO = 2 ,
  ELEKTRA_LOG_LEVEL_DEBUG = 1 }

## 573.68.1 Detailed Description

Logger Interface.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.68.2 Enumeration Type Documentation

### 573.68.2.1 ElektraLogLevel

```
enum ElektraLogLevel
```

**Enumerator**

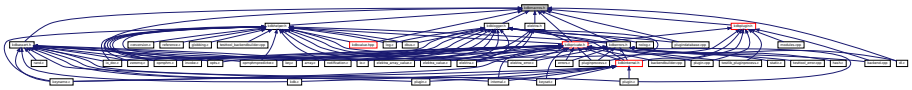
| | |
|---|---|
| ELEKTRA_LOG_LEVEL_ERROR | assertion failed, will abort Should only be used by ELEKTRA_ASSERT, not by other code! |

**Enumerator**

| | |
|---|---|
| ELEKTRA_LOG_LEVEL_WARNING | something happened an end user would be interested in The action requested by the user failed in an unusual way, e.g. no memory. This level is especially for API misuse, something we want to report the user but otherwise could not (e.g. because we are not in kdb∗ context and cannot yield warnings/errors). |
| ELEKTRA_LOG_LEVEL_NOTICE | something important happened which usually should not Despite its name this level can be used for errors, failed activities and so on, if other plugins or similar might recover the error, the action will be retried, and/or the user will not notice or notice by warnings/errors/return value anyway. |
| ELEKTRA_LOG_LEVEL_INFO | The standard log level. Use it to report mentionable occurrences which do not flood the system too badly. They should be indicative for what the problem is in the most cases. |
| ELEKTRA_LOG_LEVEL_DEBUG | The debug log level. Can be used to log anything which was useful to debug the code. Is usually only turned on by the developer itself and you do not need consider the number of output it produces (expected to be used together with line-based suppressions in log.c). |

## 573.69 kdbmacros.h File Reference

Macros by Elektra.
This graph shows which files directly or indirectly include this file:



### Macros

- #define ELEKTRA_QUOTE(x) #x

  *Surround a value with double quotes.*
- #define ELEKTRA_STRINGIFY(x) ELEKTRA_QUOTE (x)

  *Surround a **macro value** with double quotes.*
- #define ELEKTRA_CONCAT2(X, Y) X##Y

  *Concat two values.*
- #define ELEKTRA_CONCAT(X, Y) ELEKTRA_CONCAT2 (X, Y)

  *Concat two **macro values***
- #define ELEKTRA_SET_ERROR_READ_ONLY(info, returned, error)

  *Sets error if info != returned.*
- #define ELEKTRA_SYMVER(sym, impl) sym##_##impl

  *Helper macro to create a versioned name of a symbol.*
- #define ELEKTRA_SYMVER_DECLARE(ver, sym, impl) ELEKTRA_SYMVER_COMMAND (ELEKTRA_STRINGIFY (ELEKTRA_SYMVER (sym, impl)), #sym "@" ver)

  *Declares another version of a symbol using the `.symver` assembler pseudo command.*

### 573.69.1 Detailed Description

Macros by Elektra.
Macros start with ELEKTRA_ and are uppercase.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.69.2 Macro Definition Documentation

### 573.69.2.1 ELEKTRA_SET_ERROR_READ_ONLY

```
#define ELEKTRA_SET_ERROR_READ_ONLY(
              info,
              returned,
              error )
```

Sets error if info != returned.

**Parameters**

| | |
|---|---|
| *info* | how the info is now (freshly received) |
| *returned* | how the info passed from user is |
| *error* | key to set error to |

**Returns**

with -1 on error

### 573.69.2.2 ELEKTRA_SYMVER

```
#define ELEKTRA_SYMVER(
              sym,
              impl ) sym##_##impl
```

Helper macro to create a versioned name of a symbol.

**Parameters**

| | |
|---|---|
| *sym* | unversioned name of the symbol |
| *impl* | version suffix |

### 573.69.2.3 ELEKTRA_SYMVER_DECLARE

```
#define ELEKTRA_SYMVER_DECLARE(
              ver,
              sym,
              impl ) ELEKTRA_SYMVER_COMMAND (ELEKTRA_STRINGIFY (ELEKTRA_SYMVER (sym, impl)),
#sym "@" ver)
```

Declares another version of a symbol using the `.symver` assembler pseudo command.

**Parameters**

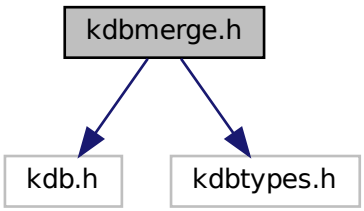
| | |
|---|---|
| *ver* | the version name as declared versions.def |
| *sym* | the unversioned name of the symbol |
| *impl* | the version suffix to use for this version |

# 573.70 kdbmerge.h File Reference

Kdb merge tool.

```
#include "kdb.h"
#include "kdbtypes.h"
```

Include dependency graph for kdbmerge.h:




## Enumerations

- enum MergeStrategy { MERGE_STRATEGY_ABORT = 1 , MERGE_STRATEGY_OUR = 3 , MERGE_STRATEGY_THEIR = 4 }

  *The strategy to use when a merge conflict occurs.*

## Functions

- KeySet ∗ elektraMerge (KeySet ∗our, Key ∗ourRoot, KeySet ∗their, Key ∗theirRoot, KeySet ∗base, Key ∗baseRoot, Key ∗resultKey, enum MergeStrategy strategy, Key ∗informationKey)

  *This function can incorporate changes from two modified versions (our and their) into a common preceding version (base) of a key set.*
- int elektraMergeGetConflicts (Key ∗informationKey)

  *This function returns the number of conflicts that is store in the key.*
- KeySet ∗ elektraMergeGetConflictingKeys (Key ∗informationKey, Key ∗root)

  *Returns a keyset with all conflicting keys under the specified root.*
- bool elektraMergeIsKeyConflicting (Key ∗informationKey, Key ∗root, Key ∗key)

  *Check whether the given key was part of a conflict.*

## 573.70.1 Detailed Description

Kdb merge tool.

**Copyright**

BSD License (see LICENSE.md or https://ww.libelektra.org)

## 573.70.2 Enumeration Type Documentation

### 573.70.2.1 MergeStrategy

```
enum MergeStrategy
```

The strategy to use when a merge conflict occurs.

**Enumerator**

| | |
|---|---|
| MERGE_STRATEGY_ABORT | Abort merging if a conflict occurs |
| MERGE_STRATEGY_OUR | Prefer our keys in case of conflict |
| MERGE_STRATEGY_THEIR | Prefer their keys in case of conflict |

## 573.70.3 Function Documentation

### 573.70.3.1 elektraMerge()

```
KeySet* elektraMerge (
              KeySet * our,
              Key * ourRoot,
              KeySet * their,
              Key * theirRoot,
              KeySet * base,
              Key * baseRoot,
              Key * resultRoot,
              enum MergeStrategy strategy,
              Key * informationKey )
```

This function can incorporate changes from two modified versions (our and their) into a common preceding version (base) of a key set.

This lets you merge the sets of changes represented by the two newer key sets. This is called a three-way merge between key sets.

```
#include <kdb.h>
#include <kdbmerge.h>
#include <stdio.h>
static void print_results (KeySet * result, Key * resultRoot, Key * informationKey)
{
        KeySet * conflictingKeys = elektraMergeGetConflictingKeys (informationKey, resultRoot);
        for (elektraCursor i = 0; i < ksGetSize (conflictingKeys); i++)
        {
                printf ("Conflict in key %s\n", keyName (ksAtCursor (conflictingKeys, i)));
        }
        printf ("Result:\n");
        for (elektraCursor i = 0; i < ksGetSize (result); i++)
        {
                Key * k = ksAtCursor (result, i);
                printf ("%s = %s\n", keyName (k), keyString (k));
        }
        if (result == NULL)
        {
                printf ("Aborted.\n");
        }
        ksDel (conflictingKeys);
}
int main (void)
{
        Key * baseRoot = keyNew ("user:/screen", KEY_END);
        KeySet * base = ksNew (1, keyNew ("user:/screen/background", KEY_VALUE, "blue", KEY_END), KS_END);
        Key * theirRoot = keyDup (baseRoot, KEY_CP_ALL);
        KeySet * their = ksNew (2, keyNew ("user:/screen/background", KEY_VALUE, "red", KEY_END),
                                keyNew ("user:/screen/brightness", KEY_VALUE, "100", KEY_END), KS_END);
        Key * ourRoot = keyDup (baseRoot, KEY_CP_ALL);
        KeySet * our = ksNew (2, keyNew ("user:/screen/background", KEY_VALUE, "purple", KEY_END),
                              keyNew ("user:/screen/standby", KEY_VALUE, "on", KEY_END), KS_END);
        Key * resultRoot = keyDup (baseRoot, KEY_CP_ALL);
        Key * informationKey = keyDup (baseRoot, KEY_CP_ALL);
        // Strategy: Abort
        // Will abort due to user:/screen/background changes in both their and our
        printf ("Trying merge strategy ABORT\n");
        KeySet * result = elektraMerge (our, ourRoot, their, theirRoot, base, baseRoot, resultRoot,
       MERGE_STRATEGY_ABORT, informationKey);
        print_results (result, resultRoot, informationKey);
        ksDel (result);
        // Strategy: Our
        // Will take value of our for user:/screen/background
        printf ("\nTrying merge strategy OUR\n");
        result = elektraMerge (our, ourRoot, their, theirRoot, base, baseRoot, resultRoot,
       MERGE_STRATEGY_OUR, informationKey);
```

```
        print_results (result, resultRoot, informationKey);
        ksDel (result);
        // Strategy: Their
        // Will take value of their for user:/screen/background
        printf ("\nTrying merge strategy THEIR\n");
        result = elektraMerge (our, ourRoot, their, theirRoot, base, baseRoot, resultRoot,
       MERGE_STRATEGY_THEIR, informationKey);
        print_results (result, resultRoot, informationKey);
        ksDel (result);
        ksDel (base);
        ksDel (their);
        ksDel (our);
        keyDel (baseRoot);
        keyDel (theirRoot);
        keyDel (ourRoot);
        keyDel (informationKey);
        keyDel (resultRoot);
        return 0;
}
```

Join three key sets together

**Parameters**

| | |
|---|---|
| *our* | our key set |
| *ourRoot* | key that has the root of our as name |
| *their* | their key set |
| *theirRoot* | key that has the root of their as name |
| *base* | base key set |
| *baseRoot* | key that has the root of base as name |
| *resultRoot* | the name of this key determines where the resulting key set will be stored |
| *strategy* | specify which merge strategy to choose in case of a conflict |
| *informationKey* | stores errors as well as statistics |

**Returns**

the merged key set and NULL on error

### 573.70.3.2 elektraMergeGetConflictingKeys()

```
KeySet* elektraMergeGetConflictingKeys (
             Key * informationKey,
             Key * root )
```

Returns a keyset with all conflicting keys under the specified root.

```
#include <kdb.h>
#include <kdbmerge.h>
#include <stdio.h>
static void print_results (KeySet * result, Key * resultRoot, Key * informationKey)
{
        KeySet * conflictingKeys = elektraMergeGetConflictingKeys (informationKey, resultRoot);
        for (elektraCursor i = 0; i < ksGetSize (conflictingKeys); i++)
        {
                printf ("Conflict in key %s\n", keyName (ksAtCursor (conflictingKeys, i)));
        }
        printf ("Result:\n");
        for (elektraCursor i = 0; i < ksGetSize (result); i++)
        {
                Key * k = ksAtCursor (result, i);
                printf ("%s = %s\n", keyName (k), keyString (k));
        }
        if (result == NULL)
        {
                printf ("Aborted.\n");
        }
        ksDel (conflictingKeys);
}
int main (void)
{
        Key * baseRoot = keyNew ("user:/screen", KEY_END);
        KeySet * base = ksNew (1, keyNew ("user:/screen/background", KEY_VALUE, "blue", KEY_END), KS_END);
        Key * theirRoot = keyDup (baseRoot, KEY_CP_ALL);
```

```
        KeySet * their = ksNew (2, keyNew ("user:/screen/background", KEY_VALUE, "red", KEY_END),
                                keyNew ("user:/screen/brightness", KEY_VALUE, "100", KEY_END), KS_END);
        Key * ourRoot = keyDup (baseRoot, KEY_CP_ALL);
        KeySet * our = ksNew (2, keyNew ("user:/screen/background", KEY_VALUE, "purple", KEY_END),
                              keyNew ("user:/screen/standby", KEY_VALUE, "on", KEY_END), KS_END);
        Key * resultRoot = keyDup (baseRoot, KEY_CP_ALL);
        Key * informationKey = keyDup (baseRoot, KEY_CP_ALL);
        // Strategy: Abort
        // Will abort due to user:/screen/background changes in both their and our
        printf ("Trying merge strategy ABORT\n");
        KeySet * result = elektraMerge (our, ourRoot, their, theirRoot, base, baseRoot, resultRoot,
      MERGE_STRATEGY_ABORT, informationKey);
        print_results (result, resultRoot, informationKey);
        ksDel (result);
        // Strategy: Our
        // Will take value of our for user:/screen/background
        printf ("\nTrying merge strategy OUR\n");
        result = elektraMerge (our, ourRoot, their, theirRoot, base, baseRoot, resultRoot,
      MERGE_STRATEGY_OUR, informationKey);
        print_results (result, resultRoot, informationKey);
        ksDel (result);
        // Strategy: Their
        // Will take value of their for user:/screen/background
        printf ("\nTrying merge strategy THEIR\n");
        result = elektraMerge (our, ourRoot, their, theirRoot, base, baseRoot, resultRoot,
      MERGE_STRATEGY_THEIR, informationKey);
        print_results (result, resultRoot, informationKey);
        ksDel (result);
        ksDel (base);
        ksDel (their);
        ksDel (our);
        keyDel (baseRoot);
        keyDel (theirRoot);
        keyDel (ourRoot);
        keyDel (informationKey);
        keyDel (resultRoot);
        return 0;
}
```

**Parameters**

| | |
|---|---|
| *informationKey* | stores errors as well as statistics |
| *root* | The root key (either for base, theirs, ours or result) that was used in the merge |

**Returns**

KeySet containing all keys that are part of a merge conflict. Will be empty if the specified root key was not part of the merge.

### 573.70.3.3 elektraMergeGetConflicts()

```
int elektraMergeGetConflicts (
            Key * informationKey )
```

This function returns the number of conflicts that is store in the key.

**Parameters**

| | |
|---|---|
| *informationKey* | contains the statistics in its meta information |

**Returns**

the number of conflicts stored in the key

### 573.70.3.4 elektraMergeIsKeyConflicting()

```
bool elektraMergeIsKeyConflicting (
            Key * informationKey,
```

```
Key * root,
Key * key )
```

Check whether the given key was part of a conflict.
NOTE: Even if the conflict was resolved via the provided conflict strategy, the key will still be marked as being part of a conflict.

**Parameters**

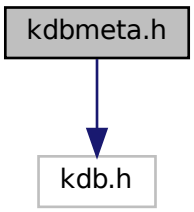
| | |
|---|---|
| *informationKey* | stores errors as well as statistics |
| *root* | The root key (either for base, theirs, ours or result) that was used in the merge |
| *key* | That key that should be checked. Must be located under the specified root key |

**Return values**

| | |
|---|---|
| *1* | if the key was part of a conflict |
| *0* | if the key was not part of a conflict or the given root was not part of the merge |

## 573.71 kdbmeta.h File Reference

metadata functions
`#include "kdb.h"`
Include dependency graph for kdbmeta.h:




This graph shows which files directly or indirectly include this file:

### Functions

- const char * keyComment (const Key *key)

  *Return a pointer to the real internal `key` comment.*
- ssize_t keyGetCommentSize (const Key *key)

  *Calculates number of bytes needed to store a key comment, including final NULL.*
- ssize_t keyGetComment (const Key *key, char *returnedDesc, size_t maxSize)

  *Get the key comment.*

- ssize_t keySetComment (Key *key, const char *newDesc)

  *Set a comment for a key.*
- int elektraKeyCmpOrder (const Key *a, const Key *b)

  *Compare the order metadata of two keys.*
- KeySet * elektraMetaArrayToKS (Key *, const char *)

  *Create a `KeySet` from a metakey array.*
- void elektraMetaArrayAdd (Key *, const char *, const char *)

  *creates an metadata array or appends another element to an existing metadata array e.g.*
- char * elektraMetaArrayToString (const Key *, const char *, const char *)

  *returns the metakey array as a string separated by delim*
- int elektraSortTopology (KeySet *, Key **)

  *topological sorting*

### 573.71.1 Detailed Description

metadata functions
These functions might be removed in a later version.

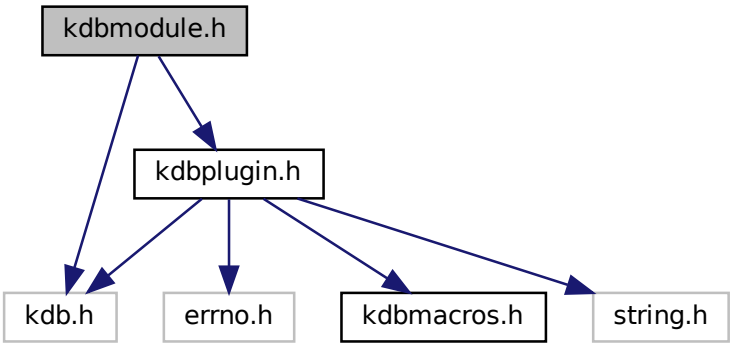
**Copyright**

BSD License (see LICENSE.md or `https://www.libelektra.org`)

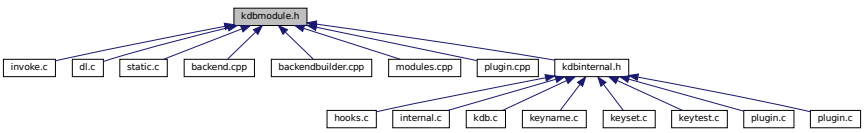
## 573.72 kdbmodule.h File Reference

```
#include <kdb.h>
#include <kdbplugin.h>
```

Include dependency graph for kdbmodule.h:




This graph shows which files directly or indirectly include this file:



### Functions

- int elektraModulesInit (KeySet ∗modules, Key ∗error)

  *Initialises the module loading system.*
- elektraPluginFactory elektraModulesLoad (KeySet ∗modules, const char ∗name, Key ∗error)

  *Load a library with the given name.*
- int elektraModulesClose (KeySet ∗modules, Key ∗error)

  *Close all modules.*

#### 573.72.1 Detailed Description

**Copyright**

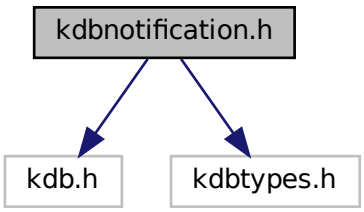
BSD License (see LICENSE.md or `https://www.libelektra.org`)

## 573.73 kdbnotification.h File Reference

Elektra-Notification structures and declarations for application developers.

```
#include "kdb.h"
#include "kdbtypes.h"
```

Include dependency graph for kdbnotification.h:




This graph shows which files directly or indirectly include this file:

### Typedefs

- typedef void(∗ ElektraNotificationConversionErrorCallback) (Key ∗key, void ∗context)

  *Callback function called when string to number conversion failed.*
- typedef void(∗ ElektraNotificationChangeCallback) (Key ∗key, void ∗context)

  *Callback function for key changes.*

### Functions

- int elektraNotificationContract (KeySet ∗contract)

  *Creates a contract for use with kdbOpen() that sets up notifications.*
- int elektraNotificationRegisterCallback (KDB ∗kdb, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

*Subscribe for updates via callback when a given key value is changed.*

- int elektraNotificationRegisterCallbackSameOrBelow (KDB ∗kdb, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

  *Subscribe for updates via callback when a given key or a key below changed.*

- int elektraNotificationRegisterInt (KDB ∗kdb, Key ∗key, int ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterUnsignedInt (KDB ∗kdb, Key ∗key, unsigned int ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterLong (KDB ∗kdb, Key ∗key, long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterUnsignedLong (KDB ∗kdb, Key ∗key, unsigned long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterLongLong (KDB ∗kdb, Key ∗key, long long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterUnsignedLongLong (KDB ∗kdb, Key ∗key, unsigned long long ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterFloat (KDB ∗kdb, Key ∗key, float ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterDouble (KDB ∗kdb, Key ∗key, double ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbBoolean (KDB ∗kdb, Key ∗key, kdb_boolean_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbChar (KDB ∗kdb, Key ∗key, kdb_char_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbOctet (KDB ∗kdb, Key ∗key, kdb_octet_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbShort (KDB ∗kdb, Key ∗key, kdb_short_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbUnsignedShort (KDB ∗kdb, Key ∗key, kdb_unsigned_short_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbLong (KDB ∗kdb, Key ∗key, kdb_long_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbUnsignedLong (KDB ∗kdb, Key ∗key, kdb_unsigned_long_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbLongLong (KDB ∗kdb, Key ∗key, kdb_long_long_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbUnsignedLongLong (KDB ∗kdb, Key ∗key, kdb_unsigned_long_long_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbFloat (KDB ∗kdb, Key ∗key, kdb_float_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*
- int elektraNotificationRegisterKdbDouble (KDB ∗kdb, Key ∗key, kdb_double_t ∗variable)

  *Subscribe for automatic updates to a given variable when the given key value is changed.*

### 573.73.1 Detailed Description

Elektra-Notification structures and declarations for application developers.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.73.2 Function Documentation

#### 573.73.2.1 elektraNotificationContract()

```
int elektraNotificationContract (
            KeySet * contract )
```

Creates a contract for use with kdbOpen() that sets up notifications.
When you call kdbOpen() with this contract, the `internalnotification` plugin will be mounted automatically.
This allows you to call other `elektraNotification*` functions.
If you need to configure notification transport plugins, you should manually add the relevant keys to `contract`.

**Parameters**

| | |
|---|---|
| *contract* | The keyset into which the contract is written. |

**Return values**

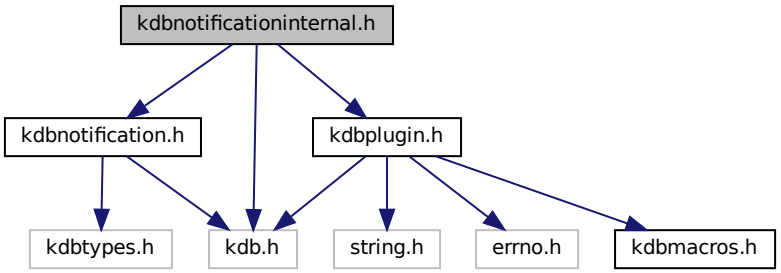
| | |
|---|---|
| *-1* | if `contract` is NULL |
| *0* | on success |

## 573.74 kdbnotificationinternal.h File Reference

Elektra-Notification structures and declarations for developing notification and transport plugins.

```
#include "kdb.h"
#include "kdbnotification.h"
#include "kdbplugin.h"
```

Include dependency graph for kdbnotificationinternal.h:




This graph shows which files directly or indirectly include this file:

### Typedefs

- typedef int(* ElektraNotificationPluginRegisterCallback) (Plugin *handle, Key *key, ElektraNotificationChangeCallback callback, void *context)

*Subscribe for updates via callback when a given key value is changed.*

- typedef int(∗ ElektraNotificationPluginRegisterCallbackSameOrBelow) (Plugin ∗handle, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

  *Subscribe for updates via callback when a given key or a key below is changed.*
- typedef void(∗ ElektraNotificationSetConversionErrorCallback) (Plugin ∗handle, ElektraNotificationConversionErrorCallback callback, void ∗context)

  *Allow setting a callback that is called when a value conversion failed.*
- typedef struct _ElektraNotificationCallbackContext ElektraNotificationCallbackContext

  *Context for notification callbacks.*
- typedef void(∗ ElektraNotificationCallback) (Key ∗key, ElektraNotificationCallbackContext ∗context)

  *Notify notification library of changes to a key.*
- typedef void(∗ ElektraNotificationKdbUpdate) (KDB ∗kdb, Key ∗changedKey)

  *Used by notification plugins to get values from the key database.*

## 573.74.1 Detailed Description

Elektra-Notification structures and declarations for developing notification and transport plugins.
Only available in Elektra's source.
Only used by elektra-notification library, notification plugins (e.g. internalnotification) and transport plugins.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.74.2 Typedef Documentation

### 573.74.2.1 ElektraNotificationCallback

```
typedef void(* ElektraNotificationCallback) (Key *key, ElektraNotificationCallbackContext
*context)
```

Notify notification library of changes to a key.
This callback is provided by the `internalnotification` plugin and for use by notification transport plugins.

**Parameters**

| | |
|---|---|
| *key* | changed key |
| *context* | additional data |

### 573.74.2.2 ElektraNotificationKdbUpdate

```
typedef void(* ElektraNotificationKdbUpdate) (KDB *kdb, Key *changedKey)
```

Used by notification plugins to get values from the key database.

**Parameters**

| | |
|---|---|
| *kdb* | kdb handle |
| *changedKey* | which key was updated |

### 573.74.2.3 ElektraNotificationPluginRegisterCallback

```
typedef int(* ElektraNotificationPluginRegisterCallback) (Plugin *handle, Key *key, ElektraNotificationChangeC
```

callback, void *context)
Subscribe for updates via callback when a given key value is changed.
Exported as "registerCallback" by notification plugins.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *callback* | callback function |
| *context* | user supplied context passed to callback function |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 573.74.2.4 ElektraNotificationPluginRegisterCallbackSameOrBelow

typedef int(* ElektraNotificationPluginRegisterCallbackSameOrBelow) (Plugin *handle, Key *key, ElektraNotificationChangeCallback callback, void *context)
Subscribe for updates via callback when a given key or a key below is changed.
Exported as "registerCallbackSameOrBelow" by notification plugins.

**Parameters**

| | |
|---|---|
| *handle* | plugin handle |
| *key* | key to watch for changes |
| *callback* | callback function |
| *context* | user supplied context passed to callback function |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on failure |

#### 573.74.2.5 ElektraNotificationSetConversionErrorCallback

typedef void(* ElektraNotificationSetConversionErrorCallback) (Plugin *handle, ElektraNotificationConversionEr
callback, void *context)
Allow setting a callback that is called when a value conversion failed.
Exported as "setConversionErrorCallback" notification plugins.

**Parameters**

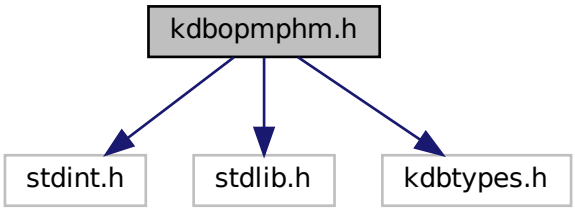
| | |
|---|---|
| *kdb* | kdb handle |
| *callback* | callback |
| *context* | context |

# 573.75 kdbopmphm.h File Reference

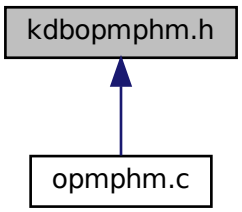
Defines for the Order Preserving Minimal Perfect Hash Map.

```
#include <stdint.h>
#include <stdlib.h>
#include <kdbtypes.h>
```

Include dependency graph for kdbopmphm.h:




This graph shows which files directly or indirectly include this file:




## Classes

- struct OpmphmEdge

  *Order Preserving Minimal Perfect Hash Map.*
- struct Opmphm

  *The opmphm.*

## Macros

- #define OPMPHM_HASHFUNCTION_ROT(x, k) (((x) << (k)) | ((x) >> (32 - (k))))

  *Hash function By Bob Jenkins, May 2006* http://burtleburtle.net/bob/c/lookup3.c.

## Typedefs

- typedef const char *(* opmphmGetName) (void *)

  *Only needed for Initialisation.*

## Enumerations

- enum opmphmflag_t { OPMPHM_FLAG_MMAP_STRUCT = 1 , OPMPHM_FLAG_MMAP_HASHFUNCTIONSEEDS = 1 << 2 , OPMPHM_FLAG_MMAP_GRAPH = 1 << 3 }

  *Opmphm Flags.*

## Functions

- double opmphmMinC (uint8_t r)

  *Functions providing `r` and `c`*
- uint8_t opmphmOptR (size_t n)

  *Provides the optimal `r` value for a given `n`*
- double opmphmOptC (size_t n)

  *Provides the optimal `c` value for a given `n`*
- OpmphmGraph * opmphmGraphNew (Opmphm *opmphm, uint8_t r, size_t n, double c)

  *Graph functions.*
- void opmphmGraphClear (const Opmphm *opmphm, OpmphmGraph *graph)

  *Clears the OpmphmGraph.*
- void opmphmGraphDel (OpmphmGraph *graph)

  *Deletes the OpmphmGraph.*
- int opmphmMapping (Opmphm *opmphm, OpmphmGraph *graph, OpmphmInit *init, size_t n)

  *Build functions.*
- int opmphmAssignment (Opmphm *opmphm, OpmphmGraph *graph, size_t n, int defaultOrder)

  *Assigns the vertices of the r-uniform r-partite hypergraph.*
- size_t opmphmLookup (Opmphm *opmphm, size_t n, const void *name)

  *Lookup functions.*
- Opmphm * opmphmNew (void)

  *Basic functions.*
- void opmphmDel (Opmphm *opmphm)

  *Deletes the OPMPHM.*
- int opmphmIsBuild (const Opmphm *opmphm)

  *OPMPHM is build.*
- int opmphmCopy (Opmphm *dest, const Opmphm *source)

  *Copies OPMPHM from source to destination.*
- void opmphmClear (Opmphm *opmphm)

  *Clears the OPMPHM.*
- uint32_t opmphmHashfunction (const void *key, size_t length, uint32_t initval)

  *Hash function By Bob Jenkins, May 2006 http://burtleburtle.net/bob/c/lookup3.c Original name: hashlitte.*

### 573.75.1 Detailed Description

Defines for the Order Preserving Minimal Perfect Hash Map.

**Copyright**

BSD License (see doc/COPYING or https://www.libelektra.org)

### 573.75.2 Enumeration Type Documentation

#### 573.75.2.1 opmphmflag_t

enum opmphmflag_t

Opmphm Flags.

**Enumerator**

| | |
|---|---|
| OPMPHM_FLAG_MMAP_STRUCT | Opmphm struct lies inside a mmap region. This flag is set for Opmphm structs inside a mapped region. It prevents erroneous free() calls on these Opmphm structs. |
| OPMPHM_FLAG_MMAP_HASHFUNCTIONSEEDS | Opmphm hashFunctionSeeds lies inside a mmap region. This flag is set for Opmphm hashFunctionSeeds inside a mapped region. It prevents erroneous free() calls on these hashFunctionSeeds. |
| OPMPHM_FLAG_MMAP_GRAPH | Opmphm graph lies inside a mmap region. This flag is set for Opmphm graphs inside a mapped region. It prevents erroneous free() calls on these graphs. |

## 573.75.3 Function Documentation

### 573.75.3.1 opmphmAssignment()

```
int opmphmAssignment (
            Opmphm * opmphm,
            OpmphmGraph * graph,
            size_t n,
            int defaultOrder )
```

Assigns the vertices of the r-uniform r-partite hypergraph.

Allocs the memory for the final OPMPHM `Opmphm->graph`. Uses the remove sequence `Opmphm->Graph->removeSequence`, generated during cycle check, to assign each vertex. Either with `Opmphm->Edge->order` or the default order, default is the order of `OpmphmInit->data`.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *graph* | the OpmphmGraph |
| *n* | the number of elements |
| *defaultOrder* | boolean flag |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on memory error |

### 573.75.3.2 opmphmClear()

```
void opmphmClear (
            Opmphm * opmphm )
```

Clears the OPMPHM.

The OPMPHM must be successfully created with `opmphmNew ()`. Clears and frees all internal memory of Opmphm, but not `Opmphm->hashFunctionSeeds` and the Opmphm instance.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

#### 573.75.3.3 opmphmCopy()

```
int opmphmCopy (
            Opmphm * dest,
            const Opmphm * source )
```

Copies OPMPHM from source to destination.
Clears the dest and copies memory and values from source.

**Parameters**

| | |
|---|---|
| *dest* | the OPMPHM destination |
| *source* | the OPMPHM source |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on memory error |

#### 573.75.3.4 opmphmDel()

```
void opmphmDel (
            Opmphm * opmphm )
```

Deletes the OPMPHM.
Clears and frees all memory in Opmphm.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

#### 573.75.3.5 opmphmGraphClear()

```
void opmphmGraphClear (
            const Opmphm * opmphm,
            OpmphmGraph * graph )
```

Clears the OpmphmGraph.
Sets all vertices to 0.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *graph* | the OpmphmGraph |

#### 573.75.3.6 opmphmGraphDel()

```
void opmphmGraphDel (
            OpmphmGraph * graph )
```

Deletes the OpmphmGraph.

**Parameters**

| | |
|---|---|
| *graph* | the OpmphmGraph |

#### 573.75.3.7 opmphmGraphNew()

```
OpmphmGraph* opmphmGraphNew (
            Opmphm * opmphm,
            uint8_t r,
            size_t n,
            double c )
```

Graph functions.
Graph functions.
The OpmphmGraph represents a r-uniform r-partite hypergraph. Lazy initializes the `opmphm->hashFunctionSeeds` with r. Calculates also the size of one partition in the r-uniform r-partite hypergraph and stores it in `opmphm->componentSize`. Allocates all memory for the OpmphmGraph.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *r* | the rUniPar |
| *n* | the number of elements |
| *c* | space influencing parameter |

**Return values**

| | |
|---|---|
| *OpmphmGraph* | * success |
| *NULL* | memory error |

#### 573.75.3.8 opmphmIsBuild()

```
int opmphmIsBuild (
            const Opmphm * opmphm )
```

OPMPHM is build.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

**Return values**

| | |
|---|---|
| *0* | on false |
| *-1* | on true or NULL |

#### 573.75.3.9 opmphmLookup()

```
size_t opmphmLookup (
            Opmphm * opmphm,
            size_t n,
```

```
            const void * name )
```

Lookup functions.
Lookup functions.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *n* | the number of elements |
| *name* | the name of the element |

**Return values**

| | |
|---|---|
| *size_t* | the order of the element. |

#### 573.75.3.10 opmphmMapping()

```
int opmphmMapping (
            Opmphm * opmphm,
            OpmphmGraph * graph,
            OpmphmInit * init,
            size_t n )
```

Build functions.
Build functions.
Sets the seeds for the opmphmHashfunctions, `OpmphmInit->initSeed` will be changed. Inserts each element as edge in the r-uniform r-partite hypergraph and checks if the graph contains a cycle. If there are cycles the `graph` will be cleaned

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *graph* | the OpmphmGraph |
| *init* | the OpmphmInit |
| *n* | the number of elements |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | mapping not possible |

#### 573.75.3.11 opmphmMinC()

```
double opmphmMinC (
            uint8_t r )
```

Functions providing `r` and `c`
Functions providing `r` and `c`
This minimal values come from Fabiano Cupertino Botelho, Near-Optimal Space Perfect Hashing Algorithms, 2008.

**Parameters**

| | |
|---|---|
| *r* | the rUniPar |

**Return values**

| | |
|---|---|
| *c* | the minimal c value |

### 573.75.3.12 opmphmNew()

```
Opmphm* opmphmNew (
            void  )
```

Basic functions.
Basic functions.

**Return values**

| | |
|---|---|
| *Opmphm* | * success |
| *NULL* | memory error |

### 573.75.3.13 opmphmOptC()

```
double opmphmOptC (
            size_t n )
```

Provides the optimal `c` value for a given `n`
This is a heuristic, the return values follow from the mapping benchmark.

**Parameters**

| | |
|---|---|
| *n* | the number of elements to hash |

**Return values**

| | |
|---|---|
| *c* | the optimal `c` value |

### 573.75.3.14 opmphmOptR()

```
uint8_t opmphmOptR (
            size_t n )
```

Provides the optimal `r` value for a given `n`
This is a heuristic, the return values follow from the mapping benchmark.

**Parameters**

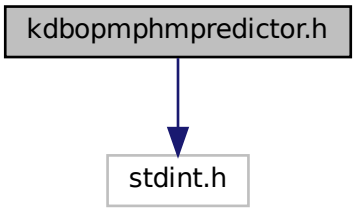
| | |
|---|---|
| *n* | the number of elements to hash |

**Return values**

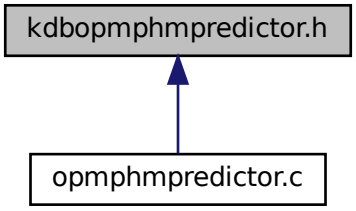
| | |
|---|---|
| *r* | the optimal rUniPar |

## 573.76 kdbopmphmpredictor.h File Reference

Defines for the Order Preserving Minimal Perfect Hash Map Predictor.
`#include <stdint.h>`
Include dependency graph for kdbopmphmpredictor.h:




This graph shows which files directly or indirectly include this file:




### Enumerations

- enum predictorflag_t { OPMPHM_PREDICTOR_FLAG_MMAP_STRUCT = 1 , OPMPHM_PREDICTOR_FLAG_MMAP_PATTE = 1 << 2 }

  *OpmphmPredictor Flags.*

### Functions

- OpmphmPredictor * opmphmPredictorNew (void)

  *Basic functions.*
- void opmphmPredictorDel (OpmphmPredictor *op)

  *Deletes the OpmphmPredictor.*
- void opmphmPredictorCopy (OpmphmPredictor *dest, OpmphmPredictor *source)

  *Make a copy of the OpmphmPredictor.*
- size_t opmphmPredictorWorthOpmphm (size_t n)

  *Heuristic function.*
- int opmphmPredictor (OpmphmPredictor *op, size_t n)

  *Predictor functions.*
- void opmphmPredictorIncCountOpmphm (OpmphmPredictor *op)

*Increases the counter when the OPMPHM was used for the ksLookup (...) .*

- int opmphmPredictorIncCountBinarySearch (OpmphmPredictor *op, size_t n)

  *Increases the counter when the Binary Search was used for the ksLookup (...) .*

## Variables

- const uint16_t opmphmPredictorHistoryMask

  *Order Preserving Minimal Perfect Hash Map Predictor.*

- const size_t opmphmPredictorActionLimit

  *The opmphmPredictorActionLimit define the minimum KeySet size necessary for predictor actions.*

### 573.76.1 Detailed Description

Defines for the Order Preserving Minimal Perfect Hash Map Predictor.

**Copyright**

BSD License (see doc/COPYING or https://www.libelektra.org)

### 573.76.2 Enumeration Type Documentation

#### 573.76.2.1 predictorflag_t

```
enum predictorflag_t
```

OpmphmPredictor Flags.

**Enumerator**

| | |
|---|---|
| OPMPHM_PREDICTOR_FLAG_MMAP_STRUCT | OpmphmPredictor struct lies inside a mmap region. This flag is set for OpmphmPredictor structs inside a mapped region. It prevents erroneous free() calls on these OpmphmPredictors. |
| OPMPHM_PREDICTOR_FLAG_MMAP_↩ PATTERNTABLE | OpmphmPredictor patternTable lies inside a mmap region. This flag is set for OpmphmPredictor patternTables inside a mapped region. It prevents erroneous free() calls on these patternTables. |

### 573.76.3 Function Documentation

#### 573.76.3.1 opmphmPredictor()

```
int opmphmPredictor (
            OpmphmPredictor * op,
            size_t n )
```

Predictor functions.
Predictor functions.
Uses the opmphmPredictorWorthOpmphm (...) to check if the previous sequence of ksLookup (...) invocations without alteration was worth using the OPMPHM. Updates the state with the predictionAutomata and the worth information of the previous history. Alters the history with the worth information and makes the prediction for the next sequence of ksLookup (...) invocations.

**Parameters**

| | |
|---|---|
| *op* | the Predictor |

**Parameters**

| | |
|---|---|
| *n* | the number of elements in the KeySet |

**Return values**

| | |
|---|---|
| *1* | it is worth using the OPMPHM |
| *0* | it is not worth using the OPMPHM |

#### 573.76.3.2 opmphmPredictorCopy()

```
void opmphmPredictorCopy (
            OpmphmPredictor * dest,
            OpmphmPredictor * source )
```

Make a copy of the OpmphmPredictor.

**Parameters**

| | |
|---|---|
| *source* | the OpmphmPredictor source |
| *dest* | the OpmphmPredictor destination |

#### 573.76.3.3 opmphmPredictorDel()

```
void opmphmPredictorDel (
            OpmphmPredictor * op )
```

Deletes the OpmphmPredictor.
Clears and frees all memory in OpmphmPredictor.

**Parameters**

| | |
|---|---|
| *op* | the OpmphmPredictor |

#### 573.76.3.4 opmphmPredictorIncCountBinarySearch()

```
int opmphmPredictorIncCountBinarySearch (
            OpmphmPredictor * op,
            size_t n )  [inline]
```

Increases the counter when the Binary Search was used for the ksLookup (...) .
Prevents also an endless Binary Search usage by a simple heuristic.

**Parameters**

| | |
|---|---|
| *op* | the Predictor |
| *n* | the number of elements in the KeySet |

**Return values**

| | |
|---|---|
| *1* | it is worth using the OPMPHM |
| *0* | it is not worth using the OPMPHM |

#### 573.76.3.5 opmphmPredictorIncCountOpmphm()

```
void opmphmPredictorIncCountOpmphm (
            OpmphmPredictor * op )  [inline]
```

Increases the counter when the OPMPHM was used for the ksLookup (...) .

**Parameters**

| | |
|---|---|
| *op* | the Predictor |

#### 573.76.3.6 opmphmPredictorNew()

```
OpmphmPredictor* opmphmPredictorNew (
            void  )
```

Basic functions.
Basic functions.
Reserves for all possible values of opmphmPredictorHistoryMask two bits to store all 4 states of the Prediction Automata A2. Sets the initial state to 0.

**Return values**

| | |
|---|---|
| *OpmphmPredictor* | * success |
| *NULL* | memory error |

#### 573.76.3.7 opmphmPredictorWorthOpmphm()

```
size_t opmphmPredictorWorthOpmphm (
            size_t n )  [inline]
```

Heuristic function.
Heuristic function.
This easy and fast computable heuristic function tells how many ksLookup (...) invocations without alteration of the KeySet need to be made to justify the OPMPHM usage. This heuristic function is developed and measured by benchmarks.

**Parameters**

| | |
|---|---|
| *n* | the number of elements in the KeySet |

**Return values**

| | |
|---|---|
| *size_t* | the heuristic value |

### 573.76.4 Variable Documentation

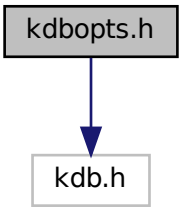
#### 573.76.4.1 opmphmPredictorHistoryMask

```
const uint16_t opmphmPredictorHistoryMask  [extern]
```

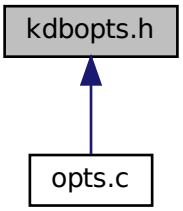
Order Preserving Minimal Perfect Hash Map Predictor.
A modified Global History Register and Global Pattern History Table branch prediction algorithm from
Tse-Yu Yeh and Yale N. Patt "Alternative Implementations of Two-Level Adaptive Branch Prediction" In: The 19th Annual International Symposium on Computer Architecture (1992), pp. 124-134
Maps the event of branch taken or not to a sequence of ksLookup (...) invocations without KeySet alteration that is worth using the OPMPHM or not. The predictor looks at past events to predict the future, to keep track of past events the `lookupCount` and the `ksSize` must be stored. opmphmPredictorHistoryMask defines the length and extraction mask of the global history register interpreted binary it must be a series of 1, with a minimum value of 0x3 and a maximum value of 0x7FFF.
Order Preserving Minimal Perfect Hash Map Predictor.

## 573.77 kdbopts.h File Reference

INTERNAL header for libelektra-opts.
`#include <kdb.h>`
Include dependency graph for kdbopts.h:




This graph shows which files directly or indirectly include this file:




### Functions

- int elektraGetOpts (KeySet *ks, int argc, const char **argv, const char **envp, Key *parentKey)

  *This functions parses a specification of program options, together with a list of arguments and environment variables to extract the option values.*
- char * elektraGetOptsHelpMessage (Key *helpKey, const char *usage, const char *prefix)

  *Extracts the help message from the `helpKey` used in elektraGetOpts().*

## 573.77.1 Detailed Description

INTERNAL header for libelektra-opts.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.77.2 Function Documentation

### 573.77.2.1 elektraGetOpts()

```
int elektraGetOpts (
            KeySet * ks,
            int argc,
            const char ** argv,
            const char ** envp,
            Key * parentKey )
```

This functions parses a specification of program options, together with a list of arguments and environment variables to extract the option values.
The options have to be defined in the metadata of keys in the spec namespace. If an option value is found for any of the given keys, a new key with the same path but inside the proc namespace will be inserted into `ks`. This enables a cascading lookup to find these values.
Take look at https://www.libelektra.org/tutorials/command-line-options for information on how exactly the specification works.
NOTE: Per default option processing DOES NOT stop, when a non-option string is encountered in `argv`. If you want processing to stop, set the metadata `posixly = 1` on `parentKey`.

**Parameters**

| | |
|---|---|
| *ks* | The KeySet containing the specification for the options. |
| *argc* | The number of strings in argv. |
| *argv* | The arguments to be processed. |
| *envp* | A list of environment variables. This needs to be a null-terminated list of strings of the format 'KEY=VALUE'. |
| *parentKey* | The parent key below which the function will search for option specifications. Also used for error reporting. The key will be translated into the spec namespace automatically, e.g. 'user:/test/parent' will be translated into 'spec:/test/parent', before checking against spec keys. |

**Return values**

| | |
|---|---|
| *0* | on success, this is the only case in which `ks` will be modified |
| *-1* | on error, the error will be set as metadata in `errorKey` |
| *1* | if the help option `--help` was found, use elektraGetOptsHelpMessage() access the generated help message |

### 573.77.2.2 elektraGetOptsHelpMessage()

```
char* elektraGetOptsHelpMessage (
            Key * helpKey,
            const char * usage,
            const char * prefix )
```

Extracts the help message from the `helpKey` used in elektraGetOpts().

**Parameters**

| | |
|---|---|
| *helpKey* | The same Key as passed to elektraGetOpts() as parentKey. |
| *usage* | If this is not NULL, it will be used instead of the default usage line. Use elektraGetOptsHelpCommand() to check which command was invoked to get the right usage line. |
| *prefix* | If this is not NULL, it will be inserted between the usage line and the options list. |

**Returns**

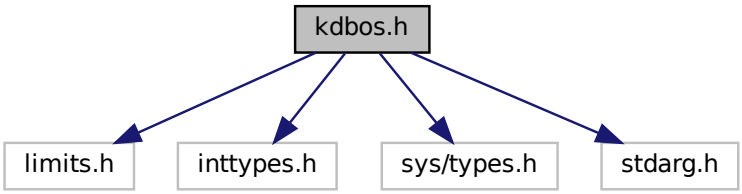
The full help message extracted from `helpKey`, or NULL if no help message was found. The returned string has to be freed with elektraFree().

## 573.78 kdbos.h File Reference

Operating system specific workarounds.

```
#include <limits.h>
#include <inttypes.h>
#include <sys/types.h>
#include <stdarg.h>
```

Include dependency graph for kdbos.h:




### Macros

- #define ELEKTRA_MAX_ARRAY_SIZE (1 + 19 + 20 + 1)

  *The buffer size needed for an array name.*
- #define KDB_FILE_MODE 0600

  *Default Mode.*
- #define KDB_DIR_MODE 0100

  *Default directory mode.*
- #define KDB_MAX_PATH_LENGTH _POSIX_PATH_MAX

  *KDB_MAX_PATH_LENGTH will be the value for longest possible filenames on the system.*

### 573.78.1 Detailed Description

Operating system specific workarounds.
Integer Types must be at least 32bit:
Type Purpose Limits int Integral Fast Type INT_MIN, INT_MAX size_t size of array or string 0, SIZE_MAX ssize_t size with error cond. -1, SSIZE_MAX(<SIZE_MAX) time_t Seconds since 1970 0,.. recommended: 64 bit
Following constants must be defined:
KDB_FILE_MODE the standard mode for keys KDB_DIR_MODE the mode to add (|=) for key directories
Following limits must be defined:
KDB_MAX_PATH_LENGTH the maximum length for a pathname

In addition to the types the ... or va_list must be supported, this is ISO C and should happen by including <stdarg.h>.
Go ahead and write a #ifdef block for your operating system when the POSIX defaults are not ok

**Copyright**



### 573.78.2 Macro Definition Documentation

#### 573.78.2.1 ELEKTRA_MAX_ARRAY_SIZE

```
#define ELEKTRA_MAX_ARRAY_SIZE (1 + 19 + 20 + 1)
```

The buffer size needed for an array name.
The size of the buffer so that the buffer can contain:

- (1) a # in the beginning
- (9) up to 19 underscores are needed as prefix
- (20) a 64bit number has a maximum of 20 digits
- (1) one byte for null termination

E.g. #_____18446744073709551615\0

#### 573.78.2.2 KDB_DIR_MODE

```
#define KDB_DIR_MODE 0100
```

Default directory mode.
This mode will be used for new directories. Will be ORed together with KDB_FILE_MODE to get the permissions of an directory.

#### 573.78.2.3 KDB_FILE_MODE

```
#define KDB_FILE_MODE 0600
```

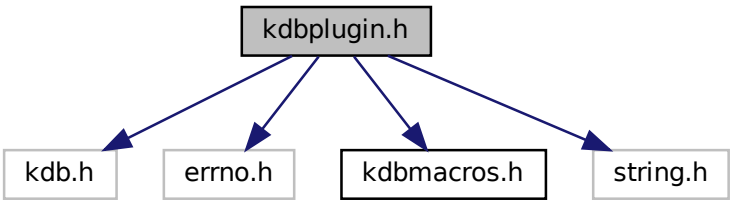
Default Mode.
This mode will be used for new files

## 573.79 kdbplugin.h File Reference

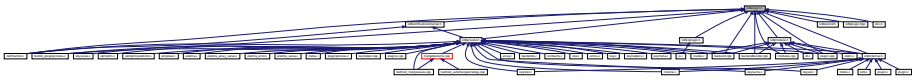
Methods for plugin programing.

```
#include <kdb.h>
#include <errno.h>
#include <kdbmacros.h>
#include <string.h>
```

Include dependency graph for kdbplugin.h:




This graph shows which files directly or indirectly include this file:



## Macros

- #define ELEKTRA_PLUGIN_FUNCTION(function) ELEKTRA_PLUGIN_FUNCTION2 (ELEKTRA_PLUGIN↩ _NAME_C, function)

  *Declare a plugin's function name suitable for compilation variants (see doc/tutorials).*
- #define ELEKTRA_README ELEKTRA_README2 (ELEKTRA_PLUGIN_NAME_C)

  *The filename for inclusion of the readme for compilation variants (see doc/tutorials).*
- #define ELEKTRA_PLUGIN_STATUS_ERROR -1

  *An error occurred inside the plugin function.*
- #define ELEKTRA_PLUGIN_STATUS_SUCCESS 1

  *Everything went fine.*
- #define ELEKTRA_PLUGIN_STATUS_NO_UPDATE 0

  *Everything went fine and the function **did not** update the given keyset/configuration.*
- #define ELEKTRA_PLUGIN_STATUS_CACHE_HIT 2

  *Everything went fine and we have a cache hit.*

## Enumerations

- enum plugin_t {
  ELEKTRA_PLUGIN_OPEN =1 , ELEKTRA_PLUGIN_CLOSE =1<<1 , ELEKTRA_PLUGIN_GET =1<<2 , ELEKTRA_PLUGIN_SET =1<<3 ,
  ELEKTRA_PLUGIN_ERROR =1<<4 , ELEKTRA_PLUGIN_COMMIT =1<<5 , ELEKTRA_PLUGIN_INIT =1<<6 , ELEKTRA_PLUGIN_END =0 }

  *Switches to denote the backend methods.*

## Functions

- Plugin ∗ elektraPluginExport (const char ∗pluginName,...)

  *Allows one to Export Methods for a Plugin.*
- KeySet ∗ elektraPluginGetConfig (Plugin ∗handle)

  *Returns the configuration of that plugin.*
- void elektraPluginSetData (Plugin ∗plugin, void ∗handle)

*Store a pointer to plugin specific data.*

- void * elektraPluginGetData (Plugin *plugin)

  *Get a pointer to the plugin specific data stored before.*

- KeySet * elektraPluginGetGlobalKeySet (Plugin *plugin)

  *Get a pointer to the global keyset.*

- ElektraKdbPhase elektraPluginGetPhase (Plugin *plugin)

  *Returns the current phase of the current KDB operation.*

- Plugin * elektraPluginFromMountpoint (Plugin *plugin, const char *ref)

  *Retrieves the handle for another plugin in the same mountpoint based on a reference.*

- const char * elektraPluginPhaseName (ElektraKdbPhase phase)

  *Gets a string with the name of the given constant for a plugin phase.*

## 573.79.1 Detailed Description

Methods for plugin programing.

**Copyright**

BSD License (see LICENSE.md or `https://www.libelektra.org`)

## 573.79.2 Enumeration Type Documentation

### 573.79.2.1 plugin_t

`enum plugin_t`

Switches to denote the backend methods.
Used in calls to elektraPluginExport().

**Enumerator**

| | |
|---|---|
| ELEKTRA_PLUGIN_OPEN | Next arg is backend for kdbOpen() |
| ELEKTRA_PLUGIN_CLOSE | Next arg is backend for kdbClose() |
| ELEKTRA_PLUGIN_GET | Next arg is backend for kdbGet() |
| ELEKTRA_PLUGIN_SET | Next arg is backend for kdbSet() |
| ELEKTRA_PLUGIN_ERROR | Next arg is backend for kdbError() |
| ELEKTRA_PLUGIN_COMMIT | Next arg is backend for kdbCommit() |
| ELEKTRA_PLUGIN_INIT | Next arg is backend for kdbInit() |
| ELEKTRA_PLUGIN_END | End of arguments |

## 573.79.3 Function Documentation

### 573.79.3.1 elektraPluginFromMountpoint()

```
Plugin* elektraPluginFromMountpoint (
            Plugin * plugin,
            const char * ref )
```

Retrieves the handle for another plugin in the same mountpoint based on a reference.
The plugins of a mountpoint are defined via `system:/elektra/mountpoint/<mp>/pluigns/<ref>` keys in the declaration of the mountpoint. To use this function, you must provide the `<ref>` part as `ref`.

**Parameters**

| | |
|---|---|
| *plugin* | active plugin handle |
| *ref* | reference to another plugin |

**Returns**

the plugin referenced by `ref`

**Return values**

| | |
|---|---|
| *NULL* | if `plugin`, or `ref` are `NULL`, or no plugin was found for `ref` |

#### 573.79.3.2 elektraPluginGetPhase()

```
ElektraKdbPhase elektraPluginGetPhase (
            Plugin * plugin )
```

Returns the current phase of the current KDB operation.
During kdbGet() this will be one of the `ELEKTRA_KDB_GET_PHASE_*` constants and during kdbSet() it will be one of the `ELEKTRA_KDB_SET_PHASE_*` constants.

**Parameters**

| | |
|---|---|
| *plugin* | plugin handle |

**Returns**

current phase

**Return values**

| | |
|---|---|
| *0* | if `plugin` is `NULL` |

#### 573.79.3.3 elektraPluginPhaseName()

```
const char* elektraPluginPhaseName (
            ElektraKdbPhase phase )
```

Gets a string with the name of the given constant for a plugin phase.

**Parameters**

| | |
|---|---|
| *phase* | The ElektraKdbPhase value for which a string representation should be returned |

**Returns**

A string with the name of the given phase. The returned string is a constant value and must never be freed!

**Return values**

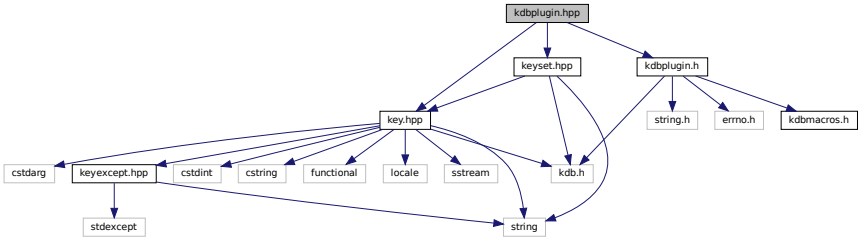
| | |
|---|---|
| *???* | if an unknown value for the phase was given |

# 573.80 kdbplugin.hpp File Reference

Helpers for creating plugins.

```
#include <kdbplugin.h>
#include <key.hpp>
#include <keyset.hpp>
```

Include dependency graph for kdbplugin.hpp:




## 573.80.1 Detailed Description

Helpers for creating plugins.

Make sure to include kdberrors.h before including this file if you want warnings/errors to be added.

Proper usage:

```
using namespace ckdb;
#include <kdberrors.h>
#include <kdbplugin.hpp>
typedef Delegator<elektra::YourPluginClass> YPC;
// then e.g. YPC::open(handle, errorKey);
```

### Copyright

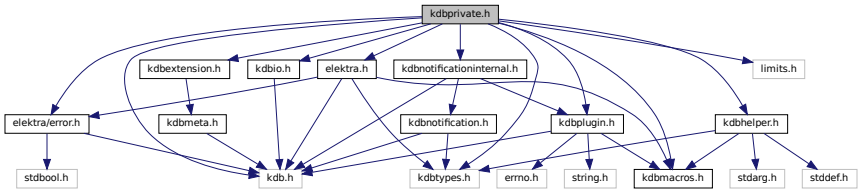
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.81 kdbprivate.h File Reference

Private declarations.

```
#include <elektra.h>
#include <elektra/error.h>
#include <kdb.h>
#include <kdbextension.h>
#include <kdbhelper.h>
#include <kdbio.h>
#include <kdbmacros.h>
#include <kdbnotificationinternal.h>
#include <kdbplugin.h>
#include <kdbtypes.h>
#include <limits.h>
```

Include dependency graph for kdbprivate.h:



This graph shows which files directly or indirectly include this file:

## Macros

- #define KEYSET_SIZE 16

  *The minimal allocation size of a keyset inclusive NULL byte.*
- #define APPROXIMATE_NR_OF_BACKENDS 16

  *Trie optimization.*
- #define KDB_MAX_UCHAR (UCHAR_MAX + 1)

  *The maximum value of unsigned char+1, needed for iteration over trie children/values:*
- #define MAX_LEN_INT 31

  *The maximum of how many characters an integer needs as decimal number.*
- #define KDB_SYSTEM_ELEKTRA "system:/elektra"

  *Backend mounting information.*
- #define KDB_CACHE_PREFIX "system:/elektra/cache"

  *All keys below this are used for cache metadata in the global keyset.*
- #define test_bit(var, bit) (((unsigned long long) (var)) & ((unsigned long long) (bit)))

  *Test a bit.*
- #define set_bit(var, bit) ((var) |= ((unsigned long long) (bit)))

  *Set a bit.*
- #define clear_bit(var, bit) ((var) &= ~((unsigned long long) (bit)))

  *Clear a bit.*

## Typedefs

- typedef struct _BackendData BackendData

  *Holds all data for one backend.*

## Functions

- struct _ElektraDiff * elektraDiffNew (KeySet *addedKeys, KeySet *removedKeys, KeySet *modifiedKeys, KeySet *modifiedNewKeys, Key *parentKey)

  *Create an ElektraDiff.*
- Plugin * elektraPluginOpen (const char *backendname, KeySet *modules, KeySet *config, Key *errorKey)

  *Opens a plugin.*
- size_t elektraPluginGetFunction (Plugin *plugin, const char *name)

  *Retrieves a function exported by a plugin.*
- int initHooks (KDB *kdb, const KeySet *config, KeySet *modules, const KeySet *contract, Key *errorKey)

  *Initializes the hooks stored in the passed KDB handle.*
- void freeHooks (KDB *kdb, Key *errorKey)

  *Uninitializes and frees all hooks in the passed KDB handle.*
- Plugin * elektraFindInternalNotificationPlugin (KDB *kdb)

  *This method looks for the hook plugin 'internalnotification'.*
- int keyReplacePrefix (Key *key, const Key *oldPrefix, const Key *newPrefix)

  *Replaces a prefix of the key name of `key`.*
- Key * elektraKsPopAtCursor (KeySet *ks, elektraCursor pos)

  *Pop key at given cursor position.*
- KeySet * ksRenameKeys (KeySet *config, const char *name)

  *Takes the first key and cuts off this common part for all other keys, instead name will be prepended.*

- ssize_t ksRename (KeySet *ks, const Key *root, const Key *newRoot)

  *Moves all keys below `root` to below `newRoot`.*
- elektraCursor ksFindHierarchy (const KeySet *ks, const Key *root, elektraCursor *end)

  *Searches for the start and optionally end of the key hierarchy rooted at `root` in `ks`.*
- KeySet * ksBelow (const KeySet *ks, const Key *root)

  *Retrieves all Keys from KeySet `ks` that are below or at `root`.*
- ssize_t ksSubtract (KeySet *total, const KeySet *sub)

  *Remove all the keys in `sub` from `total`.*
- ssize_t elektraMemcpy (Key **array1, Key **array2, size_t size)

  *Internal Methods for Elektra.*
- ssize_t elektraMemmove (Key **array1, Key **array2, size_t size)

  *Copies the key array2 into where array1 points.*
- bool elektraKeyNameValidate (const char *name, bool isComplete)

  *Takes an escaped key name and validates it.*
- void elektraKeyNameCanonicalize (const char *name, char **canonicalName, size_t *canonicalSizePtr, size_t offset, size_t *usizePtr)

  *Takes a valid (non-)canonical key name and produces its canonical form.*
- void elektraKeyNameUnescape (const char *name, char *unescapedName)

  *Takes a canonical key name and unescapes it.*
- size_t elektraKeyNameEscapePart (const char *part, char **escapedPart)

  *Takes a single key name part and produces its escaped form.*
- int elektraIsArrayPart (const char *namePart)

  *Checks if the given key name part is an array part.*
- ElektraError * elektraErrorCreate (const char *code, const char *description, const char *module, const char *file, kdb_long_t line)

  *Creates a new ElektraError using the provided values.*
- void elektraErrorAddWarning (ElektraError *error, ElektraError *warning)

  *Adds a warning to an existing ElektraError struct.*
- ElektraError * elektraErrorFromKey (Key *key)

  *Extracts the error and all warnings from the given key.*
- ElektraError * elektraErrorKeyNotFound (const char *keyname)

  *Creates a "Key not found" error.*
- ElektraError * elektraErrorWrongType (const char *keyname, KDBType expectedType, KDBType actualType)

  *Creates a "Wrong type" error.*
- ElektraError * elektraErrorNullError (const char *function)

  *Creates a "Null error argument" error.*

### 573.81.1 Detailed Description

Private declarations.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.81.2 Macro Definition Documentation

#### 573.81.2.1 clear_bit

```
#define clear_bit(
              var,
              bit ) ((var) &= ~((unsigned long long) (bit)))
```

Clear a bit.

**See also**

set_bit()

#### 573.81.2.2 KDB_MAX_UCHAR

```
#define KDB_MAX_UCHAR (UCHAR_MAX + 1)
```

The maximum value of unsigned char+1, needed for iteration over trie children/values:
for (i=0; i<KDB_MAX_UCHAR; ++i)

#### 573.81.2.3 KDB_SYSTEM_ELEKTRA

```
#define KDB_SYSTEM_ELEKTRA "system:/elektra"
```

Backend mounting information.
This key directory tells you where each backend is mounted to which mountpoint.

#### 573.81.2.4 KEYSET_SIZE

```
#define KEYSET_SIZE 16
```

The minimal allocation size of a keyset inclusive NULL byte.
ksGetAlloc() will return one less because it says how much can actually be stored.

#### 573.81.2.5 set_bit

```
#define set_bit(
              var,
              bit ) ((var) |= ((unsigned long long) (bit)))
```

Set a bit.

**See also**

clear_bit()

#### 573.81.2.6 test_bit

```
#define test_bit(
              var,
              bit ) (((unsigned long long) (var)) & ((unsigned long long) (bit)))
```

Test a bit.

**See also**

set_bit(), clear_bit()

### 573.81.3 Typedef Documentation

#### 573.81.3.1 BackendData

```
typedef struct _BackendData BackendData
```

Holds all data for one backend.
This struct is used for the key values in _KDB.backends

## 573.81.4 Function Documentation

### 573.81.4.1 elektraDiffNew()

```
struct _ElektraDiff* elektraDiffNew (
            KeySet * addedKeys,
            KeySet * removedKeys,
            KeySet * modifiedKeys,
            KeySet * modifiedNewKeys,
            Key * parentKey )
```

Create an ElektraDiff.
The returned ElektraDiff contains the same KeySets that are passed in, so be sure to `ksIncRef` them if you plan to use them after deleting the ElektraDiff.
Same goes for the parent key, please use `keyIncRef` if you plan to use it after deleting the ElektraDiff

**Parameters**

| | |
|---|---|
| *addedKeys* | the added keys |
| *removedKeys* | the removed keys |
| *modifiedKeys* | the modified keys |
| *modifiedNewKeys* | the modified keys with the new values |
| *parentKey* | the parent key |

**Returns**

ElektraDiff with the provided parameters

### 573.81.4.2 elektraErrorAddWarning()

```
void elektraErrorAddWarning (
            ElektraError * error,
            ElektraError * warning )
```

Adds a warning to an existing ElektraError struct.
If you want to report a warning without an error, create a dummy error with elektraErrorPureWarning() and then add a warning to it.

**Parameters**

| | |
|---|---|
| *error* | The error to which `warning` shall be added. |
| *warning* | The warning to add. Once added it is owned by `error`. DO NOT call elektraErrorReset() on it afterwards. |

### 573.81.4.3 elektraErrorCreate()

```
ElektraError* elektraErrorCreate (
            const char * code,
            const char * description,
            const char * module,
            const char * file,
            kdb_long_t line )
```

Creates a new ElektraError using the provided values.
The returned value will be allocated with elektraCalloc().

**Parameters**

| | |
|---|---|
| *code* | The error code of the error. Will be copied and stored in the struct. |
| *description* | The description of the error. Will be copied and stored in the struct. |
| *module* | The module that raised the error. Will be copied and stored in the struct. |
| *file* | The file that raised the error. Will be copied and stored in the struct. |
| *line* | The line in which the error was raised. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.81.4.4 elektraErrorFromKey()

```
ElektraError* elektraErrorFromKey (
            Key * key )
```

Extracts the error and all warnings from the given key.
If no error exists, a pure warning error will be used.

**See also**

elektraErrorPureWarning

**Note**

Use the functions in src/libs/elektra/errors.c to add errors to a key.

**Parameters**

| | |
|---|---|
| *key* | The to extract error and warnings from. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.81.4.5 elektraErrorKeyNotFound()

```
ElektraError* elektraErrorKeyNotFound (
            const char * keyname )
```

Creates a "Key not found" error.

**Parameters**

| | |
|---|---|
| *keyname* | The name of the key that wasn't found. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.81.4.6 elektraErrorNullError()

```
ElektraError* elektraErrorNullError (
            const char * function )
```

Creates a "Null error argument" error.

**Parameters**

| | |
|---|---|
| *function* | The name of the function that was called with a null pointer error argument. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.81.4.7 elektraErrorWrongType()

```
ElektraError* elektraErrorWrongType (
            const char * keyname,
            KDBType expectedType,
            KDBType actualType )
```

Creates a "Wrong type" error.

**Parameters**

| | |
|---|---|
| *keyname* | The name of the key that had the wrong type. |
| *expectedType* | The type that was expected. |
| *actualType* | The type that was actually found. |

**Returns**

A newly allocated ElektraError (free with elektraErrorReset()).

#### 573.81.4.8 elektraFindInternalNotificationPlugin()

```
Plugin* elektraFindInternalNotificationPlugin (
            KDB * kdb )
```

This method looks for the hook plugin 'internalnotification'.

**Parameters**

| | |
|---|---|
| *kdb* | the KDB instance in which to search |

**Returns**

NULL if not loaded, pointer to the plugin otherwise

#### 573.81.4.9 elektraKsPopAtCursor()

```
Key* elektraKsPopAtCursor (
            KeySet * ks,
            elektraCursor pos )
```

Pop key at given cursor position.

**Parameters**

| | |
|---|---|
| *ks* | the keyset to pop key from |
| *c* | where to pop |

**Returns**

the popped key

**Return values**

| | |
|---|---|
| *0* | if ks is 0 |

### 573.81.4.10 elektraMemcpy()

```
ssize_t elektraMemcpy (
            Key ** array1,
            Key ** array2,
            size_t size )
```

Internal Methods for Elektra.
To use them:
`#include <kdbinternal.h>`
There are some areas where libraries have to reimplement some basic functions to archive support for non-standard systems, for testing purposes or to provide a little more convenience. Copies the key array2 into where array1 points.
It copies size elements.
Overlapping is prohibited, use elektraMemmove() instead.

**Parameters**

| | |
|---|---|
| *array1* | the destination |
| *array2* | the source |
| *size* | how many pointer to Keys to copy |

**Return values**

| | |
|---|---|
| *-1* | on null pointers |
| *0* | if nothing was done |

**Returns**

size how many keys were copied

### 573.81.4.11 elektraMemmove()

```
ssize_t elektraMemmove (
            Key ** array1,
            Key ** array2,
            size_t size )
```

Copies the key array2 into where array1 points.
It copies size elements.
Overlapping is ok. If they do not overlap consider elektraMemcpy() instead.

**Parameters**

| | |
|---|---|
| *array1* | the destination |
| *array2* | the source |
| *size* | how many pointer to Keys to copy |

**Return values**

| | |
|---|---|
| *-1* | on null pointers |
| *0* | if nothing was done |

**Returns**

size how many keys were copied

#### 573.81.4.12 elektraPluginGetFunction()

```
size_t elektraPluginGetFunction (
            Plugin * plugin,
            const char * name )
```

Retrieves a function exported by a plugin.

**Parameters**

| | |
|---|---|
| *plugin* | Plugin handle |
| *name* | Function name. Must be a valid key name suffix. May not contain the sequence '..' |

**Returns**

Pointer to function. NULL if function not found or not enough memory available

#### 573.81.4.13 elektraPluginOpen()

```
Plugin* elektraPluginOpen (
            const char * name,
            KeySet * modules,
            KeySet * config,
            Key * errorKey )
```

Opens a plugin.

The config will be used as is. So be sure to transfer ownership of the config to it, with e.g. ksDup(). elektraPlugin↩ Close() will delete the config.

**Returns**

a pointer to a new created plugin or 0 on error

#### 573.81.4.14 freeHooks()

```
void freeHooks (
            KDB * kdb,
            Key * errorKey )
```

Uninitializes and frees all hooks in the passed KDB handle.

**Parameters**

| | |
|---|---|
| *kdb* | the KDB handle where the hooks should be freed |
| *errorKey* | the key which holds errors and warnings which were issued |

#### 573.81.4.15 initHooks()

```
int initHooks (
            KDB * kdb,
            const KeySet * config,
            KeySet * modules,
            const KeySet * contract,
            Key * errorKey )
```

Initializes the hooks stored in the passed KDB handle.
If the handle already contains initialized hooks, they will be reinitialized, including unloading and loading of their plugins. Parameters `config` and `contract` will be used to determine which hooks to populate.

**Parameters**

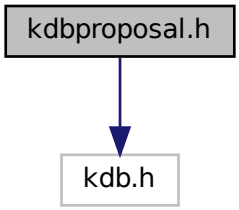
| | |
|---|---|
| *kdb* | the KDB instance where the hooks should be initialized |
| *config* | KeySet containing the current config in `system:/elektra` namespace |
| *modules* | the current list of loaded modules |
| *contract* | the contract passed to `kdbOpen` |
| *errorKey* | the key which holds errors and warnings which were issued |

**Returns**

0 on success, -1 on failure

## 573.82 kdbproposal.h File Reference

Proposed declarations.
`#include <kdb.h>`
Include dependency graph for kdbproposal.h:




### 573.82.1 Detailed Description

Proposed declarations.
These functions are likely not API/ABI stable.

**Copyright**

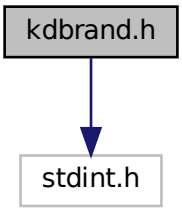
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.83 kdbrand.h File Reference

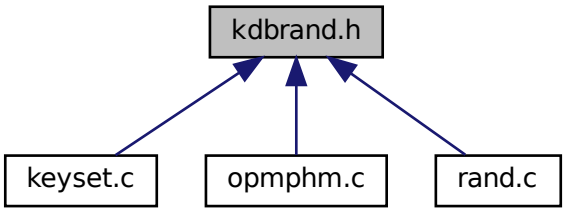
Defines for Rand.

```
#include <stdint.h>
```

Include dependency graph for kdbrand.h:




This graph shows which files directly or indirectly include this file:




## Functions

- void elektraRand (int32_t *seed)

  *Non cryptographic pseudo random number generator.*
- int32_t elektraRandGetInitSeed (void)

  *Random initial seed generator.*

### 573.83.1 Detailed Description

Defines for Rand.

**Copyright**

BSD License (see doc/COPYING or https://www.libelektra.org)

### 573.83.2 Function Documentation

#### 573.83.2.1 elektraRand()

```
void elektraRand (
            int32_t * seed )
```

Non cryptographic pseudo random number generator.

By Ray Gardner www8.cs.umu.se/~isak/snippets/rg_rand.c
based on "Random Number Generators: Good Ones Are Hard to Find", S.K. Park and K.W. Miller, Communications of the ACM 31:10 (Oct 1988), and "Two Fast Implementations of the 'Minimal Standard' Random Number Generator", David G. Carta, Comm. ACM 33, 1 (Jan 1990), p. 87-88
linear congruential generator f(z) = 16807 z mod (2 ∗∗ 31 - 1)
uses L. Schrage's method to avoid overflow problems
Make sure the initial seed is: 0 < seed < ELEKTRARANDMAX

**Parameters**

| | |
|---|---|
| *seed* | a pointer to the seed |

#### 573.83.2.2 elektraRandGetInitSeed()

```
int32_t elektraRandGetInitSeed (
            void  )
```

Random initial seed generator.
Generates a random initial seed for the `elektraRand (...)` function. Two invocations in the same second return the same random initial seed, due to the usage of `time (0)`.

**Return values**

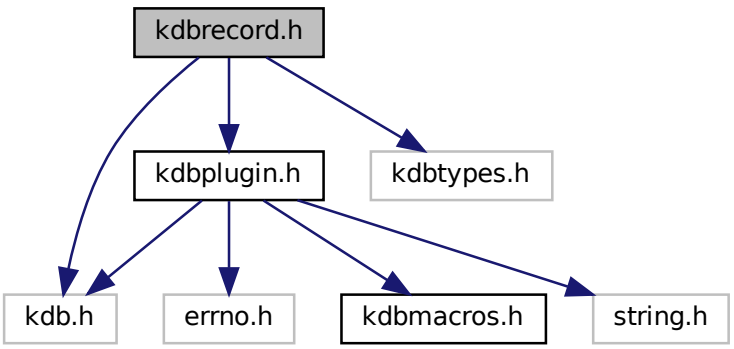
| | |
|---|---|
| *random* | initial seed |

## 573.84 kdbrecord.h File Reference

Defines for record.

```
#include <kdb.h>
#include <kdbdiff.h>
#include <kdbplugin.h>
#include <kdbtypes.h>
```

Include dependency graph for kdbrecord.h:




### Functions

- bool elektraRecordEnableRecording (KDB ∗handle, const Key ∗parentKey, Key ∗errorKey)

*Enable session recording.*

- bool elektraRecordDisableRecording (KDB ∗handle, Key ∗errorKey)

  *Disable session recording.*

- bool elektraRecordResetSession (KDB ∗handle, Key ∗errorKey)

  *Clears all recorded data.*

- bool elektraRecordRecord (KDB ∗handle, KDB ∗sessionStorageHandle, KeySet ∗newKeys, Key ∗parentKey, Key ∗errorKey)

  *Diff and record changes to the KDB instance in* `handle`*.*

- bool elektraRecordUndo (KDB ∗handle, KDB ∗sessionStorageHandle, Key ∗parentKey, Key ∗errorKey)

  *Undo changes that were recorded in the current recording session.*

- bool elektraRecordRemoveKeys (KDB ∗handle, KeySet ∗toRemove, bool recursive, Key ∗errorKey)

  *Removes the provided keys from the recording session.*

- bool elektraRecordIsActive (KDB ∗handle)

  *Check whether session recording is active in the given KDB instance.*

## 573.84.1 Detailed Description

Defines for record.

**Copyright**

BSD License (see doc/COPYING or `https://www.libelektra.org`)

## 573.84.2 Function Documentation

### 573.84.2.1 elektraRecordDisableRecording()

```
bool elektraRecordDisableRecording (
            KDB * handle,
            Key * errorKey )
```

Disable session recording.
This affects both the given `handle` as well as every KDB instance that will be created after this method has been called.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance to use |
| *errorKey* | used for reporting errors and warnings. As usual, they will be found as meta keys attached to this key. |

**Return values**

| | |
|---|---|
| *true* | - recording has been disabled successfully @retfal false - there was an error disabling recording - see `errorKey` for further details |

### 573.84.2.2 elektraRecordEnableRecording()

```
bool elektraRecordEnableRecording (
            KDB * handle,
            const Key * parentKey,
            Key * errorKey )
```

Enable session recording.
This affects both the given `handle` as well as every KDB instance that will be created after this method has been

called.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance to use |
| *parentKey* | recording will be enabled for every key that is same or below this key |
| *errorKey* | used for reporting errors and warnings. As usual, they will be found as meta keys attached to this key. |

**Return values**

| | |
|---|---|
| *true* | - recording has been enabled successfully @retfal false - there was an error enabling recording - see `errorKey` for further details |

#### 573.84.2.3 elektraRecordIsActive()

```
bool elektraRecordIsActive (
            KDB * handle )
```

Check whether session recording is active in the given KDB instance.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance to check |

**Return values**

| | |
|---|---|
| | |

#### 573.84.2.4 elektraRecordRecord()

```
bool elektraRecordRecord (
            KDB * handle,
            KDB * sessionStorageHandle,
            KeySet * newKeys,
            Key * parentKey,
            Key * errorKey )
```

Diff and record changes to the KDB instance in `handle`.
This function is mainly intended for use in the recorder plugin.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance that changes occured on |
| *sessionStorageHandle* | the KDB instances that shall be used to persist the session diff. You can use the same as for `handle`. |
| *newKeys* | the keyset with changed keys |
| *parentKey* | only changes same or below this key will be determined |
| *errorKey* | used for reporting errors and warnings. As usual, they will be found as meta keys attached to this key. |

**Return values**

| | |
|---|---|
| *true* | - changes were recorded successfully @retfal false - there was an error during recording - see `errorKey` for further details |

#### 573.84.2.5 elektraRecordRemoveKeys()

```
bool elektraRecordRemoveKeys (
            KDB * handle,
            KeySet * toRemove,
            bool recursive,
            Key * errorKey )
```

Removes the provided keys from the recording session.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance to use for accessing recording data. |
| *toRemove* | the keys in this keyset will be removed. |
| *recursive* | if `true`, also the keys below every specified key will be removed. |
| *errorKey* | used for reporting errors and warnings. As usual, they will be found as meta keys attached to this key. |

**Return values**

| | |
|---|---|
| *true* | - removal was successful @retfal false - there was an error during the removal operation - see `errorKey` for further details |

#### 573.84.2.6 elektraRecordResetSession()

```
bool elektraRecordResetSession (
            KDB * handle,
            Key * errorKey )
```

Clears all recorded data.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance to use |
| *errorKey* | used for reporting errors and warnings. As usual, they will be found as meta keys attached to this key. |

**Return values**

| | |
|---|---|
| *true* | - recording session has been cleared successfully @retfal false - there was an error clearing the recording session - see `errorKey` for further details |

#### 573.84.2.7 elektraRecordUndo()

```
bool elektraRecordUndo (
            KDB * handle,
            KDB * sessionStorageHandle,
            Key * parentKey,
```

```
Key * errorKey )
```

Undo changes that were recorded in the current recording session.
After executing this function, the state of KDB should be the same as it was before starting the recording session.

**Parameters**

| | |
|---|---|
| *handle* | the KDB instance to use for accessing configuration data |
| *sessionStorageHandle* | the KDB instance to use for accessing recording data. This can be the same as for `handle` |
| *parentKey* | only changes same or below this key are undone. |
| *errorKey* | used for reporting errors and warnings. As usual, they will be found as meta keys attached to this key. |

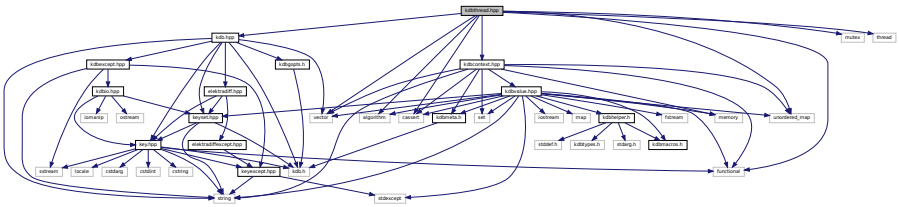
**Return values**

| | |
|---|---|
| *true* | - changes were undone successfully @retfal false - there was an error during the undo operation - see `errorKey` for further details |

## 573.85 kdbthread.hpp File Reference

```
#include <kdbcontext.hpp>
#include <kdb.hpp>
#include <algorithm>
#include <cassert>
#include <functional>
#include <mutex>
#include <thread>
#include <unordered_map>
#include <vector>
```

Include dependency graph for kdbthread.hpp:



### Classes

- class kdb::ThreadSubject

  *Subject from Observer pattern for ThreadContext.*
- struct kdb::PerContext

  *A data structure that is stored by context inside the Coordinator.*
- class kdb::Coordinator

  *Thread safe coordination of ThreadContext per Threads.*

### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### Typedefs

- typedef std::unordered_map< std::string, LayerAction > kdb::LayerMap

  *A vector of layers.*

### 573.85.1 Detailed Description

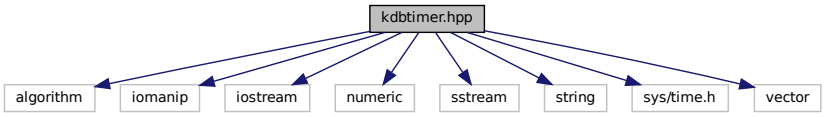
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

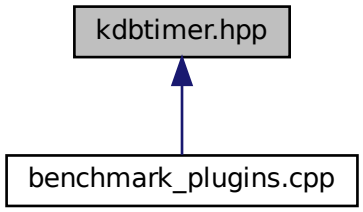
## 573.86 kdbtimer.hpp File Reference

```
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <numeric>
#include <sstream>
#include <string>
#include <sys/time.h>
#include <vector>
```

Include dependency graph for kdbtimer.hpp:




This graph shows which files directly or indirectly include this file:




### 573.86.1 Detailed Description

**Copyright**

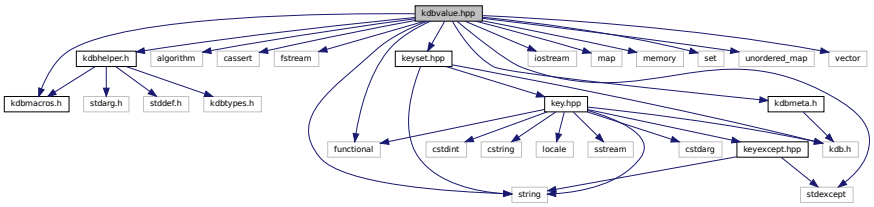
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.87 kdbvalue.hpp File Reference

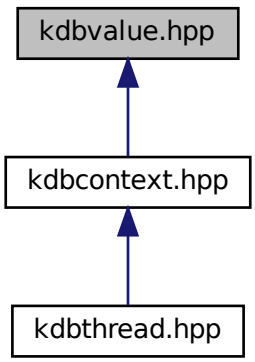
```
#include <kdbmacros.h>
#include <algorithm>
```

```
#include <cassert>
#include <fstream>
#include <functional>
#include <iostream>
#include <kdbmeta.h>
#include <map>
#include <memory>
#include <set>
#include <stdexcept>
#include <string>
#include <unordered_map>
#include <vector>
#include <kdbhelper.h>
#include <keyset.hpp>
```

Include dependency graph for kdbvalue.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::none_t

  *This type is being used as bottom type that always fails.*
- class kdb::Layer

  *Base class for all layers.*
- class kdb::Wrapped

  *Everything implementing this interface can be used as layer.*
- class kdb::ValueObserver

*Base class for values to be observed.*

- struct kdb::Command

  *Used by contexts for callbacks (to run code using a mutex).*
- class kdb::DefaultGetPolicy

  *Implements lookup with spec.*
- class kdb::DefaultSetPolicy

  *Implements creating user:/ key when key is not found.*
- class kdb::GetPolicyIs< Policy >

  *Needed by the user to set one of the policies.*
- class kdb::SetPolicyIs< Policy >

  *Needed by the user to set one of the policies.*
- class kdb::ContextPolicyIs< Policy >

  *Needed by the user to set one of the policies.*
- class kdb::WritePolicyIs< Policy >

  *Needed by the user to set one of the policies.*
- class kdb::ObserverPolicyIs< Policy >

  *Needed by the user to set one of the policies.*
- class kdb::LockPolicyIs< Policy >

  *Needed by the user to set one of the policies.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

## Functions

- bool kdb::operator< (ValueObserver const &lhs, ValueObserver const &rhs)

  *Needed to put a ValueObserver in a map.*

### 573.87.1 Detailed Description

**Copyright**

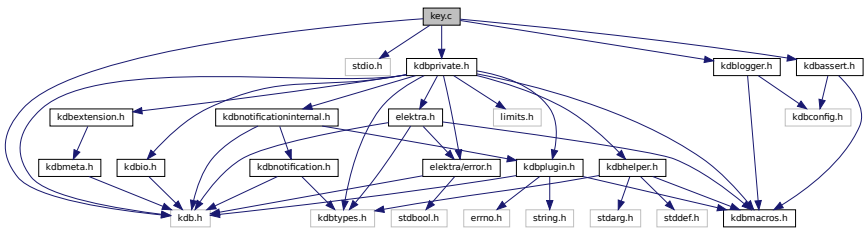
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.88 key.c File Reference

Methods for Key manipulation.

```
#include "kdblogger.h"
#include <stdio.h>
#include "kdb.h"
#include "kdbprivate.h"
#include <kdbassert.h>
```

Include dependency graph for key.c:



## Functions

- Key * keyNew (const char *name,...)

  *A practical way to fully create a Key object in one step.*
- Key * keyVNew (const char *name, va_list va)

  *A practical way to fully create a Key object in one step.*
- Key * keyCopy (Key *dest, const Key *source, elektraCopyFlags flags)

  *Copy or clear a key.*
- int keyDel (Key *key)

  *A destructor for Key objects.*
- int keyClear (Key *key)

  *Will clear all internal data of a Key.*
- uint16_t keyIncRef (Key *key)

  *Increment the reference counter of a Key object.*
- uint16_t keyDecRef (Key *key)

  *Decrement the reference counter of a Key object.*
- uint16_t keyGetRef (const Key *key)

  *Return the current reference counter value of a Key object.*
- int keyLock (Key *key, elektraLockFlags what)

  *Permanently lock parts of a Key.*
- int keyIsLocked (const Key *key, elektraLockFlags what)

  *Checks which parts of a Key are locked.*

### 573.88.1 Detailed Description

Methods for Key manipulation.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.88.2 Function Documentation

#### 573.88.2.1 keyVNew()

```
Key* keyVNew (
            const char * name,
            va_list va )
```

A practical way to fully create a Key object in one step.
To just get a key object, simple do:

```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/some/example", KEY_END);
// work with it
keyDel (k);
```

keyNew() allocates memory for a key object and keyDel() cleans everything up.
If you want the key object to contain a name, value, comment and other meta info read on.

**Note**

When you already have a key with similar properties its easier to keyDup() the key.

You can call keyNew() in many different ways depending on the attribute tags you pass as parameters. Tags are represented as elektraKeyFlags values, and tell keyNew() which Key attribute comes next. The Key attribute tags are the following:

- KEY_VALUE
  Next parameter is a pointer to the value that will be used. If no KEY_BINARY was used before, a string is assumed.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex0",
        KEY_VALUE, "some data",     // set a string value
        KEY_END);                   // end of args
```

- KEY_SIZE
  Define a maximum length of the value. This is only used when setting a binary key.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex1",
        KEY_SIZE, 4,                // has no effect on strings
        KEY_VALUE, "some data",     // set a string value
        KEY_END);                   // end of args
```

- KEY_META
  Next two parameter is a metaname and a metavalue. See keySetMeta().
```
Key *k=keyNew("user:/tmp/ex3",
        KEY_META, "comment/#0", "a comment",  // with a comment
        KEY_META, "owner", "root",          // and an owner
        KEY_META, "special", "yes",         // and any other metadata
        KEY_END);                   // end of args
```

- KEY_END
  Must be the last parameter passed to keyNew(). It is always required, unless the `keyName` is 0.

- KEY_FLAGS
  Bitwise disjunction of flags, which don't require one or more values. recommended way to set multiple flags. overrides previously defined flags.
```
Key *k=keyNew("user:/tmp/ex3",
        KEY_BINARY,                   // binary key
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",                 // value that will be truncated in 7 bytes
        KEY_END);                       // end of args
```

- KEY_BINARY
  Allows one to change the key to a binary key. Make sure that you also pass KEY_SIZE before you set the value. Otherwise it will be cut off with first \0 in the string. So this flag toggle from keySetString() to keySetBinary(). If no value (nor size) is given, it will be a NULL key.
```
// Create and initialize a key with a name and nothing else
Key *k=keyNew("user:/tmp/ex2",
        KEY_BINARY,
        KEY_SIZE, 4,                // now the size is important
        KEY_VALUE, "some data",     // sets the binary value ("some")
        KEY_END);                   // end of args
Key *k=keyNew("user:/tmp/ex4",
        KEY_BINARY,                   // key type
        KEY_SIZE, 7,                    // assume binary length 7
        KEY_VALUE, "some data",                 // value that will be truncated in 7 bytes
        KEY_META, "comment/#0", "value is truncated",
        KEY_END);                       // end of args
```

**Precondition**

`name` is a valid Key name

Variable arguments are a valid combination

**Postcondition**

returns a new, fully initialized Key object with the valid Key name and all data given by variable arguments

**Parameters**

| | |
|---|---|
| *name* | a valid name to the key (see keySetName()) |

**Returns**

a pointer to a new allocated and initialized Key object.

**Return values**

| | |
|---|---|
| *NULL* | on allocation error or if an invalid `name` was passed (see keySetName()). |

**Since**

1.0.0

**See also**

keyDel() for deallocating a created Key object
keySetName() for rules about which names are considered valid

**Precondition**

caller must use va_start and va_end on va

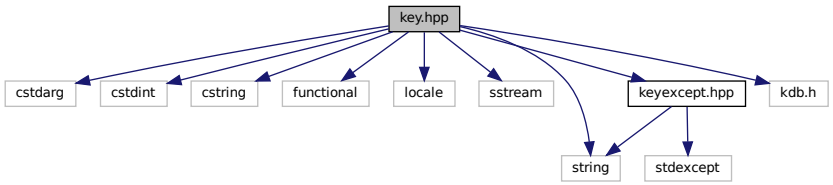
**Parameters**

| | |
|---|---|
| *va* | the variadic argument list |

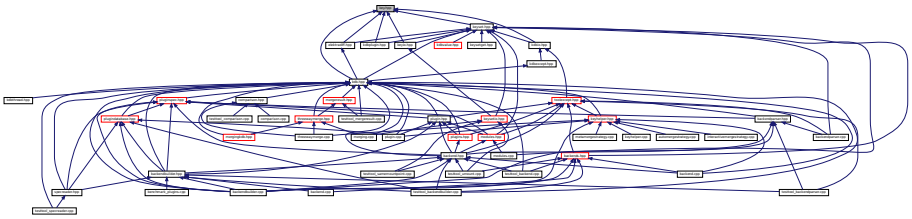
## 573.89 key.hpp File Reference

```
#include <cstdarg>
#include <cstdint>
#include <cstring>
#include <functional>
#include <locale>
#include <sstream>
#include <string>
#include <keyexcept.hpp>
#include <kdb.h>
```

Include dependency graph for key.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- class kdb::Key

  *Key is an essential class that encapsulates key name , value and metainfo .*
- class kdb::NameIterator

  *For C++ forward Iteration over Names.*
- class kdb::NameReverseIterator

  *For C++ reverse Iteration over Names.*
- struct std::hash< kdb::Key >

  *Support for putting Key in a hash.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.89.1 Detailed Description

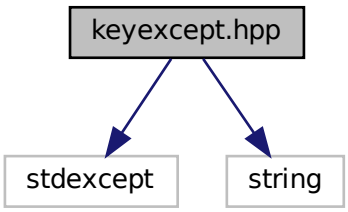
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

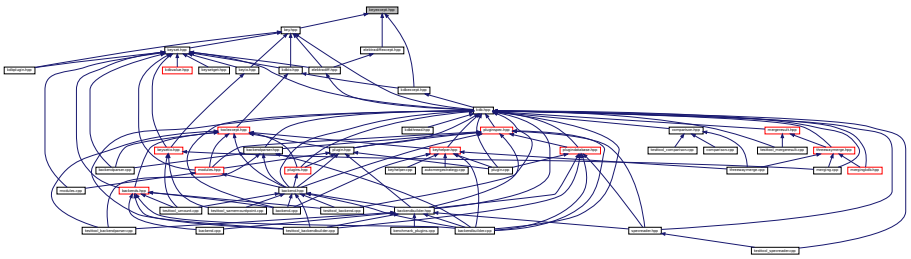
# 573.90 keyexcept.hpp File Reference

```
#include <stdexcept>
#include <string>
```

Include dependency graph for keyexcept.hpp:




This graph shows which files directly or indirectly include this file:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.90.1 Detailed Description

**Copyright**

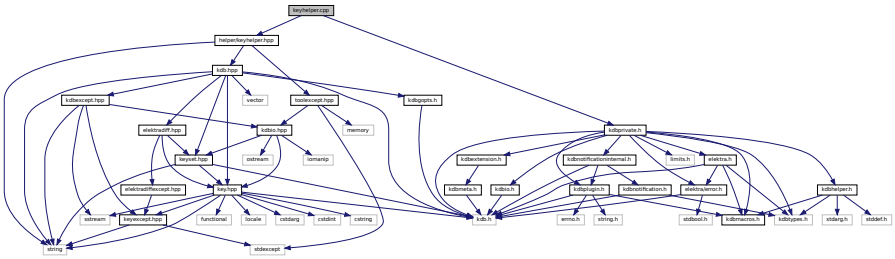
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.91 keyhelper.cpp File Reference

Key helper functions.

```
#include <helper/keyhelper.hpp>
#include <kdbprivate.h>
```

Include dependency graph for keyhelper.cpp:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- std::string kdb::tools::helper::rebasePath (const Key &key, const Key &oldParent, const Key &newParent)

  *Rebases the relative path of the passed key from the old parent to the new parent and returns the new path.*
- Key kdb::tools::helper::rebaseKey (const Key &key, const Key &oldParent, const Key &newParent)

  *Rebases the supplied key from the old parent to the new parent.*
- void kdb::tools::helper::removeNamespace (Key &key)

  *Removes the namespace.*
- Key kdb::tools::helper::prependNamespace (Key const &root, std::string const &ns)

  *Prepends the namespace to the key.*
- KeySet kdb::tools::helper::prependNamespace (KeySet const &resultKeys, std::string const &ns)

  *Prepends the namespace to each key of the key set.*
- void kdb::tools::helper::copyAllMeta (KeySet &to, KeySet const &from)

  *Copies all metadata of each key in from to their pendant in to.*
- Key kdb::tools::helper::commonKeyName (Key key1, Key key2)

  *Find common name between two keys.*

### 573.91.1 Detailed Description

Key helper functions.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.91.2 Function Documentation

#### 573.91.2.1 commonKeyName()

```
Key kdb::tools::helper::commonKeyName (
            Key key1,
            Key key2 )
```

Find common name between two keys.

**Returns**

the longest common name found

#### 573.91.2.2 copyAllMeta()

```
void kdb::tools::helper::copyAllMeta (
            KeySet & to,
            KeySet const & from )
```

Copies all metadata of each key in from to their pendant in to.
The pendant is determined by ksLookup.

**Parameters**

| | |
|---|---|
| *to* | metadata is copied to keys in this key set |
| *from* | metadata is copied from keys in this key set |

#### 573.91.2.3 prependNamespace() [1/2]

```
Key kdb::tools::helper::prependNamespace (
            Key const & root,
            std::string const & ns )
```

Prepends the namespace to the key.

**Parameters**

| | |
|---|---|
| *root* | is the key where the namespace will be prepended |
| *ns* | is the namespace to be prepended |

**Returns**

a duplicate of root with the new namespace

#### 573.91.2.4 prependNamespace() [2/2]

```
KeySet kdb::tools::helper::prependNamespace (
```

```
            KeySet const & resultKeys,
            std::string const & ns )
```

Prepends the namespace to each key of the key set.

**Parameters**

| | |
|---|---|
| *resultKeys* | which keys to prepend the namepsace to |
| *ns* | is the namespace to be prepended to each key |

**Returns**

a keyset where each key has the new namespace

#### 573.91.2.5 rebaseKey()

```
Key kdb::tools::helper::rebaseKey (
            const Key & key,
            const Key & oldParent,
            const Key & newParent )
```

Rebases the supplied key from the old parent to the new parent.

**See also**

ThreeWayMerge::rebasePath

**Parameters**

| | |
|---|---|
| *key* | the key to be rebased |
| *oldParent* | the old parent of the key |
| *newParent* | the new parent of the key |

**Returns**

a rebased copy of the supplied key

**Exceptions**

| | |
|---|---|
| *InvalidRebaseException* | if the key is not below the old parent |

#### 573.91.2.6 rebasePath()

```
string kdb::tools::helper::rebasePath (
            const Key & key,
            const Key & oldParent,
            const Key & newParent )
```

Rebases the relative path of the passed key from the old parent to the new parent and returns the new path.
For example a key /user/example/config/key1 with the oldparent /user/example and the new parent /user/newexample/newpath would result in /user/newexample/newpath/config/key1
If any of the parent keys is a cascading key the namespace of the key to be rebased is assumed instead.

**Parameters**

| | |
|---|---|
| *key* | the key whose path should be rebased |

**Parameters**

| | |
|---|---|
| *oldParent* | the old parent of the key |
| *newParent* | the new parent of the key |

**Returns**

the rebased path

**Exceptions**

| | |
|---|---|
| *InvalidRebaseException* | if the key is not below the old parent |

#### 573.91.2.7 removeNamespace()

```
void kdb::tools::helper::removeNamespace (
            Key & key )
```

Removes the namespace.

**Parameters**

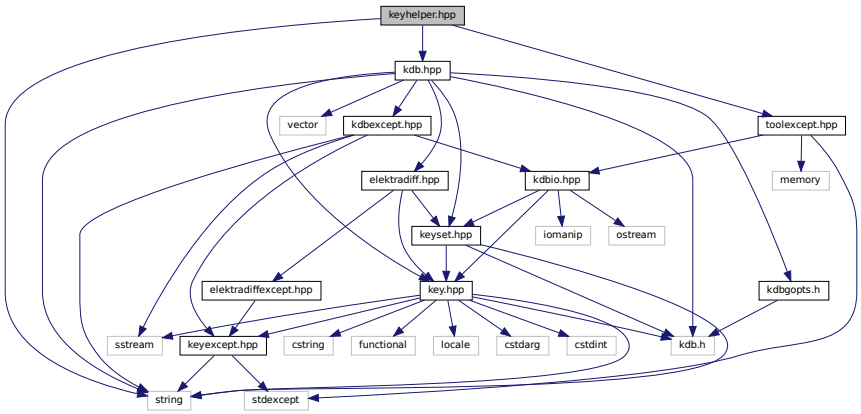
| | |
|---|---|
| *key* | will be made to a cascading key |

## 573.92 keyhelper.hpp File Reference

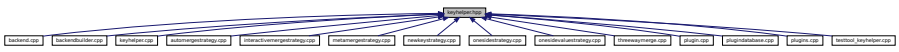
Key helper functions.

```
#include <kdb.hpp>
#include <string>
#include <toolexcept.hpp>
```

Include dependency graph for keyhelper.hpp:




This graph shows which files directly or indirectly include this file:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- Key kdb::tools::helper::rebaseKey (const Key &key, const Key &oldParent, const Key &newParent)

  *Rebases the supplied key from the old parent to the new parent.*
- std::string kdb::tools::helper::rebasePath (const Key &key, const Key &oldParent, const Key &newParent)

  *Rebases the relative path of the passed key from the old parent to the new parent and returns the new path.*
- void kdb::tools::helper::removeNamespace (Key &key)

  *Removes the namespace.*
- Key kdb::tools::helper::prependNamespace (Key const &root, std::string const &ns)

  *Prepends the namespace to the key.*
- KeySet kdb::tools::helper::prependNamespace (KeySet const &resultKeys, std::string const &ns)

  *Prepends the namespace to each key of the key set.*
- void kdb::tools::helper::copyAllMeta (KeySet &to, KeySet const &from)

  *Copies all metadata of each key in from to their pendant in to.*
- Key kdb::tools::helper::commonKeyName (Key key1, Key key2)

  *Find common name between two keys.*

### 573.92.1 Detailed Description

Key helper functions.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.92.2 Function Documentation

#### 573.92.2.1 commonKeyName()

```
Key kdb::tools::helper::commonKeyName (
            Key key1,
            Key key2 )
```

Find common name between two keys.

**Returns**

the longest common name found

#### 573.92.2.2 copyAllMeta()

```
void kdb::tools::helper::copyAllMeta (
            KeySet & to,
            KeySet const & from )
```

Copies all metadata of each key in from to their pendant in to.
The pendant is determined by ksLookup.

**Parameters**

| | |
|---|---|
| *to* | metadata is copied to keys in this key set |
| *from* | metadata is copied from keys in this key set |

#### 573.92.2.3 prependNamespace() [1/2]

```
Key kdb::tools::helper::prependNamespace (
            Key const & root,
            std::string const & ns )
```

Prepends the namespace to the key.

**Parameters**

| | |
|---|---|
| *root* | is the key where the namespace will be prepended |
| *ns* | is the namespace to be prepended |

**Returns**

a duplicate of root with the new namespace

#### 573.92.2.4 prependNamespace() [2/2]

```
KeySet kdb::tools::helper::prependNamespace (
            KeySet const & resultKeys,
            std::string const & ns )
```

Prepends the namespace to each key of the key set.

**Parameters**

| | |
|---|---|
| *resultKeys* | which keys to prepend the namepsace to |
| *ns* | is the namespace to be prepended to each key |

**Returns**

a keyset where each key has the new namespace

#### 573.92.2.5 rebaseKey()

```
Key kdb::tools::helper::rebaseKey (
            const Key & key,
            const Key & oldParent,
            const Key & newParent )
```

Rebases the supplied key from the old parent to the new parent.

**See also**

ThreeWayMerge::rebasePath

**Parameters**

| | |
|---|---|
| *key* | the key to be rebased |
| *oldParent* | the old parent of the key |
| *newParent* | the new parent of the key |

**Returns**

a rebased copy of the supplied key

**Exceptions**

| | |
|---|---|
| *InvalidRebaseException* | if the key is not below the old parent |

#### 573.92.2.6 rebasePath()

```
string kdb::tools::helper::rebasePath (
            const Key & key,
            const Key & oldParent,
            const Key & newParent )
```

Rebases the relative path of the passed key from the old parent to the new parent and returns the new path.
For example a key /user/example/config/key1 with the oldparent /user/example and the new parent /user/newexample/newpath would result in /user/newexample/newpath/config/key1
If any of the parent keys is a cascading key the namespace of the key to be rebased is assumed instead.

**Parameters**

| | |
|---|---|
| *key* | the key whose path should be rebased |
| *oldParent* | the old parent of the key |
| *newParent* | the new parent of the key |

**Returns**

the rebased path

**Exceptions**

| | |
|---|---|
| *InvalidRebaseException* | if the key is not below the old parent |

#### 573.92.2.7 removeNamespace()

```
void kdb::tools::helper::removeNamespace (
            Key & key )
```

Removes the namespace.

**Parameters**

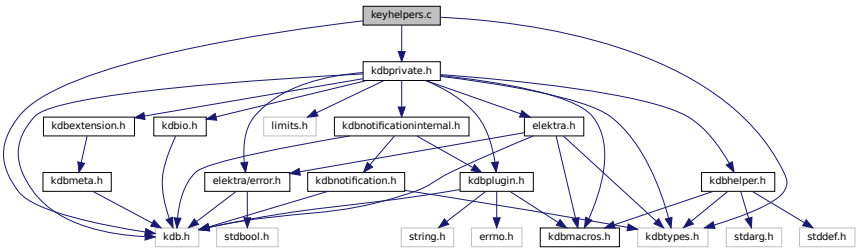
| | |
|---|---|
| *key* | will be made to a cascading key |

## 573.93 keyhelpers.c File Reference

Helpers for key manipulation.

```
#include "kdb.h"
#include "kdbprivate.h"
#include "kdbtypes.h"
```

Include dependency graph for keyhelpers.c:




### 573.93.1 Detailed Description

Helpers for key manipulation.

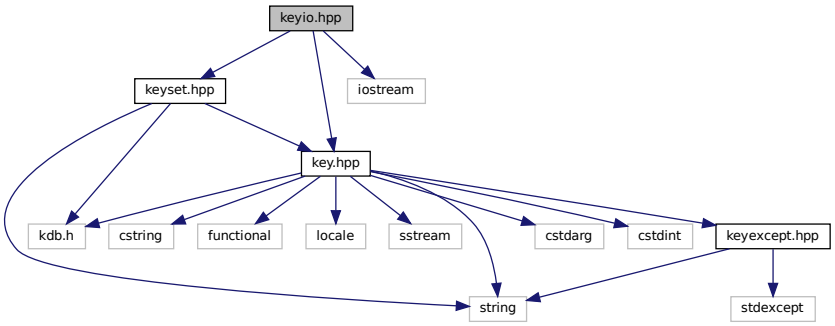
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

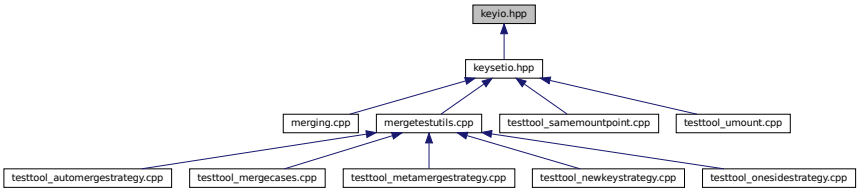
## 573.94 keyio.hpp File Reference

```
#include <key.hpp>
#include <keyset.hpp>
#include <iostream>
```

Include dependency graph for keyio.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

*This is the main namespace for the C++ binding and libraries.*

### Functions

- std::ostream & kdb::operator<< (std::ostream &os, kdb::Key const &k)

  *Stream the name of a key.*
- std::istream & kdb::operator>> (std::istream &is, kdb::Key &k)

  *Reads a line with a keys name.*

#### 573.94.1 Detailed Description

**Copyright**

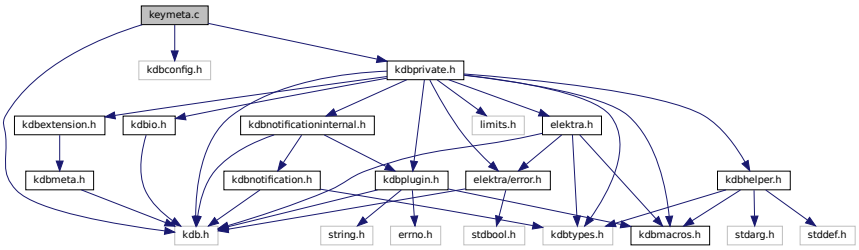
BSD License (see LICENSE.md or `https://www.libelektra.org`)

## 573.95 keymeta.c File Reference

Methods to do various operations on Key metadata.

```
#include <kdb.h>
#include <kdbconfig.h>
#include <kdbprivate.h>
```

Include dependency graph for keymeta.c:




### Functions

- const Key * keyNextMeta (Key *key)

  *Get the next metadata entry of a Key.*
- int keyCopyMeta (Key *dest, const Key *source, const char *metaName)

  *Do a shallow copy of metadata with name `metaName` from source to dest.*
- int keyCopyAllMeta (Key *dest, const Key *source)

  *Do a shallow copy of all metadata from source to dest.*
- const Key * keyGetMeta (const Key *key, const char *metaName)

  *Returns the Key for a metadata entry with name `metaName`.*
- ssize_t keySetMeta (Key *key, const char *metaName, const char *newMetaString)

  *Set a new metadata Key.*
- KeySet * keyMeta (Key *key)

  *Returns the KeySet holding the given Key's metadata.*

#### 573.95.1 Detailed Description

Methods to do various operations on Key metadata.

**Copyright**

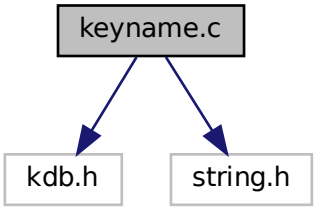
BSD License (see LICENSE.md or `https://www.libelektra.org`)

# 573.96 keyname.c File Reference

Methods for accessing key names.

```
#include <kdb.h>
#include <string.h>
```

Include dependency graph for ease/keyname.c:




## Functions

- const char * elektraKeyGetRelativeName (Key const *cur, Key const *parentKey)

  *get relative position of key based on parentKey*

## 573.96.1 Detailed Description

Methods for accessing key names.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.96.2 Function Documentation

### 573.96.2.1 elektraKeyGetRelativeName()

```
const char* elektraKeyGetRelativeName (
            Key const * cur,
            Key const * parentKey )
```

get relative position of key based on parentKey

**Precondition**

parentKey is either the same key as cur, or one of its parents

**Postcondition**

a pointer to the relevant part of the parent key's name, the full name if there is no relation to the parentKey

If the parentKey does not fulfill the precondition, the result won't be the correct relative key of cur.

**Parameters**

| | |
|---|---|
| *cur* | the key below parentKey we want to get the relative basename of |
| *parentKey* | the key that defines the root/base |

**Returns**

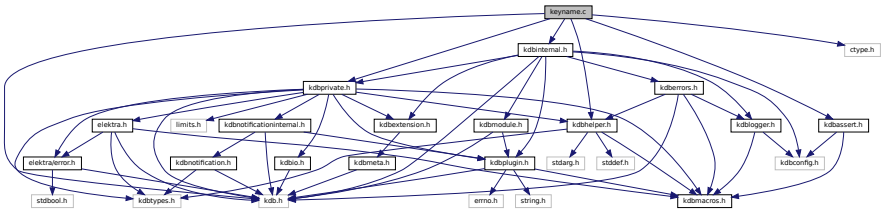
a pointer to the relative part name of the key cur

## 573.97 keyname.c File Reference

Methods for Key name manipulation.

```
#include "kdbprivate.h"
#include <kdbassert.h>
#include <ctype.h>
#include "kdb.h"
#include "kdbhelper.h"
#include "kdbinternal.h"
```

Include dependency graph for elektra/keyname.c:




### Functions

- const char * keyName (const Key *key)

  *Returns a pointer to the abbreviated real internal `key` name.*
- ssize_t keyGetNameSize (const Key *key)

  *Bytes needed to store the Key's name (excluding owner).*
- const void * keyUnescapedName (const Key *key)

  *Returns a Key's name, separated by NULL bytes and without backslashes for escaping.*
- ssize_t keyGetUnescapedNameSize (const Key *key)

  *Returns the size of the Key's unescaped name including embedded and terminating NULL characters.*
- ssize_t keyGetName (const Key *key, char *returnedName, size_t maxSize)

  *Get abbreviated Key name (excluding owner).*
- ssize_t keyGetUnescapedName (const Key *key, char *returnedName, size_t maxSize)

  *Copies the unescaped name of a Key into `returnedName`.*
- ssize_t keySetName (Key *key, const char *newName)

  *Set a new name to a Key.*
- ssize_t keyAddName (Key *key, const char *newName)

  *Add an already escaped name part to the Key's name.*
- int keyReplacePrefix (Key *key, const Key *oldPrefix, const Key *newPrefix)

  *Replaces a prefix of the key name of `key`.*
- bool elektraKeyNameValidate (const char *name, bool isComplete)

  *Takes an escaped key name and validates it.*
- void elektraKeyNameCanonicalize (const char *name, char **canonicalName, size_t *canonicalSizePtr, size_t offset, size_t *usizePtr)

  *Takes a valid (non-)canonical key name and produces its canonical form.*
- void elektraKeyNameUnescape (const char *canonicalName, char *unescapedName)

  *Takes a canonical key name and unescapes it.*
- const char * keyBaseName (const Key *key)

  *Returns a pointer to the unescaped Key's name where the basename starts.*

- ssize_t keyGetBaseNameSize (const Key ∗key)

  *Calculates number of bytes needed to store basename of `key` (including NULL terminator).*
- ssize_t keyGetBaseName (const Key ∗key, char ∗returned, size_t maxSize)

  *Copy the Key's basename to `returned`.*
- size_t elektraKeyNameEscapePart (const char ∗part, char ∗∗escapedPart)

  *Takes a single key name part and produces its escaped form.*
- ssize_t keyAddBaseName (Key ∗key, const char ∗baseName)

  *Adds `baseName` to the name of `key`.*
- ssize_t keySetBaseName (Key ∗key, const char ∗baseName)

  *Sets `baseName` as the new basename for `key`.*
- elektraNamespace keyGetNamespace (const Key ∗key)

  *Returns the elektraNamespace for a Key.*
- ssize_t keySetNamespace (Key ∗key, elektraNamespace ns)

  *Changes the namespace of a Key.*
- int elektraIsArrayPart (const char ∗namePart)

  *Checks if the given key name part is an array part.*

### 573.97.1 Detailed Description

Methods for Key name manipulation.

**Copyright**

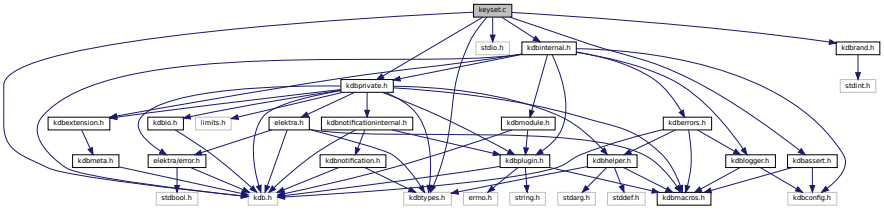
BSD License (see LICENSE.md or `https://www.libelektra.org`)

## 573.98 keyset.c File Reference

Methods for key sets.

```
#include "kdbprivate.h"
#include <kdb.h>
#include <stdio.h>
#include <kdbtypes.h>
#include "kdbinternal.h"
#include <kdbassert.h>
#include <kdbrand.h>
```

Include dependency graph for keyset.c:




## Functions

- KeySet ∗ ksNew (size_t alloc,...)

  *Allocate, initialize and return a new KeySet object.*
- KeySet ∗ ksVNew (size_t alloc, va_list va)

  *Allocate, initialize and return a new KeySet object.*
- KeySet ∗ ksDup (const KeySet ∗source)

  *Return a duplicate of a KeySet.*

- int ksCopy (KeySet *dest, const KeySet *source)

  *Replace the content of a KeySet with another one.*
- int ksDel (KeySet *ks)

  *A destructor for KeySet objects.*
- int ksClear (KeySet *ks)

  *Empties a KeySet.*
- uint16_t ksIncRef (KeySet *ks)

  *Increment the reference counter of a KeySet object.*
- uint16_t ksDecRef (KeySet *ks)

  *Decrement the reference counter of a KeySet object.*
- uint16_t ksGetRef (const KeySet *ks)

  *Return the current reference counter value of a KeySet object.*
- int keyCmp (const Key *k1, const Key *k2)

  *Compare the name of two Keys.*
- ssize_t ksGetSize (const KeySet *ks)

  *Return the number of Keys that `ks` contains.*
- ssize_t ksSearch (const KeySet *ks, const Key *key)

  *Search in a key set, either yielding the actual index of the key, if the key has been found within the key set, or a negative value indicating the insertion index of the key, if the key would be inserted.*
- ssize_t ksAppendKey (KeySet *ks, Key *toAppend)

  *Appends a Key to the end of `ks`.*
- ssize_t ksAppend (KeySet *ks, const KeySet *toAppend)

  *Append all Keys in `toAppend` to the end of the KeySet `ks`.*
- ssize_t ksRename (KeySet *ks, const Key *root, const Key *newRoot)

  *Moves all keys below `root` to below `newRoot`.*
- elektraCursor ksFindHierarchy (const KeySet *ks, const Key *root, elektraCursor *end)

  *Searches for the start and optionally end of the key hierarchy rooted at `root` in `ks`.*
- KeySet * ksBelow (const KeySet *ks, const Key *root)

  *Retrieves all Keys from KeySet `ks` that are below or at `root`.*
- KeySet * ksCut (KeySet *ks, const Key *cutpoint)

  *Cuts out all Keys from KeySet `ks` that are below or at `cutpoint`.*
- Key * ksPop (KeySet *ks)

  *Remove and return the last Key of `ks`.*
- int ksRewind (KeySet *ks)

  *Rewinds the KeySet internal cursor.*
- Key * ksNext (KeySet *ks)

  *Returns the next Key in a KeySet.*
- Key * ksCurrent (const KeySet *ks)

  *Return the current Key.*
- elektraCursor ksGetCursor (const KeySet *ks)

  *Get the internal cursor of the KeySet.*
- Key * ksAtCursor (const KeySet *ks, elektraCursor pos)

  *Return Key at given position `pos`.*
- int ksSetCursor (KeySet *ks, elektraCursor cursor)

  *Set the KeySet internal cursor to `cursor`.*
- Key * ksLookup (KeySet *ks, Key *key, elektraLookupFlags options)

  *Look for a Key contained in `ks` that matches the name of the `key`.*
- Key * ksLookupByName (KeySet *ks, const char *name, elektraLookupFlags options)

  *Convenience method to look for a Key contained in `ks` with name `name`.*
- ssize_t ksSubtract (KeySet *total, const KeySet *sub)

  *Remove all the keys in `sub` from `total`.*

### 573.98.1 Detailed Description

Methods for key sets.

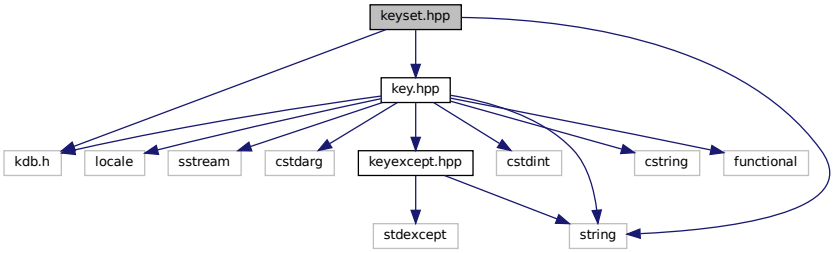
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

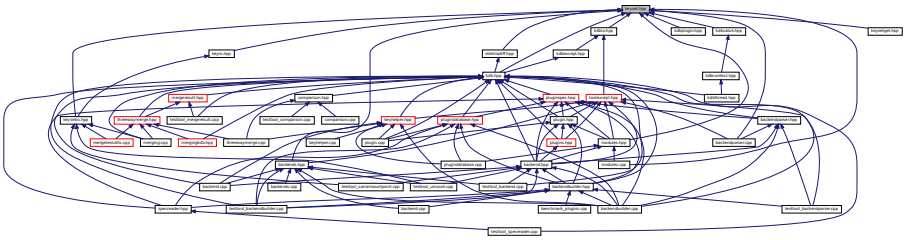
## 573.99 keyset.hpp File Reference

```
#include <kdb.h>
#include <key.hpp>
#include <string>
```

Include dependency graph for keyset.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- struct kdb::VaAlloc

  *Needed to avoid constructor ambiguity.*
- class kdb::KeySet

  *A keyset holds together a set of keys.*
- class kdb::KeySetIterator

  *For C++ forward Iteration over KeySets.*
- class kdb::KeySetReverseIterator

  *For C++ reverse Iteration over KeySets.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.99.1 Detailed Description

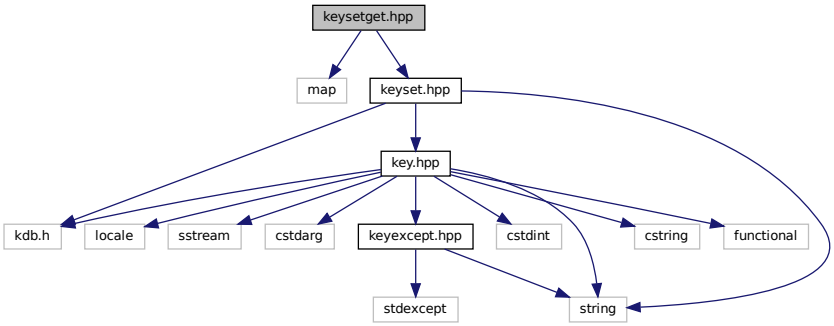
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.100 keysetget.hpp File Reference

```
#include <map>
#include <keyset.hpp>
```

Include dependency graph for keysetget.hpp:




### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

### 573.100.1 Detailed Description

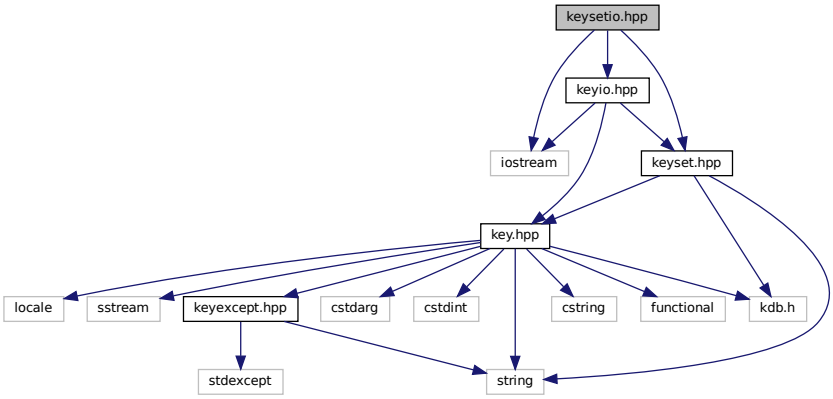
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

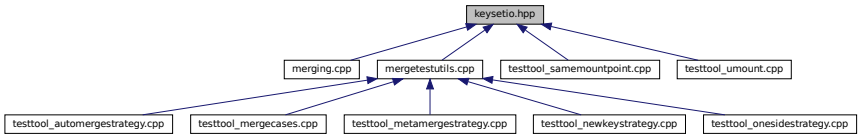
# 573.101 keysetio.hpp File Reference

```
#include <iostream>
#include <keyio.hpp>
#include <keyset.hpp>
```

Include dependency graph for keysetio.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

## Functions

- std::ostream & kdb::operator<< (std::ostream &os, kdb::KeySet const &cks)

  *Outputs line per line the keynames.*
- std::istream & kdb::operator>> (std::istream &is, kdb::KeySet &ks)

  *Reads line per line key names and appends those keys to ks.*

### 573.101.1 Detailed Description

**Copyright**

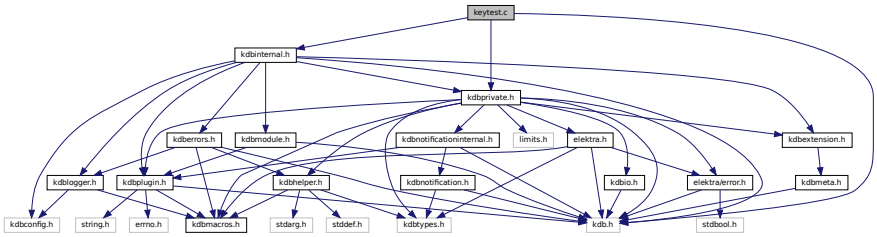
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.102 keytest.c File Reference

Methods for making tests.

```
#include "kdb.h"
#include "kdbinternal.h"
#include "kdbprivate.h"
```

Include dependency graph for keytest.c:




## Functions

- int keyIsBelow (const Key ∗key, const Key ∗check)

  *Check if the Key `check` is below the Key `key` or not.*
- int keyIsBelowOrSame (const Key ∗key, const Key ∗check)

  *Check if a key is below or same.*
- int keyIsDirectlyBelow (const Key ∗key, const Key ∗check)

  *Check whether the Key `check` is directly below the Key `key`.*
- int keyIsBinary (const Key ∗key)

  *Check if the value of a `key` is of binary type.*
- int keyIsString (const Key ∗key)

  *Check if the value of `key` is of string type.*

## 573.102.1 Detailed Description

Methods for making tests.

### Copyright

BSD License (see LICENSE.md or `https://www.libelektra.org`)

## 573.102.2 Function Documentation

### 573.102.2.1 keyIsBelowOrSame()

```
int keyIsBelowOrSame (
            const Key * key,
            const Key * check )
```

Check if a key is below or same.

### Parameters

| | |
|---|---|
| *key* | the key object to work with |

**See also**

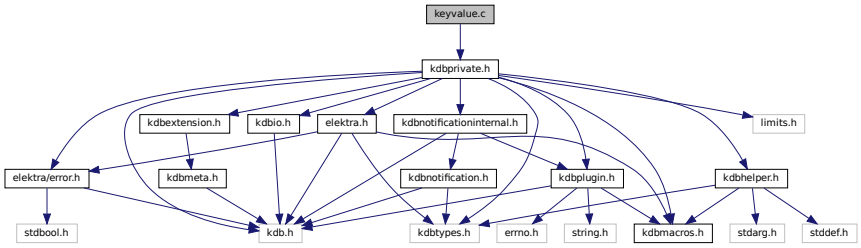
keyIsBelow()

# 573.103 keyvalue.c File Reference

Methods for Key value manipulation.

```
#include "kdbprivate.h"
```

Include dependency graph for keyvalue.c:




## Functions

- const void * keyValue (const Key *key)

  *Return a pointer to the real internal `key` value.*
- const char * keyString (const Key *key)

  *Get a pointer to the c-string representing the value.*
- ssize_t keyGetValueSize (const Key *key)

  *Returns the number of bytes needed to store the key value, including the NULL terminator.*
- ssize_t keyGetString (const Key *key, char *returnedString, size_t maxSize)

  *Copy the string value of a Key into `returnedString`.*
- ssize_t keySetString (Key *key, const char *newStringValue)

  *Set the value for `key` as `newStringValue`.*
- ssize_t keyGetBinary (const Key *key, void *returnedBinary, size_t maxSize)

  *Copy the binary value of a Key into `returnedBinary`.*
- ssize_t keySetBinary (Key *key, const void *newBinary, size_t dataSize)

  *Set the value of a Key to the binary value `newBinary`.*

### 573.103.1 Detailed Description

Methods for Key value manipulation.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

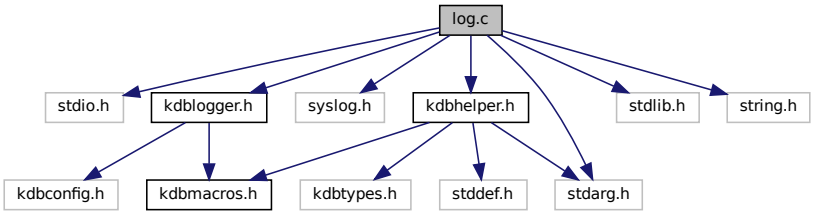
# 573.104 log.c File Reference

Non-C99 Logger Implementation.

```
#include <stdio.h>
#include <kdblogger.h>
#include <syslog.h>
#include <kdbhelper.h>
#include <stdarg.h>
#include <stdlib.h>
```

```
#include <string.h>
```

Include dependency graph for log.c:




### 573.104.1 Detailed Description

Non-C99 Logger Implementation.
If you often change the file, you might want to set CMAKE_LINK_DEPENDS_NO_SHARED to avoid relinking everything.
Do not commit changes do this file, except if you want to change the default. In that case, make sure to update also doc/tutorials/logger.md

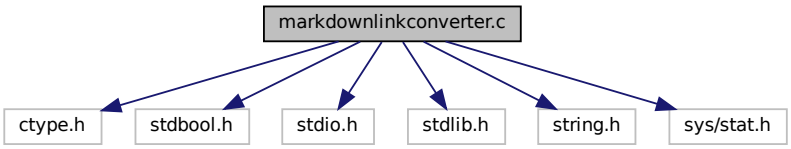
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.105 markdownlinkconverter.c File Reference

```
#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
```

Include dependency graph for markdownlinkconverter.c:




### 573.105.1 Detailed Description

**Copyright**

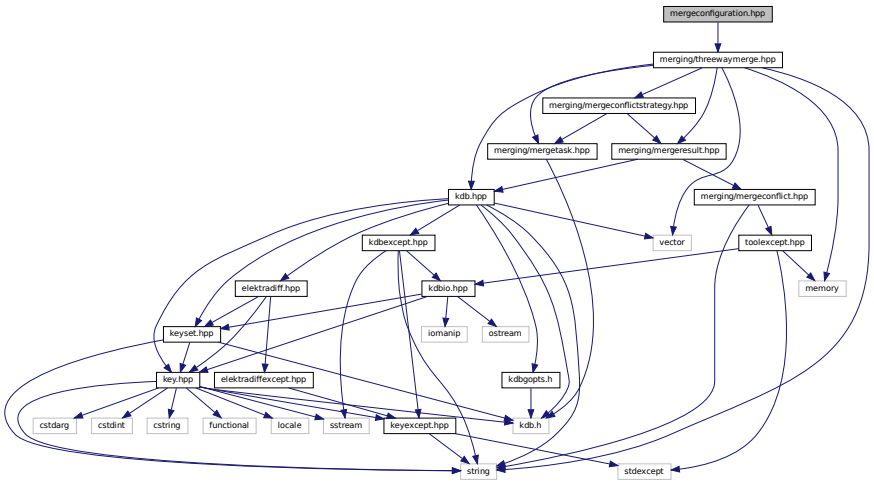
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.106 mergeconfiguration.hpp File Reference

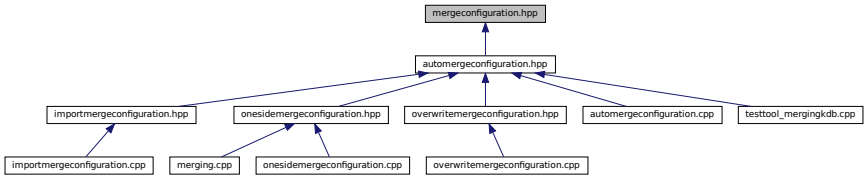
Base class for defining preconfigured merge configurations.

```
#include <merging/threewaymerge.hpp>
```

Include dependency graph for mergeconfiguration.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.106.1 Detailed Description

Base class for defining preconfigured merge configurations.

**Copyright**

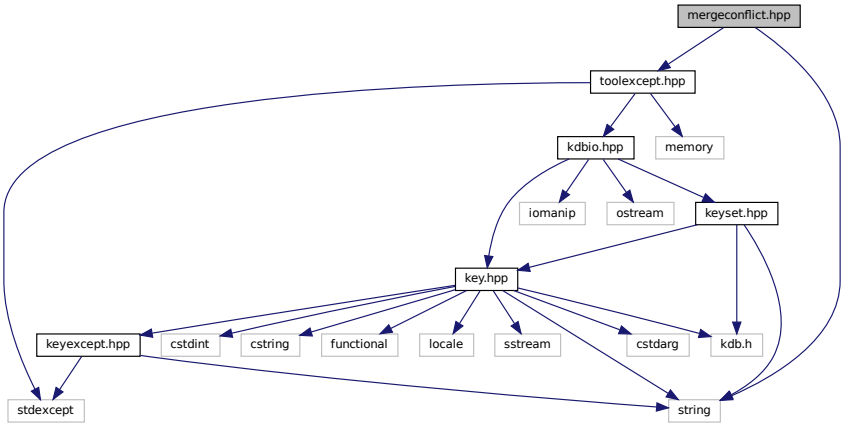
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.107 mergeconflict.hpp File Reference

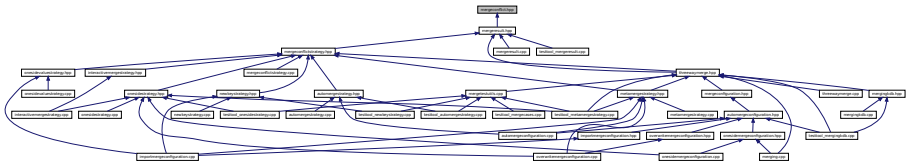
Models a merge conflict.

```
#include <string>
#include <toolexcept.hpp>
```

Include dependency graph for mergeconflict.hpp:




This graph shows which files directly or indirectly include this file:



## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Enumerations

- enum kdb::tools::merging::ConflictOperation { kdb::tools::merging::CONFLICT_ADD , **CONFLICT_DELETE** , **CONFLICT_MODIFY** , **CONFLICT_META** , **CONFLICT_SAME** }

### 573.107.1 Detailed Description

Models a merge conflict.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.107.2 Enumeration Type Documentation

#### 573.107.2.1 ConflictOperation

enum kdb::tools::merging::ConflictOperation

**Enumerator**

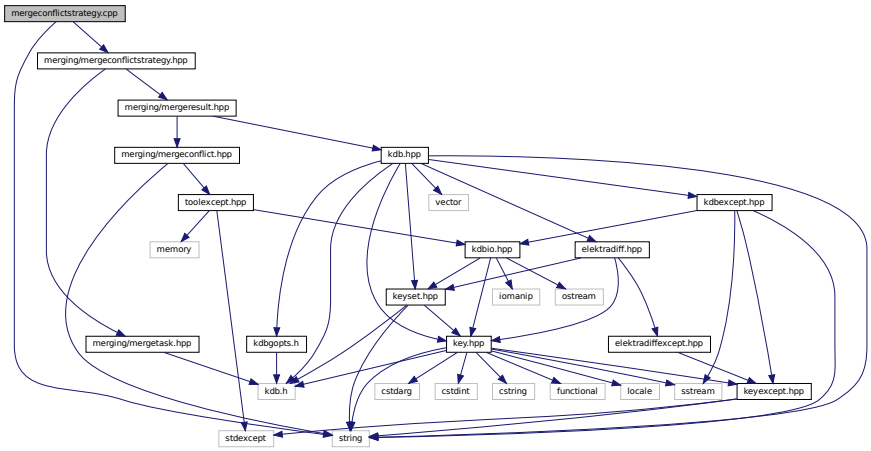
| | |
|---|---|
| CONFLICT_ADD | Conflict because of adding key. |

# 573.108 mergeconflictstrategy.cpp File Reference

Implementation of MergeConflictStrategy.

```
#include <merging/mergeconflictstrategy.hpp>
#include <string>
```

Include dependency graph for mergeconflictstrategy.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

### 573.108.1 Detailed Description

Implementation of MergeConflictStrategy.

**Copyright**

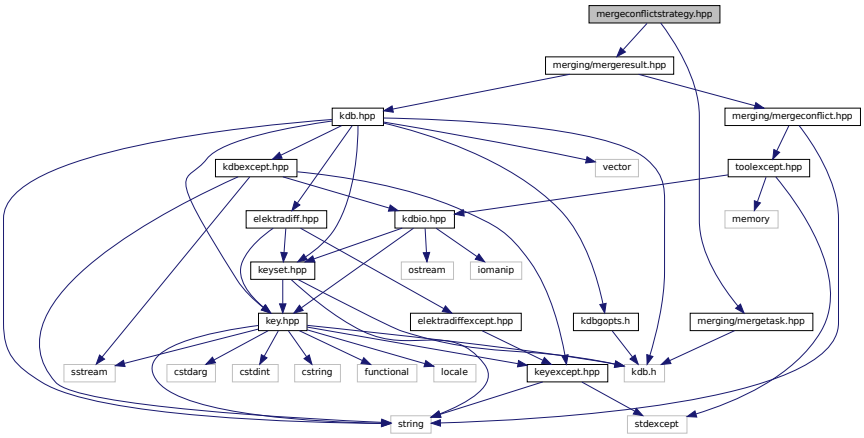
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.109 mergeconflictstrategy.hpp File Reference

Interface for a MergeConflictStrategy.

```
#include <merging/mergeresult.hpp>
#include <merging/mergetask.hpp>
```

Include dependency graph for mergeconflictstrategy.hpp:




This graph shows which files directly or indirectly include this file:

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## 573.109.1 Detailed Description

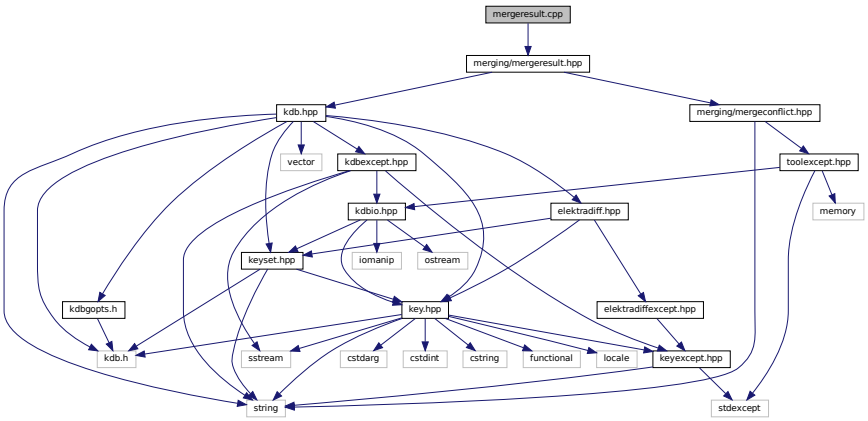
Interface for a MergeConflictStrategy.

# 573.110 mergeresult.cpp File Reference

Implementation of MergeResult.

`#include <merging/mergeresult.hpp>`

Include dependency graph for mergeresult.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

## 573.110.1 Detailed Description

Implementation of MergeResult.

**Copyright**

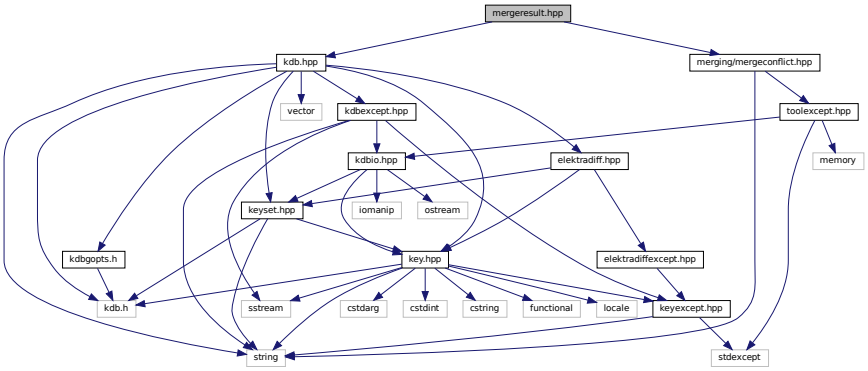
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.111 mergeresult.hpp File Reference

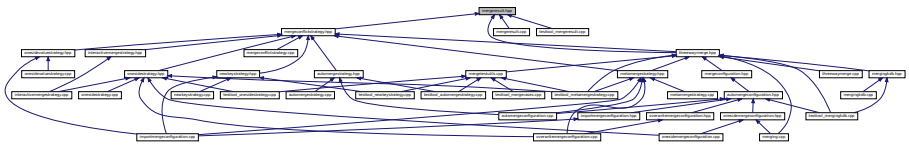
Class modelling the result of a three way merge.

```
#include <kdb.hpp>
#include <merging/mergeconflict.hpp>
```

Include dependency graph for mergeresult.hpp:




This graph shows which files directly or indirectly include this file:



### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.111.1 Detailed Description

Class modelling the result of a three way merge.

**Copyright**

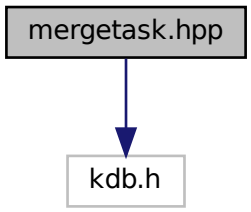
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.112 mergetask.hpp File Reference

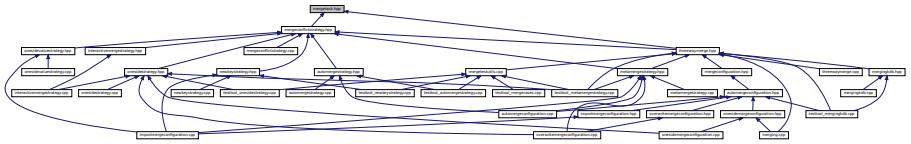
Models a merge task.

```
#include <kdb.h>
```

Include dependency graph for mergetask.hpp:




This graph shows which files directly or indirectly include this file:



### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.112.1 Detailed Description

Models a merge task.

**Copyright**

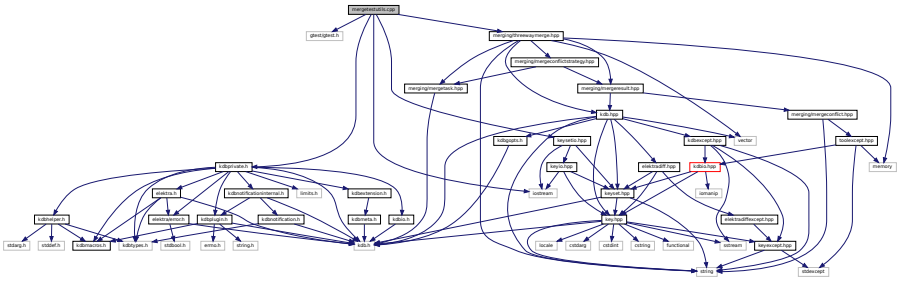
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.113 mergetestutils.cpp File Reference

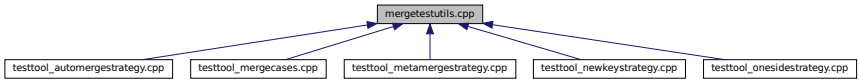
Implements a helper class for merge related tests.

```
#include <gtest/gtest.h>
#include <iostream>
#include <kdbprivate.h>
#include <keysetio.hpp>
#include <merging/threewaymerge.hpp>
```

Include dependency graph for mergetestutils.cpp:



This graph shows which files directly or indirectly include this file:




### 573.113.1 Detailed Description

Implements a helper class for merge related tests.

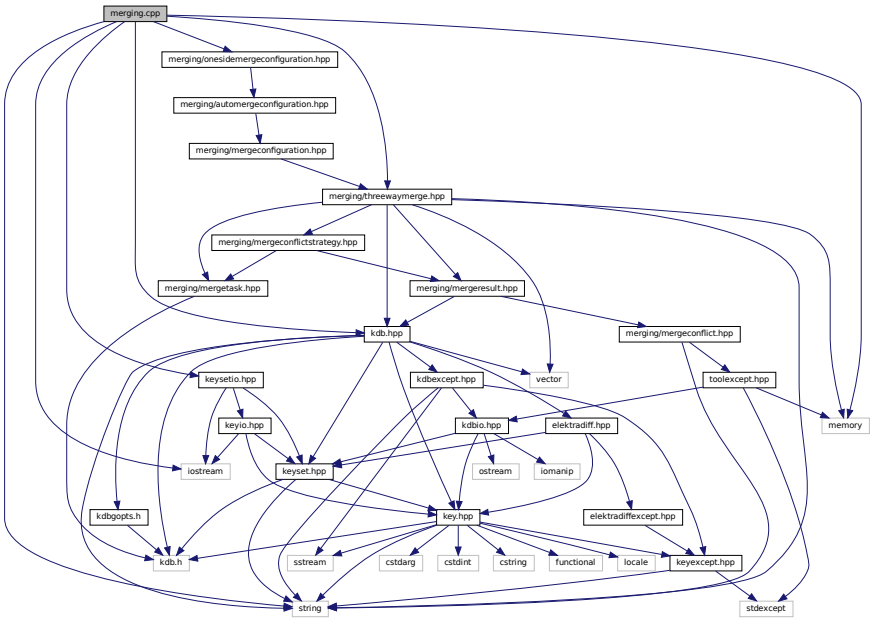
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.114 merging.cpp File Reference

```
#include <kdb.hpp>
#include <keysetio.hpp>
#include <iostream>
#include <memory>
#include <string>
#include <merging/onesidemergeconfiguration.hpp>
#include <merging/threewaymerge.hpp>
```

Include dependency graph for merging.cpp:




### 573.114.1 Detailed Description

**Copyright**

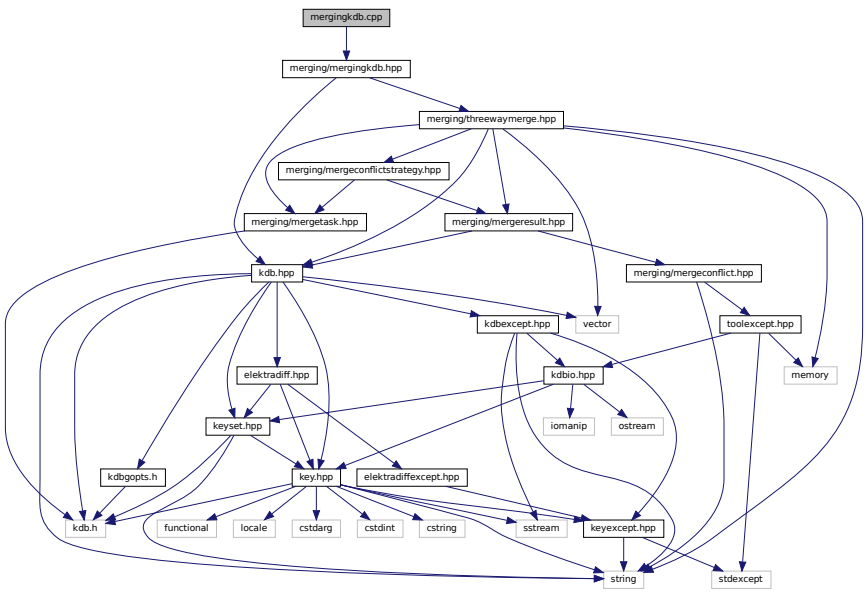
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.115 mergingkdb.cpp File Reference

Implementation of MergeResult.

`#include <merging/mergingkdb.hpp>`

Include dependency graph for mergingkdb.cpp:



### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.115.1 Detailed Description

Implementation of MergeResult.

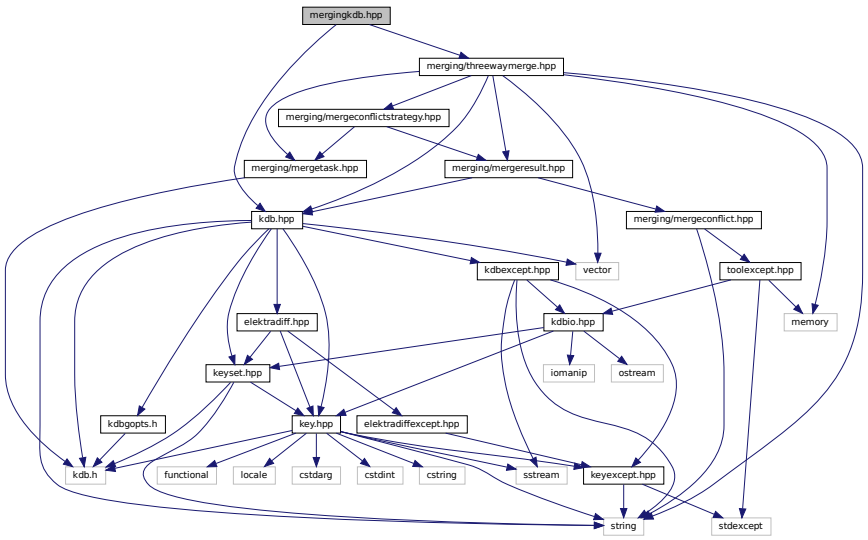
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

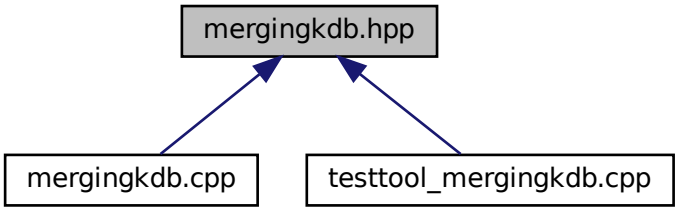
## 573.116 mergingkdb.hpp File Reference

```
#include <kdb.hpp>
#include <merging/threewaymerge.hpp>
```

Include dependency graph for mergingkdb.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- class kdb::tools::merging::MergingKDB

  *Provides a merging wrapper around a KDB instance.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.116.1 Detailed Description

**Copyright**

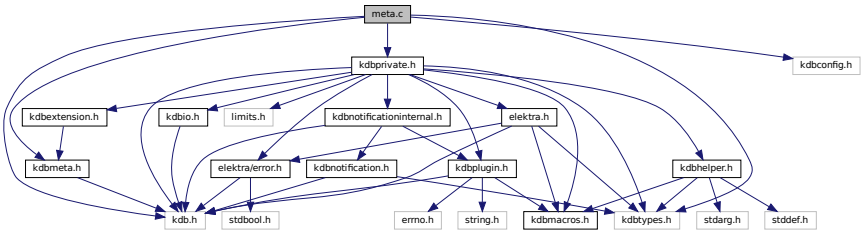
BSD License (see LICENSE.md or `https://www.libelektra.org`)

# 573.117 meta.c File Reference

Methods for metadata manipulation.

```
#include <kdb.h>
#include <kdbconfig.h>
#include <kdbease.h>
#include <kdbmeta.h>
#include <kdbprivate.h>
#include <kdbtypes.h>
```

Include dependency graph for meta.c:




## Functions

- const char * keyComment (const Key *key)

  *Return a pointer to the real internal `key` comment.*
- ssize_t keyGetCommentSize (const Key *key)

  *Calculates number of bytes needed to store a key comment, including final NULL.*
- ssize_t keyGetComment (const Key *key, char *returnedComment, size_t maxSize)

  *Get the key comment.*
- ssize_t keySetComment (Key *key, const char *newComment)

  *Set a comment for a key.*
- int elektraKeyCmpOrder (const Key *ka, const Key *kb)

  *Compare the order metadata of two keys.*
- void elektraMetaArrayAdd (Key *key, const char *metaName, const char *value)

  *creates an metadata array or appends another element to an existing metadata array e.g.*
- KeySet * elektraMetaArrayToKS (Key *key, const char *metaName)

*Create a `KeySet` from a metakey array.*

- int elektraSortTopology (KeySet *ks, Key **array)

  *topological sorting*

- char * elektraMetaArrayToString (const Key *key, const char *metaName, const char *delim)

  *returns the metakey array as a string separated by delim*

### 573.117.1 Detailed Description

Methods for metadata manipulation.

**Copyright**

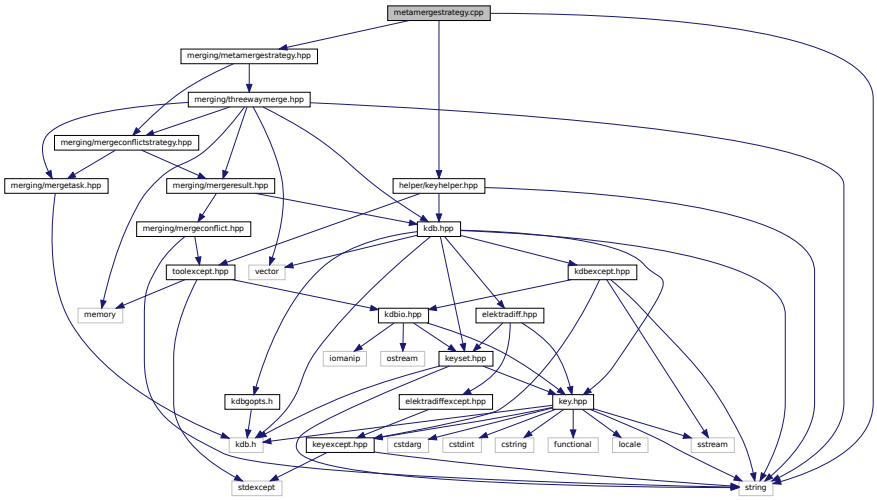
BSD License (see LICENSE.md or `https://www.libelektra.org`)

## 573.118 metamergestrategy.cpp File Reference

Implementation of MetaMergeStrategy.

```
#include <helper/keyhelper.hpp>
#include <merging/metamergestrategy.hpp>
#include <string>
```

Include dependency graph for metamergestrategy.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

### 573.118.1 Detailed Description

Implementation of MetaMergeStrategy.

**Copyright**

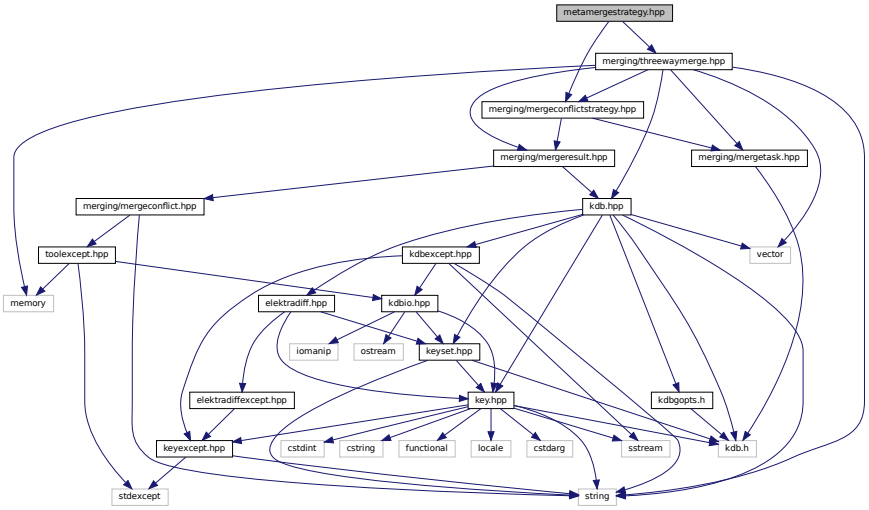
BSD License (see LICENSE.md or `https://www.libelektra.org`)

# 573.119 metamergestrategy.hpp File Reference

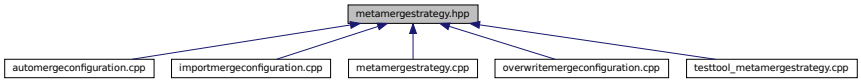
Applies a MergeConflictStrategy on the metakeys.

```
#include <merging/mergeconflictstrategy.hpp>
#include <merging/threewaymerge.hpp>
```

Include dependency graph for metamergestrategy.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.119.1 Detailed Description

Applies a MergeConflictStrategy on the metakeys.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

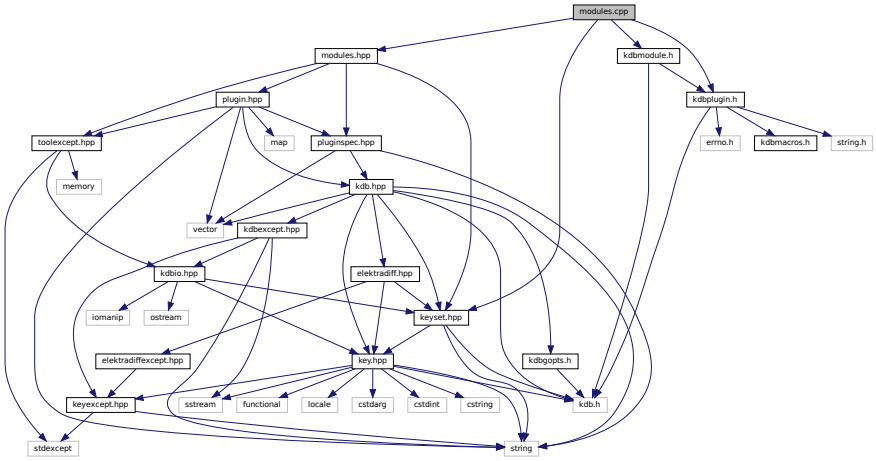
# 573.120 modules.cpp File Reference

Implementation of module loading.

```
#include <keyset.hpp>
#include <modules.hpp>
#include <kdbmodule.h>
```

```
#include <kdbplugin.h>
```

Include dependency graph for modules.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.120.1 Detailed Description

Implementation of module loading.

**Copyright**

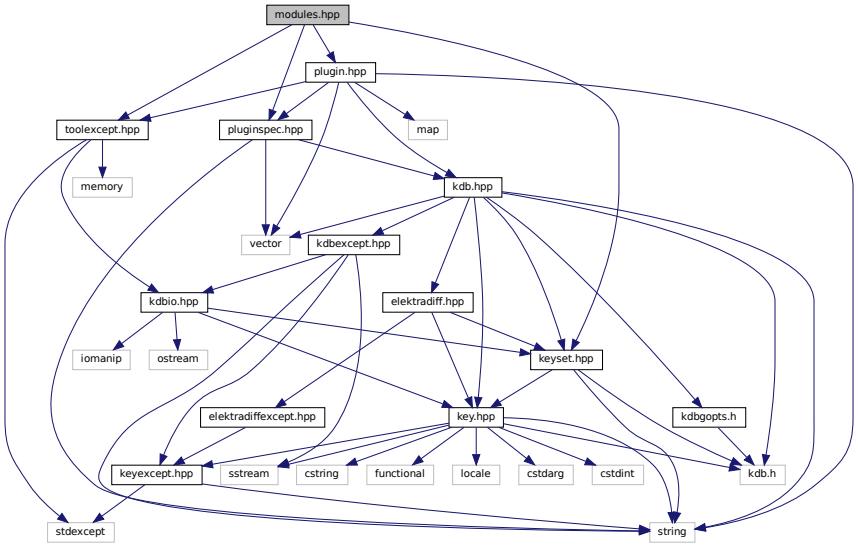
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.121 modules.hpp File Reference

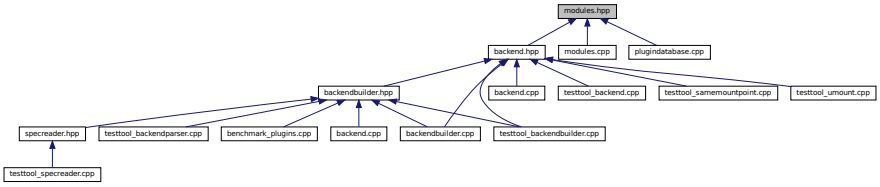
Allows one to load plugins.

```
#include <keyset.hpp>
#include <plugin.hpp>
#include <pluginspec.hpp>
#include <toolexcept.hpp>
```

Include dependency graph for modules.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::tools::Modules

  *Allows one to load plugins.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.121.1 Detailed Description

Allows one to load plugins.

**Copyright**

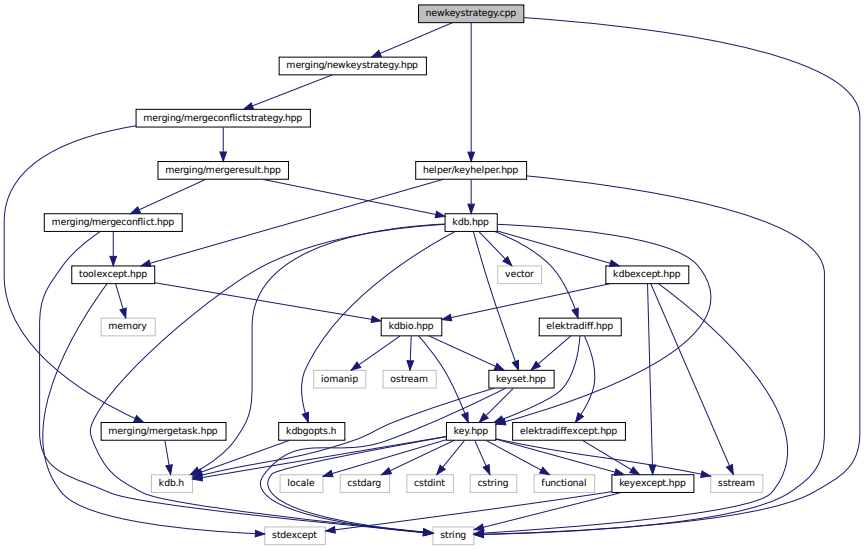
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.122 newkeystrategy.cpp File Reference

Implementation of OneSideStrategy.

```
#include <helper/keyhelper.hpp>
#include <merging/newkeystrategy.hpp>
#include <string>
```

Include dependency graph for newkeystrategy.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

### 573.122.1 Detailed Description

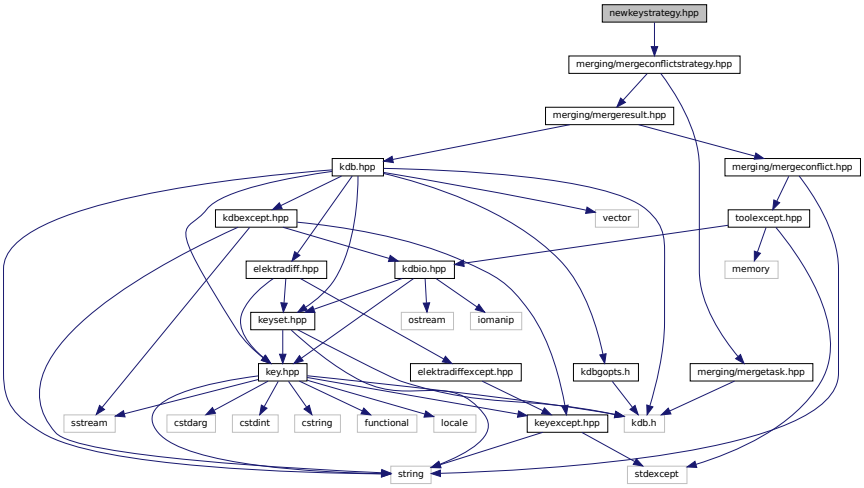
Implementation of OneSideStrategy.

**Copyright**

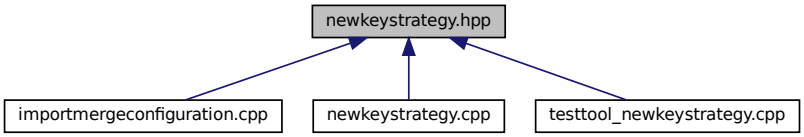
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.123 newkeystrategy.hpp File Reference

A strategy which always takes the value from one side.
`#include <merging/mergeconflictstrategy.hpp>`
Include dependency graph for newkeystrategy.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.123.1 Detailed Description

A strategy which always takes the value from one side.

**Copyright**

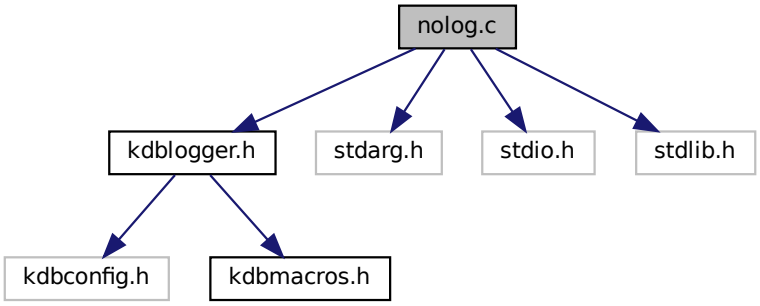
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.124 nolog.c File Reference

C99-compatible Fake Logger Implementation.

```
#include <kdblogger.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for nolog.c:




### 573.124.1 Detailed Description

C99-compatible Fake Logger Implementation.

**Copyright**

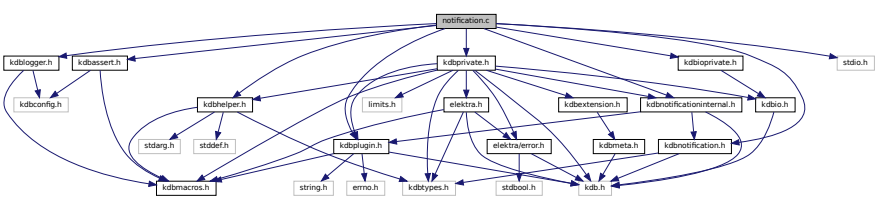
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.125 notification.c File Reference

Implementation of notification functions as defined in kdbnotification.h.

```
#include <kdbassert.h>
#include <kdbease.h>
#include <kdbhelper.h>
#include <kdbinvoke.h>
#include <kdbioprivate.h>
#include <kdblogger.h>
#include <kdbnotification.h>
#include <kdbnotificationinternal.h>
#include <kdbplugin.h>
#include <kdbprivate.h>
#include <stdio.h>
```

Include dependency graph for notification.c:



## Functions

- int elektraNotificationContract (KeySet ∗contract)

  *Creates a contract for use with kdbOpen() that sets up notifications.*
- int elektraNotificationRegisterCallback (KDB ∗kdb, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

  *Subscribe for updates via callback when a given key value is changed.*
- int elektraNotificationRegisterCallbackSameOrBelow (KDB ∗kdb, Key ∗key, ElektraNotificationChangeCallback callback, void ∗context)

  *Subscribe for updates via callback when a given key or a key below changed.*

### 573.125.1 Detailed Description

Implementation of notification functions as defined in kdbnotification.h.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.125.2 Function Documentation

#### 573.125.2.1 elektraNotificationContract()

```
int elektraNotificationContract (
            KeySet * contract )
```

Creates a contract for use with kdbOpen() that sets up notifications.
When you call kdbOpen() with this contract, the `internalnotification` plugin will be mounted automatically.
This allows you to call other `elektraNotification∗` functions.
If you need to configure notification transport plugins, you should manually add the relevant keys to `contract`.

**Parameters**

| | |
|---|---|
| *contract* | The keyset into which the contract is written. |

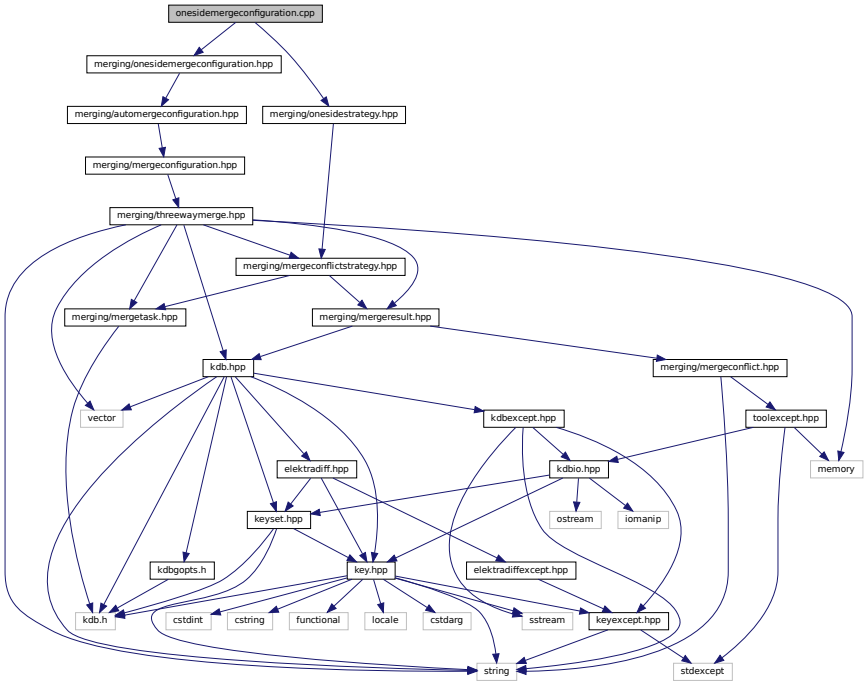
**Return values**

| | |
|---|---|
| *-1* | if `contract` is NULL |
| *0* | on success |

## 573.126 onesidemergeconfiguration.cpp File Reference

```
#include <merging/onesidemergeconfiguration.hpp>
#include <merging/onesidestrategy.hpp>
```

Include dependency graph for onesidemergeconfiguration.cpp:




## Namespaces

- kdb

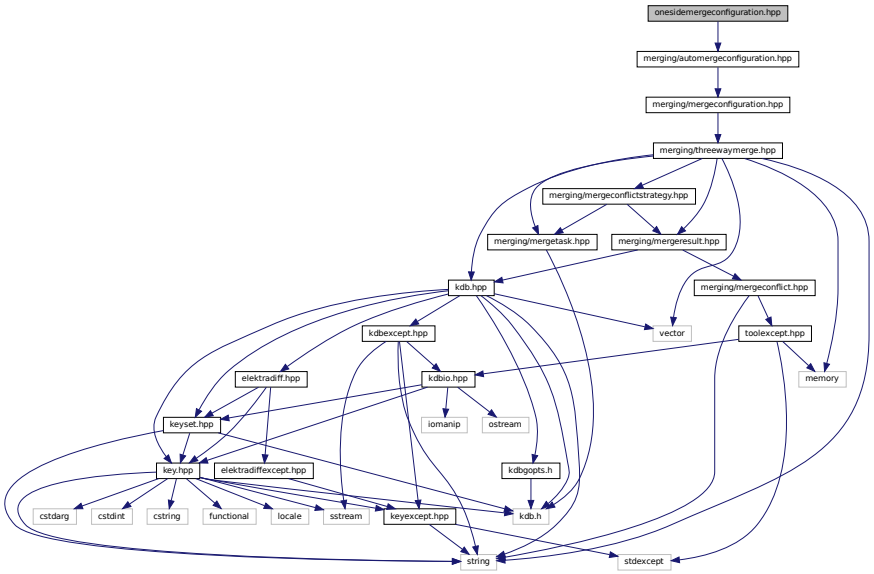
  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

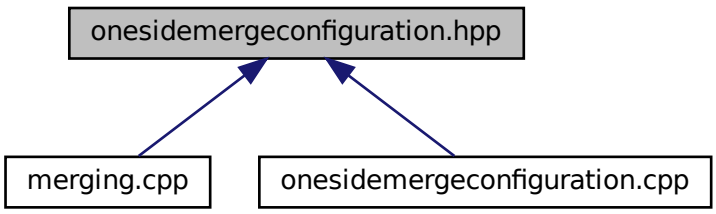
## 573.126.1 Detailed Description

# 573.127 onesidemergeconfiguration.hpp File Reference

A configuration for a simple automerge and guaranteed conflict resolution by one side.
`#include <merging/automergeconfiguration.hpp>`
Include dependency graph for onesidemergeconfiguration.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.127.1 Detailed Description

A configuration for a simple automerge and guaranteed conflict resolution by one side.

**Copyright**

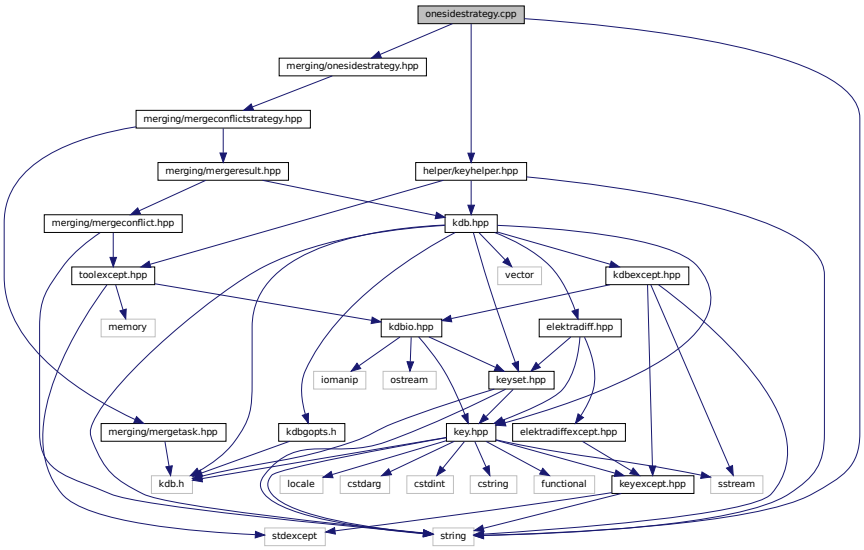
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.128 onesidestrategy.cpp File Reference

Implementation of OneSideStrategy.

```
#include <helper/keyhelper.hpp>
#include <merging/onesidestrategy.hpp>
#include <string>
```

Include dependency graph for onesidestrategy.cpp:




### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.128.1 Detailed Description

Implementation of OneSideStrategy.

**Copyright**

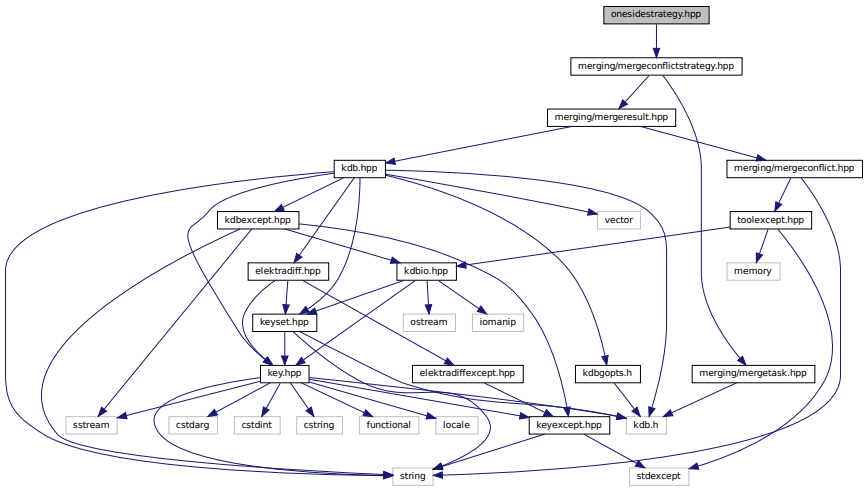
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.129 onesidestrategy.hpp File Reference

A strategy which always takes the value from one side.

```
#include <merging/mergeconflictstrategy.hpp>
```

Include dependency graph for onesidestrategy.hpp:




This graph shows which files directly or indirectly include this file:


## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.129.1 Detailed Description

A strategy which always takes the value from one side.

**Copyright**

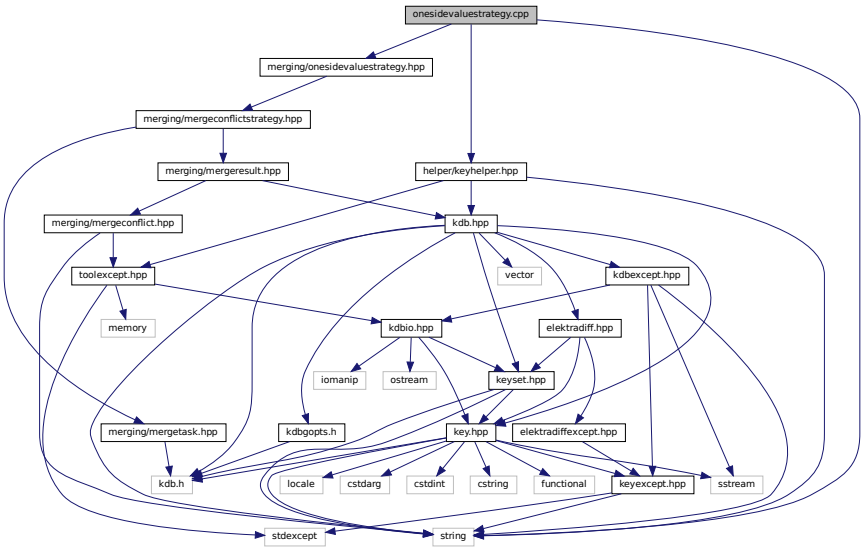
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.130 onesidevaluestrategy.cpp File Reference

Implementation of OneSideStrategy.

```
#include <helper/keyhelper.hpp>
#include <merging/onesidevaluestrategy.hpp>
#include <string>
```

Include dependency graph for onesidevaluestrategy.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.130.1 Detailed Description

Implementation of OneSideStrategy.

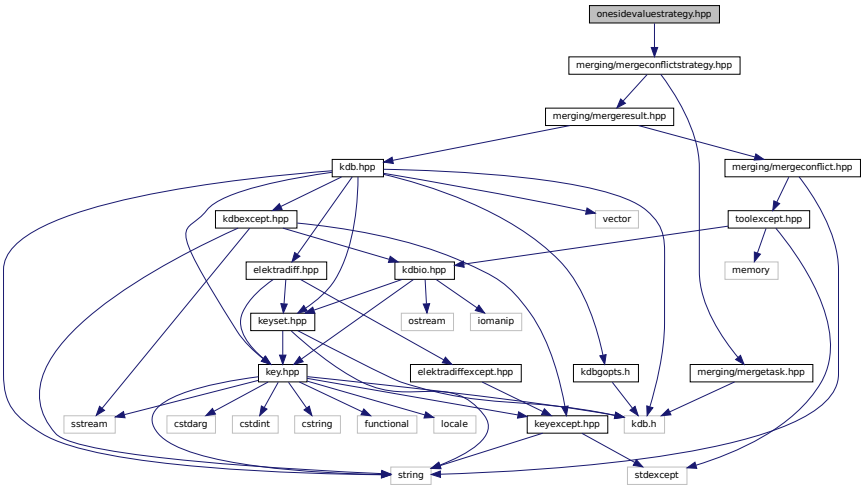
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

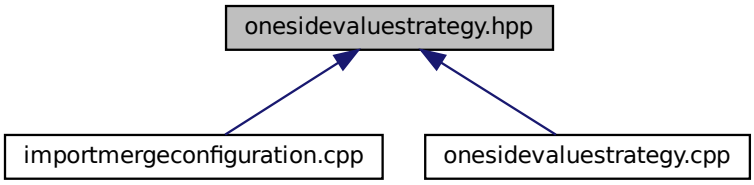
## 573.131 onesidevaluestrategy.hpp File Reference

```
#include <merging/mergeconflictstrategy.hpp>
```

Include dependency graph for onesidevaluestrategy.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.131.1 Detailed Description

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

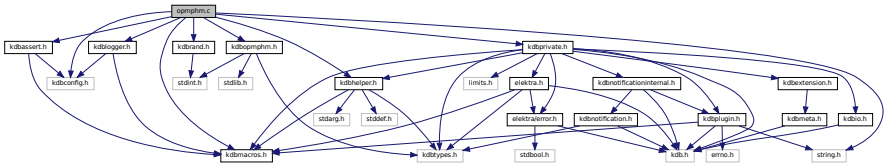
## 573.132 opmphm.c File Reference

The Order Preserving Minimal Perfect Hash Map.

```
#include <kdbassert.h>
#include <kdbconfig.h>
#include <kdbhelper.h>
#include <kdblogger.h>
#include <kdbmacros.h>
```

```
#include <kdbopmphm.h>
#include <kdbprivate.h>
#include <kdbrand.h>
#include <string.h>
```

Include dependency graph for opmphm.c:




## Functions

- size_t opmphmLookup (Opmphm *opmphm, size_t n, const void *name)

  *Looks up a element in the OPMPHM.*
- int opmphmAssignment (Opmphm *opmphm, OpmphmGraph *graph, size_t n, int defaultOrder)

  *Assigns the vertices of the r-uniform r-partite hypergraph.*
- int opmphmMapping (Opmphm *opmphm, OpmphmGraph *graph, OpmphmInit *init, size_t n)

  *Maps the elements to edges in the r-uniform r-partite hypergraph.*
- double opmphmMinC (uint8_t r)

  *Provides the minimal `c` value for a given `r`*
- uint8_t opmphmOptR (size_t n)

  *Provides the optimal `r` value for a given `n`*
- double opmphmOptC (size_t n)

  *Provides the optimal `c` value for a given `n`*
- OpmphmGraph * opmphmGraphNew (Opmphm *opmphm, uint8_t r, size_t n, double c)

  *Allocates and initializes the OpmphmGraph.*
- void opmphmGraphClear (const Opmphm *opmphm, OpmphmGraph *graph)

  *Clears the OpmphmGraph.*
- void opmphmGraphDel (OpmphmGraph *graph)

  *Deletes the OpmphmGraph.*
- Opmphm * opmphmNew (void)

  *Allocates and initializes the OPMPHM.*
- int opmphmCopy (Opmphm *dest, const Opmphm *source)

  *Copies OPMPHM from source to destination.*
- int opmphmIsBuild (const Opmphm *opmphm)

  *OPMPHM is build.*
- void opmphmDel (Opmphm *opmphm)

  *Deletes the OPMPHM.*
- void opmphmClear (Opmphm *opmphm)

  *Clears the OPMPHM.*
- ELEKTRA_NO_SANITIZE_UNDEFINED ELEKTRA_NO_SANITIZE_INTEGER ELEKTRA_NO_SANITIZE↩ _ADDRESS uint32_t opmphmHashfunction (const void *key, size_t length, uint32_t initval)

  *Hash function By Bob Jenkins, May 2006 http://burtleburtle.net/bob/c/lookup3.c Original name: hashlitte.*

## 573.132.1 Detailed Description

The Order Preserving Minimal Perfect Hash Map.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.132.2 Function Documentation

### 573.132.2.1 opmphmAssignment()

```
int opmphmAssignment (
            Opmphm * opmphm,
            OpmphmGraph * graph,
            size_t n,
            int defaultOrder )
```

Assigns the vertices of the r-uniform r-partite hypergraph.
Allocs the memory for the final OPMPHM `Opmphm->graph`. Uses the remove sequence `Opmphm->Graph->removeSequence`, generated during cycle check, to assign each vertex. Either with `Opmphm->Edge->order` or the default order, default is the order of `OpmphmInit->data`.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *graph* | the OpmphmGraph |
| *n* | the number of elements |
| *defaultOrder* | boolean flag |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on memory error |

### 573.132.2.2 opmphmClear()

```
void opmphmClear (
            Opmphm * opmphm )
```

Clears the OPMPHM.
The OPMPHM must be successfully created with `opmphmNew ()`. Clears and frees all internal memory of Opmphm, but not `Opmphm->hashFunctionSeeds` and the Opmphm instance.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

### 573.132.2.3 opmphmCopy()

```
int opmphmCopy (
            Opmphm * dest,
            const Opmphm * source )
```

Copies OPMPHM from source to destination.
Clears the dest and copies memory and values from source.

**Parameters**

| | |
|---|---|
| *dest* | the OPMPHM destination |
| *source* | the OPMPHM source |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | on memory error |

#### 573.132.2.4 opmphmDel()

```
void opmphmDel (
            Opmphm * opmphm )
```

Deletes the OPMPHM.
Clears and frees all memory in Opmphm.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

#### 573.132.2.5 opmphmGraphClear()

```
void opmphmGraphClear (
            const Opmphm * opmphm,
            OpmphmGraph * graph )
```

Clears the OpmphmGraph.
Sets all vertices to 0.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *graph* | the OpmphmGraph |

#### 573.132.2.6 opmphmGraphDel()

```
void opmphmGraphDel (
            OpmphmGraph * graph )
```

Deletes the OpmphmGraph.

**Parameters**

| | |
|---|---|
| *graph* | the OpmphmGraph |

### 573.132.2.7 opmphmGraphNew()

```
OpmphmGraph* opmphmGraphNew (
             Opmphm * opmphm,
             uint8_t r,
             size_t n,
             double c )
```

Allocates and initializes the OpmphmGraph.
Graph functions.
The OpmphmGraph represents a r-uniform r-partite hypergraph. Lazy initializes the `opmphm->hashFunctionSeeds` with r. Calculates also the size of one partition in the r-uniform r-partite hypergraph and stores it in `opmphm->componentSize`. Allocates all memory for the OpmphmGraph.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *r* | the rUniPar |
| *n* | the number of elements |
| *c* | space influencing parameter |

**Return values**

| | |
|---|---|
| *OpmphmGraph* | ∗ success |
| *NULL* | memory error |

### 573.132.2.8 opmphmIsBuild()

```
int opmphmIsBuild (
             const Opmphm * opmphm )
```

OPMPHM is build.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

**Return values**

| | |
|---|---|
| *0* | on false |
| *-1* | on true or NULL |

### 573.132.2.9 opmphmLookup()

```
size_t opmphmLookup (
             Opmphm * opmphm,
             size_t n,
             const void * name )
```

Looks up a element in the OPMPHM.
Lookup functions.

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |

**Parameters**

| | |
|---|---|
| *n* | the number of elements |
| *name* | the name of the element |

**Return values**

| | |
|---|---|
| *size_t* | the order of the element. |

### 573.132.2.10 opmphmMapping()

```
int opmphmMapping (
            Opmphm * opmphm,
            OpmphmGraph * graph,
            OpmphmInit * init,
            size_t n )
```

Maps the elements to edges in the r-uniform r-partite hypergraph.
Build functions.
Sets the seeds for the opmphmHashfunctions, `OpmphmInit->initSeed` will be changed. Inserts each element as edge in the r-uniform r-partite hypergraph and checks if the graph contains a cycle. If there are cycles the `graph` will be cleaned

**Parameters**

| | |
|---|---|
| *opmphm* | the OPMPHM |
| *graph* | the OpmphmGraph |
| *init* | the OpmphmInit |
| *n* | the number of elements |

**Return values**

| | |
|---|---|
| *0* | on success |
| *-1* | mapping not possible |

### 573.132.2.11 opmphmMinC()

```
double opmphmMinC (
            uint8_t r )
```

Provides the minimal `c` value for a given `r`
Functions providing `r` and `c`
This minimal values come from Fabiano Cupertino Botelho, Near-Optimal Space Perfect Hashing Algorithms, 2008.

**Parameters**

| | |
|---|---|
| *r* | the rUniPar |

**Return values**

| | |
|---|---|
| *c* | the minimal c value |

#### 573.132.2.12 opmphmNew()

```
Opmphm* opmphmNew (
            void  )
```

Allocates and initializes the OPMPHM.
Basic functions.

**Return values**

| | |
|---|---|
| *Opmphm* | * success |
| *NULL* | memory error |

#### 573.132.2.13 opmphmOptC()

```
double opmphmOptC (
            size_t n )
```

Provides the optimal `c` value for a given `n`
This is a heuristic, the return values follow from the mapping benchmark.

**Parameters**

| | |
|---|---|
| *n* | the number of elements to hash |

**Return values**

| | |
|---|---|
| *c* | the optimal `c` value |

#### 573.132.2.14 opmphmOptR()

```
uint8_t opmphmOptR (
            size_t n )
```

Provides the optimal `r` value for a given `n`
This is a heuristic, the return values follow from the mapping benchmark.

**Parameters**

| | |
|---|---|
| *n* | the number of elements to hash |

**Return values**

| | |
|---|---|
| *r* | the optimal rUniPar |

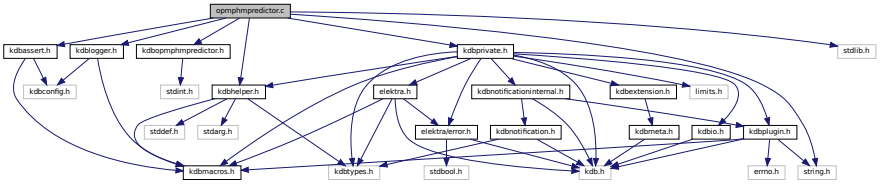
## 573.133 opmphmpredictor.c File Reference

The Order Preserving Minimal Perfect Hash Map Predictor.

```
#include <kdbassert.h>
#include <kdbhelper.h>
#include <kdblogger.h>
```

```
#include <kdbopmphmpredictor.h>
#include <kdbprivate.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for opmphmpredictor.c:




## Functions

- size_t opmphmPredictorWorthOpmphm (size_t n)

  *Heuristic function above the OPMPHM usage is worth.*
- void opmphmPredictorIncCountOpmphm (OpmphmPredictor *op)

  *Increases the counter when the OPMPHM was used for the ksLookup (...) .*
- int opmphmPredictorIncCountBinarySearch (OpmphmPredictor *op, size_t n)

  *Increases the counter when the Binary Search was used for the ksLookup (...) .*
- int opmphmPredictor (OpmphmPredictor *op, size_t n)

  *Predictcs at the first ksLookup (...) after a KeySet changed if it will be worth using the OPMPHM.*
- OpmphmPredictor * opmphmPredictorNew (void)

  *Allocates and initializes the OpmphmPredictor.*
- void opmphmPredictorCopy (OpmphmPredictor *dest, OpmphmPredictor *source)

  *Make a copy of the OpmphmPredictor.*
- void opmphmPredictorDel (OpmphmPredictor *op)

  *Deletes the OpmphmPredictor.*

## Variables

- const uint16_t opmphmPredictorHistoryMask = 0x1FF

  *The benchmarked and evaluated values of the predictors configuration.*
- const size_t opmphmPredictorActionLimit = 599

  *The opmphmPredictorActionLimit define the minimum KeySet size necessary for predictor actions.*

### 573.133.1 Detailed Description

The Order Preserving Minimal Perfect Hash Map Predictor.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.133.2 Function Documentation

#### 573.133.2.1 opmphmPredictor()

```
int opmphmPredictor (
            OpmphmPredictor * op,
            size_t n )
```

Predictcs at the first ksLookup (...) after a KeySet changed if it will be worth using the OPMPHM.
Predictor functions.
Uses the opmphmPredictorWorthOpmphm (...) to check if the previous sequence of ksLookup (...) invocations without alteration was worth using the OPMPHM. Updates the state with the predictionAutomata and the worth information of the previous history. Alters the history with the worth information and makes the prediction for the next sequence of ksLookup (...) invocations.

**Parameters**

| | |
|---|---|
| *op* | the Predictor |
| *n* | the number of elements in the KeySet |

**Return values**

| | |
|---|---|
| *1* | it is worth using the OPMPHM |
| *0* | it is not worth using the OPMPHM |

#### 573.133.2.2 opmphmPredictorCopy()

```
void opmphmPredictorCopy (
            OpmphmPredictor * dest,
            OpmphmPredictor * source )
```

Make a copy of the OpmphmPredictor.

**Parameters**

| | |
|---|---|
| *source* | the OpmphmPredictor source |
| *dest* | the OpmphmPredictor destination |

#### 573.133.2.3 opmphmPredictorDel()

```
void opmphmPredictorDel (
            OpmphmPredictor * op )
```

Deletes the OpmphmPredictor.
Clears and frees all memory in OpmphmPredictor.

**Parameters**

| | |
|---|---|
| *op* | the OpmphmPredictor |

#### 573.133.2.4 opmphmPredictorIncCountBinarySearch()

```
int opmphmPredictorIncCountBinarySearch (
            OpmphmPredictor * op,
            size_t n )  [inline]
```

Increases the counter when the Binary Search was used for the ksLookup (...) .
Prevents also an endless Binary Search usage by a simple heuristic.

**Parameters**

| | |
|---|---|
| *op* | the Predictor |
| *n* | the number of elements in the KeySet |

**Return values**

| | |
|---|---|
| *1* | it is worth using the OPMPHM |
| *0* | it is not worth using the OPMPHM |

#### 573.133.2.5 opmphmPredictorIncCountOpmphm()

```
void opmphmPredictorIncCountOpmphm (
            OpmphmPredictor * op )  [inline]
```

Increases the counter when the OPMPHM was used for the ksLookup (...) .

**Parameters**

| | |
|---|---|
| *op* | the Predictor |

#### 573.133.2.6 opmphmPredictorNew()

```
OpmphmPredictor* opmphmPredictorNew (
            void  )
```

Allocates and initializes the OpmphmPredictor.
Basic functions.
Reserves for all possible values of opmphmPredictorHistoryMask two bits to store all 4 states of the Prediction Automata A2. Sets the initial state to 0.

**Return values**

| | |
|---|---|
| *OpmphmPredictor* | * success |
| *NULL* | memory error |

#### 573.133.2.7 opmphmPredictorWorthOpmphm()

```
size_t opmphmPredictorWorthOpmphm (
            size_t n )  [inline]
```

Heuristic function above the OPMPHM usage is worth.
Heuristic function.
This easy and fast computable heuristic function tells how many ksLookup (...) invocations without alteration of the KeySet need to be made to justify the OPMPHM usage. This heuristic function is developed and measured by benchmarks.

**Parameters**

| | |
|---|---|
| *n* | the number of elements in the KeySet |

**Return values**

| | |
|---|---|
| *size_t* | the heuristic value |

### 573.133.3 Variable Documentation

#### 573.133.3.1 opmphmPredictorHistoryMask

```
const uint16_t opmphmPredictorHistoryMask = 0x1FF
```

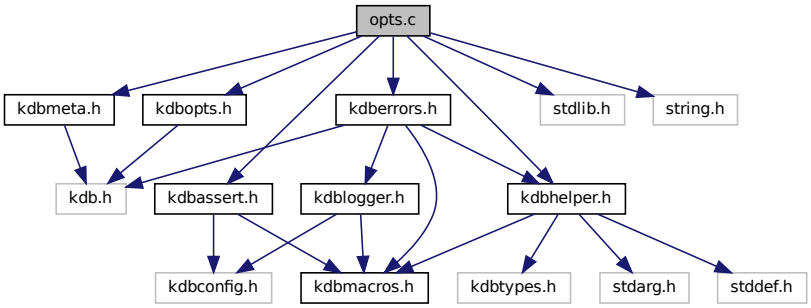
The benchmarked and evaluated values of the predictors configuration.
Order Preserving Minimal Perfect Hash Map Predictor.

## 573.134 opts.c File Reference

Support library used by plugin gopts.

```
#include <kdbopts.h>
#include <stdlib.h>
#include <string.h>
#include <kdbease.h>
#include <kdbhelper.h>
#include <kdbmeta.h>
#include <kdbassert.h>
#include <kdberrors.h>
```

Include dependency graph for opts.c:




### Functions

- char * generateUsageLine (const char *progname, Key *command, const Key *commandArgs)

  *Generate usage line for help message from optionsSpec.*
- int elektraGetOpts (KeySet *ks, int argc, const char **argv, const char **envp, Key *parentKey)

  *This functions parses a specification of program options, together with a list of arguments and environment variables to extract the option values.*
- const char * elektraGetOptsHelpCommand (Key *errorKey)

  *Extracts the command whose help message was requested from the `errorKey` used in elektraGetOpts().*
- char * elektraGetOptsHelpMessage (Key *helpKey, const char *usage, const char *prefix)

  *Extracts the help message from the `helpKey` used in elektraGetOpts().*

### 573.134.1 Detailed Description

Support library used by plugin gopts.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.134.2 Function Documentation

#### 573.134.2.1 elektraGetOpts()

```
int elektraGetOpts (
            KeySet * ks,
            int argc,
            const char ** argv,
            const char ** envp,
            Key * parentKey )
```

This functions parses a specification of program options, together with a list of arguments and environment variables to extract the option values.
The options have to be defined in the metadata of keys in the spec namespace. If an option value is found for any of the given keys, a new key with the same path but inside the proc namespace will be inserted into `ks`. This enables a cascading lookup to find these values.
Take look at https://www.libelektra.org/tutorials/command-line-options for information on how exactly the specification works.
NOTE: Per default option processing DOES NOT stop, when a non-option string is encountered in `argv`. If you want processing to stop, set the metadata `posixly = 1` on `parentKey`.

**Parameters**

| | |
|---|---|
| *ks* | The KeySet containing the specification for the options. |
| *argc* | The number of strings in argv. |
| *argv* | The arguments to be processed. |
| *envp* | A list of environment variables. This needs to be a null-terminated list of strings of the format 'KEY=VALUE'. |
| *parentKey* | The parent key below which the function will search for option specifications. Also used for error reporting. The key will be translated into the spec namespace automatically, e.g. 'user:/test/parent' will be translated into 'spec:/test/parent', before checking against spec keys. |

**Return values**

| | |
|---|---|
| *0* | on success, this is the only case in which `ks` will be modified |
| *-1* | on error, the error will be set as metadata in `errorKey` |
| *1* | if the help option `--help` was found, use elektraGetOptsHelpMessage() access the generated help message |

#### 573.134.2.2 elektraGetOptsHelpCommand()

```
const char* elektraGetOptsHelpCommand (
            Key * errorKey )
```

Extracts the command whose help message was requested from the `errorKey` used in elektraGetOpts().
NOTE: this only works, if elektraGetOpts() returned 1.

**Parameters**

| | |
|---|---|
| *errorKey* | The same Key as passed to elektraGetOpts() as errorKey. |
| *usage* | If this is not NULL, it will be used instead of the default usage line. |
| *prefix* | If this is not NULL, it will be inserted between the usage line and the options list. |

**Returns**

The command extracted from `errorKey`, or NULL if no command was found. The returned string MUST NOT be freed with elektraFree(). It will be valid as long as `errorKey` is not keyDel()'ed.

### 573.134.2.3 elektraGetOptsHelpMessage()

```
char* elektraGetOptsHelpMessage (
            Key * helpKey,
            const char * usage,
            const char * prefix )
```

Extracts the help message from the `helpKey` used in elektraGetOpts().

**Parameters**

| | |
|---|---|
| *helpKey* | The same Key as passed to elektraGetOpts() as parentKey. |
| *usage* | If this is not NULL, it will be used instead of the default usage line. Use elektraGetOptsHelpCommand() to check which command was invoked to get the right usage line. |
| *prefix* | If this is not NULL, it will be inserted between the usage line and the options list. |

**Returns**

The full help message extracted from `helpKey`, or NULL if no help message was found. The returned string has to be freed with elektraFree().

### 573.134.2.4 generateUsageLine()

```
char * generateUsageLine (
            const char * progname,
            Key * command,
            const Key * commandArgs )
```

Generate usage line for help message from optionsSpec.

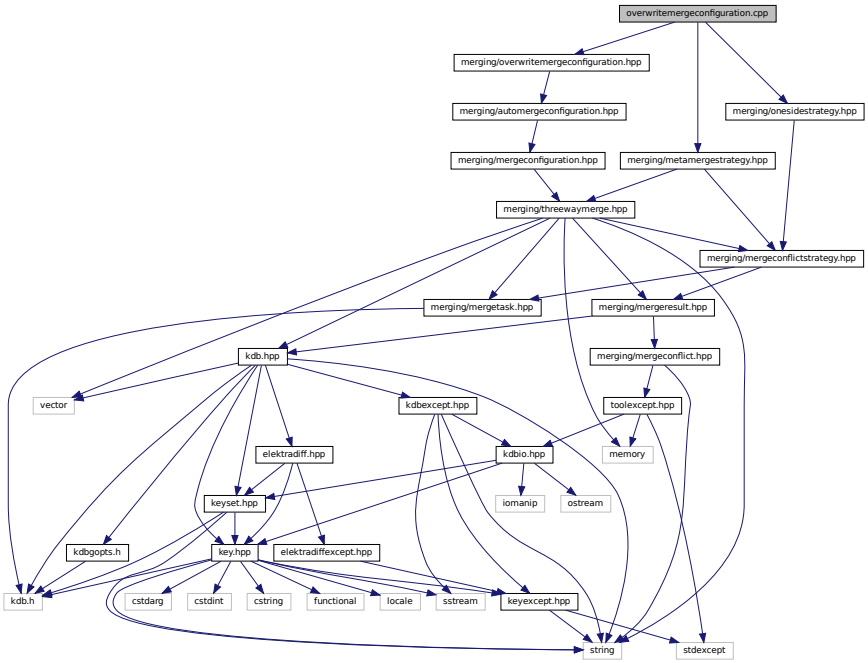
**Returns**

a newly allocated string, must be freed with elektraFree()

## 573.135 overwritemergeconfiguration.cpp File Reference

```
#include <merging/metamergestrategy.hpp>
#include <merging/onesidestrategy.hpp>
#include <merging/overwritemergeconfiguration.hpp>
```

Include dependency graph for overwritemergeconfiguration.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

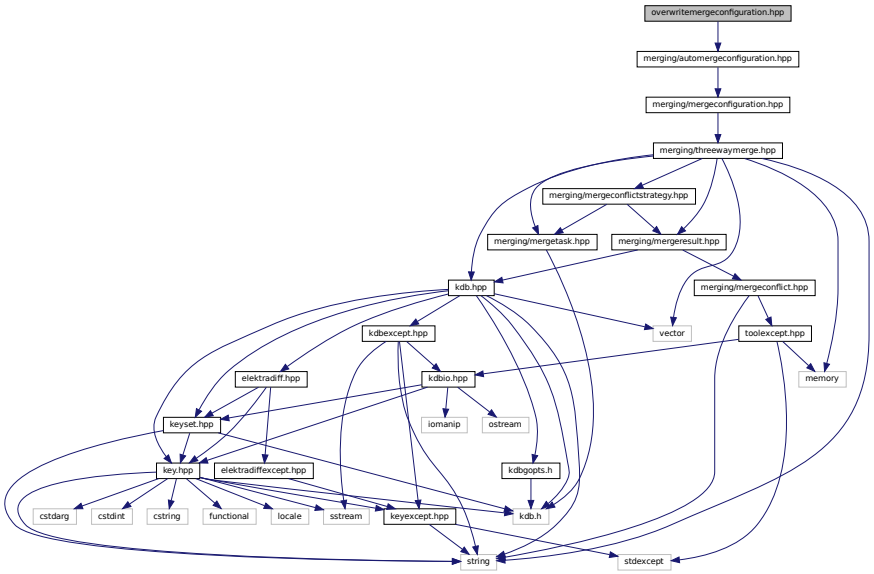
- kdb::tools

  *This namespace is for the libtool library.*

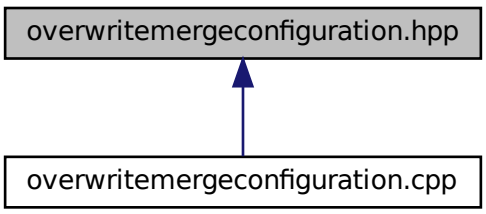
### 573.135.1 Detailed Description

# 573.136 overwritemergeconfiguration.hpp File Reference

A configuration for a simple automerge and guaranteed conflict resolution by one side.
`#include <merging/automergeconfiguration.hpp>`
Include dependency graph for overwritemergeconfiguration.hpp:




This graph shows which files directly or indirectly include this file:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.136.1 Detailed Description

A configuration for a simple automerge and guaranteed conflict resolution by one side.

**Copyright**

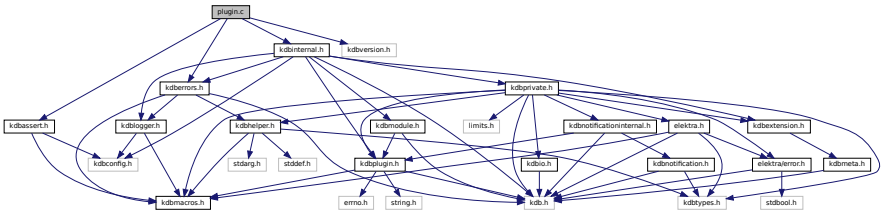
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.137 plugin.c File Reference

Interna of plugin functionality.

```
#include <kdbassert.h>
#include <kdberrors.h>
#include <kdbinternal.h>
#include <kdbversion.h>
```

Include dependency graph for elektra/plugin.c:




## Functions

- Plugin * elektraPluginOpen (const char *name, KeySet *modules, KeySet *config, Key *errorKey)

  *Opens a plugin.*
- size_t elektraPluginGetFunction (Plugin *plugin, const char *name)

  *Retrieves a function exported by a plugin.*
- const char * elektraPluginPhaseName (ElektraKdbPhase phase)

  *Gets a string with the name of the given constant for a plugin phase.*

### 573.137.1 Detailed Description

Interna of plugin functionality.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.137.2 Function Documentation

#### 573.137.2.1 elektraPluginGetFunction()

```
size_t elektraPluginGetFunction (
            Plugin * plugin,
            const char * name )
```

Retrieves a function exported by a plugin.

**Parameters**

| | |
|---|---|
| *plugin* | Plugin handle |
| *name* | Function name. Must be a valid key name suffix. May not contain the sequence '..' |

#### Returns

Pointer to function. NULL if function not found or not enough memory available

### 573.137.2.2 elektraPluginOpen()

```
Plugin* elektraPluginOpen (
            const char * name,
            KeySet * modules,
            KeySet * config,
            Key * errorKey )
```

Opens a plugin.
The config will be used as is. So be sure to transfer ownership of the config to it, with e.g. ksDup(). elektraPlugin↩ Close() will delete the config.

#### Returns

a pointer to a new created plugin or 0 on error

### 573.137.2.3 elektraPluginPhaseName()

```
const char* elektraPluginPhaseName (
            ElektraKdbPhase phase )
```

Gets a string with the name of the given constant for a plugin phase.

#### Parameters

| | |
|---|---|
| *phase* | The ElektraKdbPhase value for which a string representation should be returned |

#### Returns

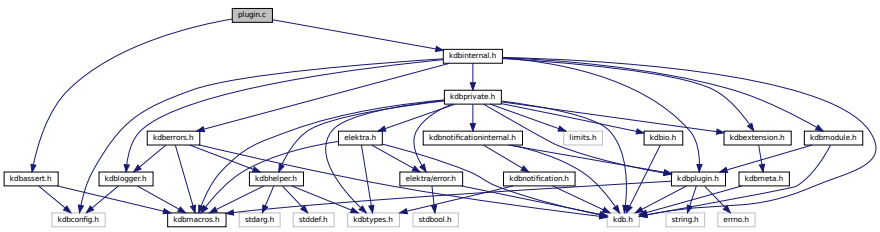
A string with the name of the given phase. The returned string is a constant value and must never be freed!

#### Return values

| | |
|---|---|
| *???* | if an unknown value for the phase was given |

## 573.138 plugin.c File Reference

Access plugin handle.
`#include <kdbassert.h>`
`#include <kdbinternal.h>`
Include dependency graph for plugin/plugin.c:



## Functions

- Plugin * elektraPluginExport (const char *pluginName,...)

  *Allows one to Export Methods for a Plugin.*
- KeySet * elektraPluginGetConfig (Plugin *handle)

  *Returns the configuration of that plugin.*
- void elektraPluginSetData (Plugin *plugin, void *data)

  *Store a pointer to plugin specific data.*
- void * elektraPluginGetData (Plugin *plugin)

  *Get a pointer to the plugin specific data stored before.*
- KeySet * elektraPluginGetGlobalKeySet (Plugin *plugin)

  *Get a pointer to the global keyset.*
- ElektraKdbPhase elektraPluginGetPhase (Plugin *plugin)

  *Returns the current phase of the current KDB operation.*
- Plugin * elektraPluginFromMountpoint (Plugin *plugin, const char *ref)

  *Retrieves the handle for another plugin in the same mountpoint based on a reference.*

### 573.138.1 Detailed Description

Access plugin handle.

**Copyright**

BSD License (see LICENSE.md or `https://www.libelektra.org`)

### 573.138.2 Function Documentation

#### 573.138.2.1 elektraPluginFromMountpoint()

```
Plugin* elektraPluginFromMountpoint (
            Plugin * plugin,
            const char * ref )
```

Retrieves the handle for another plugin in the same mountpoint based on a reference.
The plugins of a mountpoint are defined via `system:/elektra/mountpoint/<mp>/pluigns/<ref>` keys in the declaration of the mountpoint. To use this function, you must provide the `<ref>` part as `ref`.

**Parameters**

| | |
|---|---|
| *plugin* | active plugin handle |
| *ref* | reference to another plugin |

**Returns**

the plugin referenced by `ref`

**Return values**

| | |
|---|---|
| *NULL* | if `plugin`, or `ref` are `NULL`, or no plugin was found for `ref` |

#### 573.138.2.2 elektraPluginGetPhase()

```
ElektraKdbPhase elektraPluginGetPhase (
```

```
Plugin * plugin )
```

Returns the current phase of the current KDB operation.
During kdbGet() this will be one of the `ELEKTRA_KDB_GET_PHASE_*` constants and during kdbSet() it will be one of the `ELEKTRA_KDB_SET_PHASE_*` constants.

**Parameters**

| | |
|---|---|
| *plugin* | plugin handle |

**Returns**

current phase

**Return values**

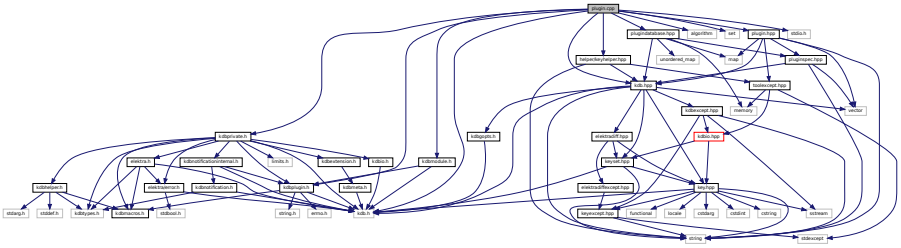
| | |
|---|---|
| *0* | if `plugin` is `NULL` |

## 573.139 plugin.cpp File Reference

Implementation of plugin.

```
#include <kdb.hpp>
#include <helper/keyhelper.hpp>
#include <kdb.h>
#include <kdbmodule.h>
#include <kdbplugin.h>
#include <kdbprivate.h>
#include <plugindatabase.hpp>
#include <algorithm>
#include <set>
#include <plugin.hpp>
#include <stdio.h>
```

Include dependency graph for plugin.cpp:



### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.139.1 Detailed Description

Implementation of plugin.

**Copyright**

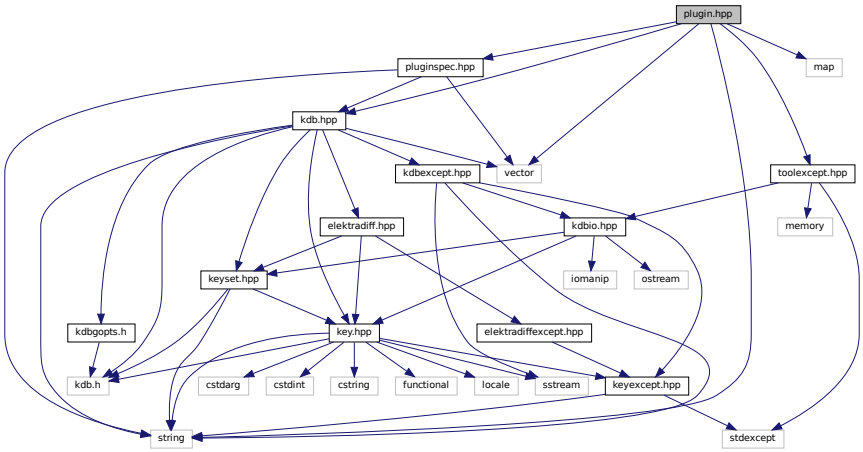
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.140 plugin.hpp File Reference

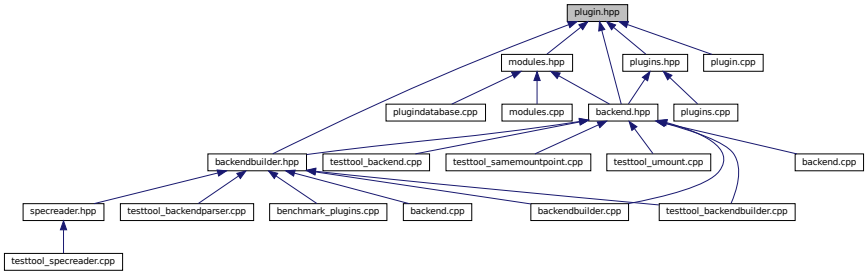
Header file of plugin.

```
#include <kdb.hpp>
#include <pluginspec.hpp>
#include <toolexcept.hpp>
#include <map>
#include <string>
#include <vector>
```

Include dependency graph for plugin.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::tools::Plugin

  *This is a C++ representation of a plugin.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

*This namespace is for the libtool library.*

### 573.140.1 Detailed Description

Header file of plugin.

**Copyright**

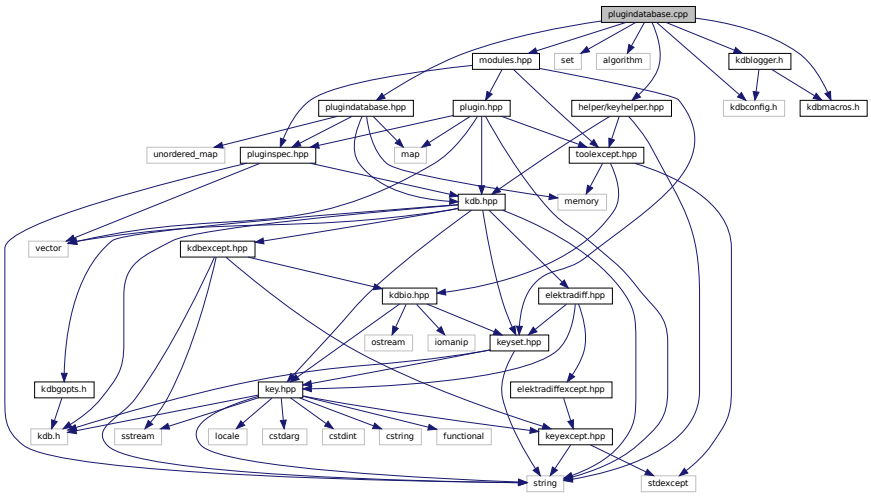
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.141 plugindatabase.cpp File Reference

Implementation of PluginDatabase(s)

```
#include <plugindatabase.hpp>
#include <modules.hpp>
#include <set>
#include <algorithm>
#include <helper/keyhelper.hpp>
#include <kdbconfig.h>
#include <kdblogger.h>
#include <kdbmacros.h>
```

Include dependency graph for plugindatabase.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.141.1 Detailed Description

Implementation of PluginDatabase(s)

**Copyright**

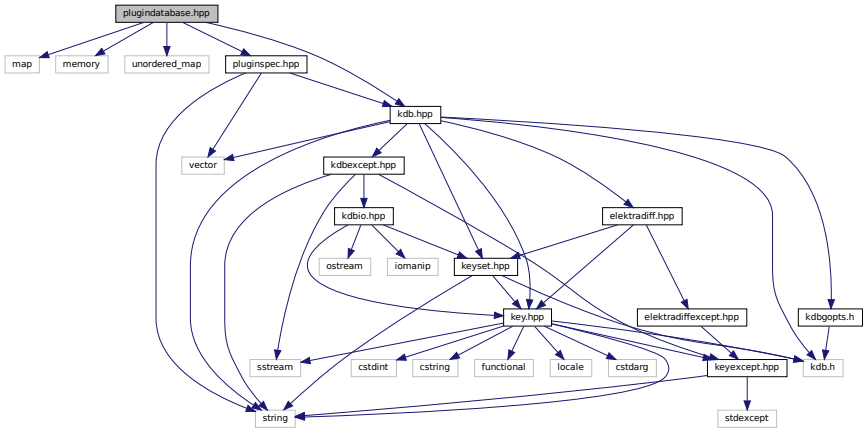
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.142 plugindatabase.hpp File Reference

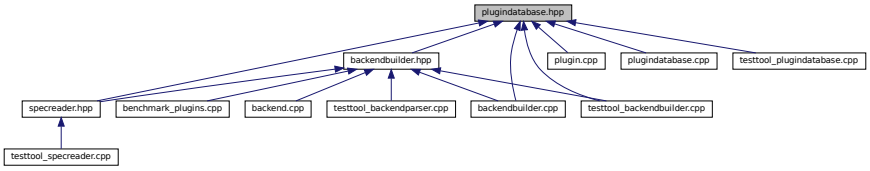
Interface to all plugins.

```
#include <map>
#include <memory>
#include <unordered_map>
#include <kdb.hpp>
#include <pluginspec.hpp>
```

Include dependency graph for plugindatabase.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::tools::PluginDatabase

  *Loads all plugins and allows us to query them.*
- class kdb::tools::ModulesPluginDatabase

  *A plugin database that works with installed modules.*
- class kdb::tools::MockPluginDatabase

  *A plugin database that works with added fake data.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.142.1 Detailed Description

Interface to all plugins.

**Copyright**

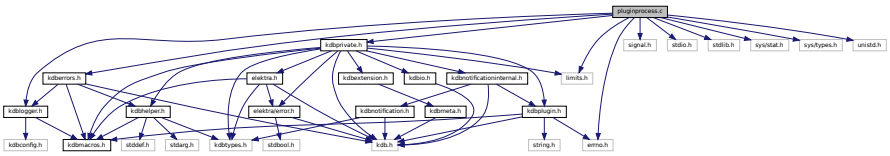
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.143 pluginprocess.c File Reference

Source for the pluginprocess library.

```
#include "kdbpluginprocess.h"
#include <kdberrors.h>
#include <kdbinvoke.h>
#include <kdblogger.h>
#include <kdbprivate.h>
#include <errno.h>
#include <limits.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
```

Include dependency graph for pluginprocess.c:




## Functions

- void elektraPluginProcessStart (Plugin *handle, ElektraPluginProcess *pp)

  *Start the child process' command loop.*
- int elektraPluginProcessSend (const ElektraPluginProcess *pp, pluginprocess_t command, KeySet *originalKeySet, Key *key)

  *Call a plugin's function in a child process.*
- int elektraPluginProcessIsParent (const ElektraPluginProcess *pp)

  *Check if a given plugin process is the parent or the child process.*
- ElektraPluginProcess * elektraPluginProcessInit (Key *errorKey)

  *Initialize a plugin to be executed in its own process.*
- int elektraPluginProcessOpen (ElektraPluginProcess *pp, Key *errorKey)

  *Call a plugin's open function in a child process.*
- ElektraPluginProcessCloseResult elektraPluginProcessClose (ElektraPluginProcess *pp, Key *errorKey)

  *Cleanup a plugin process.*
- void elektraPluginProcessSetData (ElektraPluginProcess *pp, void *data)

  *Store a pointer to any plugin related data that is being executed inside an own process.*
- void * elektraPluginProcessGetData (const ElektraPluginProcess *pp)

  *Get a pointer to any plugin related data stored before.*

### 573.143.1 Detailed Description

Source for the pluginprocess library.
Executes plugins in a separate process via fork and uses a simple communication protocol based on the dump plugin via named pipes.
The communication protocol works as follows, where Child and Parent stand for the child and the parent process: 1) Four pipes are created to handle the communication in a reliable way

- parentCommandPipe is used for a keyset containing metainformation about the communication protocol, going from Parent to Child
- parentPayloadPipe is used for transferring the actual payload that gets passed to a plugin, going from Parent to Child
- childCommandPipe is used for a keyset containing metainformation about the communication protocol, going from Child to Parent
- childPayloadPipe is used for transferring the actual payload that gets passed to a plugin, going from Child to Parent 2) The parent constructs a keyset called commandKeySet, containing:
- /pluginprocess/parent a copy of the parent key passed to the plugin interface
- /pluginprocess/parent/name the name of the parent key passed to the plugin interface
- /pluginprocess/command the command that should be executed by the child process
- /pluginprocess/payload/size the size of the payload keyset, -1 if there is no payload
- /pluginprocess/version a number indicating the version of this communication protocol 3) The parent sends over the commandKeySet over the parentCommandPipe 4) For operations requiring a keyset, the parent sends the keyset (originalKeySet) that is passed to this plugin over the parentPayloadPipe 5) Child receives the commandKeySet 6) Child receives the keyset if one exists 7) Child executes the plugin on the keyset with the originalKey 8) Child adds the result value of the plugin operation to the commandKeySet into the key /pluginprocess/result 9) Child sends the commandKeySet over the childCommandPipe 10) For operations requiring a keyset, Child sends the keyset over the childPayloadPipe 11) Parent receives the commandKey↩ Set and interprets /pluginprocess/result 12) For operations requiring a keyset, Parent receives the keyset and copies it back to originalKeySet set 13) Parent returns the result value from the child process

**Copyright**



### 573.143.2 Function Documentation

#### 573.143.2.1 elektraPluginProcessClose()

```
ElektraPluginProcessCloseResult elektraPluginProcessClose (
            ElektraPluginProcess * pp,
            Key * errorKey )
```

Cleanup a plugin process.
This will decrease the internal counter how often open/close has been called, closing opened pipes when the counter reaches 0. This will not delete the plugin data associated with the handle as it may contain other data out of the scope of this library, so this has to be done manually like

```
int elektraPluginClose (Plugin * handle, Key * errorKey)
{
        ElektraPluginProcess * pp = elektraPluginGetData (handle);
        if (pp && elektraPluginProcessIsParent (pp)) {
                ElektraPluginProcessCloseResult result = elektraPluginProcessClose (pp, errorKey);
                if (result.cleanedUp) elektraPluginSetData (handle, NULL);
                return result.result;
        }
        // actual plugin functionality to be executed in a child process
        return ELEKTRA_PLUGIN_STATUS_SUCCESS;
}
```

Note that pp might be null here if the initialization failed!

**Parameters**

| | |
|---|---|
| *pp* | the data structure containing the plugin's process information |

**Return values**

| | |
|---|---|
| *1* | if the data structure got cleaned up |
| *0* | if the data structure is still used |

### 573.143.2.2 elektraPluginProcessGetData()

```
void* elektraPluginProcessGetData (
            const ElektraPluginProcess * pp )
```

Get a pointer to any plugin related data stored before.
If elektraPluginProcessSetData was not called earlier, NULL will be returned.

**Parameters**

| | |
|---|---|
| *pp* | the data structure containing the plugin's process information |

**Return values**

| | |
|---|---|
| *a* | pointer to the data |

### 573.143.2.3 elektraPluginProcessInit()

```
ElektraPluginProcess* elektraPluginProcessInit (
            Key * errorKey )
```

Initialize a plugin to be executed in its own process.
This will prepare all the required resources and then fork the current process. After the initialization the child process will typically call the command loop while the parent starts to send commands to it. Also the resulting process information has to be stored for the plugin. In order to allow users to handle custom plugin data this will not automatically call elektraPluginSetData.
Typically called in a plugin's open function like (assuming no custom plugin data):

```
int elektraPluginOpen (Plugin * handle, Key * errorKey)
{
        ElektraPluginProcess * pp = elektraPluginGetData (handle);
        if (pp == NULL)
        {
                if ((pp = elektraPluginProcessInit (errorKey)) == NULL) return ELEKTRA_PLUGIN_STATUS_ERROR;
                elektraPluginSetData (handle, pp);
                if (!elektraPluginProcessIsParent (pp)) elektraPluginProcessStart (handle, pp);
        }
        if (elektraPluginProcessIsParent (pp)) return elektraPluginProcessOpen (pp, errorKey);
        // actual plugin functionality to be executed in a child process
        return ELEKTRA_PLUGIN_STATUS_SUCCESS;
}
```

**Parameters**

| | |
|---|---|
| *handle* | the plugin's handle |
| *errorKey* | a key where error messages will be set |

**Return values**

| | |
|---|---|
| *NULL* | if the initialization failed |
| *a* | pointer to the information |

#### 573.143.2.4 elektraPluginProcessIsParent()

```
int elektraPluginProcessIsParent (
            const ElektraPluginProcess * pp )
```

Check if a given plugin process is the parent or the child process.

**Parameters**

| | |
|---|---|
| *pp* | the data structure containing the plugin's process information |

**Return values**

| | |
|---|---|
| *0* | if it's the child process |
| *the* | child process' pid otherwise |

#### 573.143.2.5 elektraPluginProcessOpen()

```
int elektraPluginProcessOpen (
            ElektraPluginProcess * pp,
            Key * errorKey )
```

Call a plugin's open function in a child process.
This will increase the internal counter how often open/close has been called, so the opened pipes and forks will not be closed too early.

**Parameters**

| | |
|---|---|
| *pp* | the data structure containing the plugin's process information |
| *errorKey* | a key where error messages will be set |

**Return values**

| | |
|---|---|
| *the* | return value of the plugin's open function |

**See also**

elektraPluginProcessInit for an example how and where this function is typically used

#### 573.143.2.6 elektraPluginProcessSend()

```
int elektraPluginProcessSend (
            const ElektraPluginProcess * pp,
            pluginprocess_t command,
            KeySet * originalKeySet,
            Key * key )
```

Call a plugin's function in a child process.
This will wrap all the required information to execute the given command in a keyset and send it over to the child process. Then it waits for the child process's answer and copies the result back into the original plugin keyset and plugin key.
Typically called like

```
int elektraPluginSet (Plugin * handle, KeySet * returned, Key * parentKey)
{
        ElektraPluginProcess * pp = elektraPluginGetData (handle);
        if (elektraPluginProcessIsParent (pp)) return elektraPluginProcessSend (pp,
       ELEKTRA_PLUGINPROCESS_SET, returned, parentKey);
        // actual plugin functionality to be executed in a child process
        return ELEKTRA_PLUGIN_STATUS_SUCCESS;
}
```

**Parameters**

| | |
|---|---|
| *pp* | the data structure containing the plugin's process information |
| *command* | the plugin command that should be executed, e.g. ELEKTRA_PLUGINPROCESS_GET |
| *originalKeySet* | the original key set that the parent process receives |
| *key* | the original key the parent process receives |

**Return values**

| | |
|---|---|
| *ELEKTRA_PLUGIN_STATUS_ERROR* | if the child process communication failed |
| *the* | called plugin's return value otherwise |

**See also**

elektraPluginProcessIsParent for checking if we are in the parent or child process

### 573.143.2.7 elektraPluginProcessSetData()

```
void elektraPluginProcessSetData (
            ElektraPluginProcess * pp,
            void * data )
```

Store a pointer to any plugin related data that is being executed inside an own process.
This is required in case additional arbitrary plugin data should be stored. Pluginprocess has to be stored using elektraPluginSetData. Plugin data for the child process has to be stored using this function like

```
int elektraPluginOpen (Plugin * handle, Key * errorKey)
{
        ElektraPluginProcess * pp = elektraPluginGetData (handle);
        if (pp == NULL)
        {
                if ((pp = elektraPluginProcessInit (errorKey)) == NULL) return ELEKTRA_PLUGIN_STATUS_ERROR;
                ArbitraryPluginData * data = // initialize your plugin data
                elektraPluginProcessSetData (pp, data);
                elektraPluginSetData (handle, pp);
                if (!elektraPluginProcessIsParent (pp)) elektraPluginProcessStart (handle, pp);
        }
        if (elektraPluginProcessIsParent (pp)) return elektraPluginProcessOpen (pp, errorKey);
        // actual plugin functionality to be executed in a child process
        return ELEKTRA_PLUGIN_STATUS_SUCCESS;
}
```

Furthermore ensure to cleanup the data after the plugin is done like

```
int elektraPluginClose (Plugin * handle, Key * errorKey)
{
        ElektraPluginProcess * pp = elektraPluginGetData (handle);
        if (elektraPluginProcessIsParent (pp)) {
                ArbitraryPluginData * data = elektraPluginProcessGetData (pp);
                ElektraPluginProcessCloseResult result = elektraPluginProcessClose (pp, errorKey);
                if (result.cleanedUp)
                {
                        elektraPluginSetData (handle, NULL);
                        // cleanup data here, this was the last call to the plugin
                }
                return result.result;
        }
```

```
        // actual plugin functionality to be executed in a child process
        return ELEKTRA_PLUGIN_STATUS_SUCCESS;
}
```

This way you can use elektraPluginProcessGetData (handle) in your child process to get the data you want your plugin to work with.

**Parameters**

| | |
|---|---|
| *pp* | the data structure containing the plugin's process information |
| *data* | the pointer to the data |

#### 573.143.2.8 elektraPluginProcessStart()

```
void elektraPluginProcessStart (
            Plugin * handle,
            ElektraPluginProcess * pp )
```

Start the child process' command loop.
This will make the child process wait for plugin commands and execute them, returning the result to the parent. This is typically called in a plugin's open function.

**Parameters**

| | |
|---|---|
| *handle* | the plugin's handle |
| *pp* | the data structure containing the plugin's process information |

**See also**

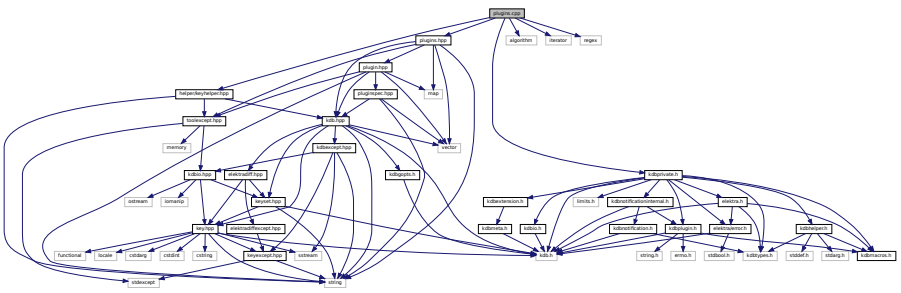
elektraPluginProcessInit how to use this function in a Plugins

## 573.144 plugins.cpp File Reference

Implementation of set/get/error plugins.

```
#include <helper/keyhelper.hpp>
#include <plugins.hpp>
#include <kdbprivate.h>
#include <algorithm>
#include <iterator>
#include <regex>
```

Include dependency graph for plugins.cpp:



### Namespaces

- kdb

*This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

### 573.144.1 Detailed Description

Implementation of set/get/error plugins.

**Copyright**

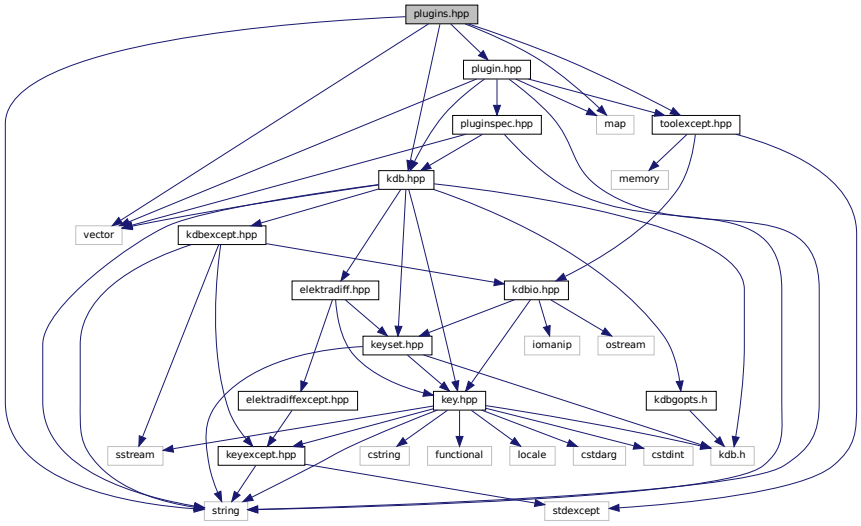
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.145 plugins.hpp File Reference

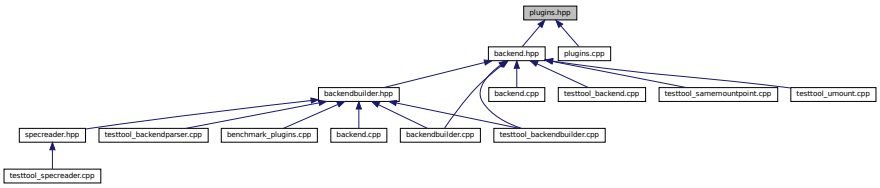
Implementation of get/set and error plugins.

```
#include <plugin.hpp>
#include <toolexcept.hpp>
#include <map>
#include <string>
#include <vector>
#include <kdb.hpp>
```

Include dependency graph for plugins.hpp:




This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::tools::Plugins

*A collection of plugins (either get, set or error)*

- class kdb::tools::GetPlugins

  *Plugins to get configuration.*
- class kdb::tools::SetPlugins

  *Plugins to set configuration.*
- class kdb::tools::ErrorPlugins

  *Plugins to handle errors during configuration access.*
- class kdb::tools::CommitPlugins

  *Plugins to handle errors during configuration access.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.145.1 Detailed Description

Implementation of get/set and error plugins.

**Copyright**

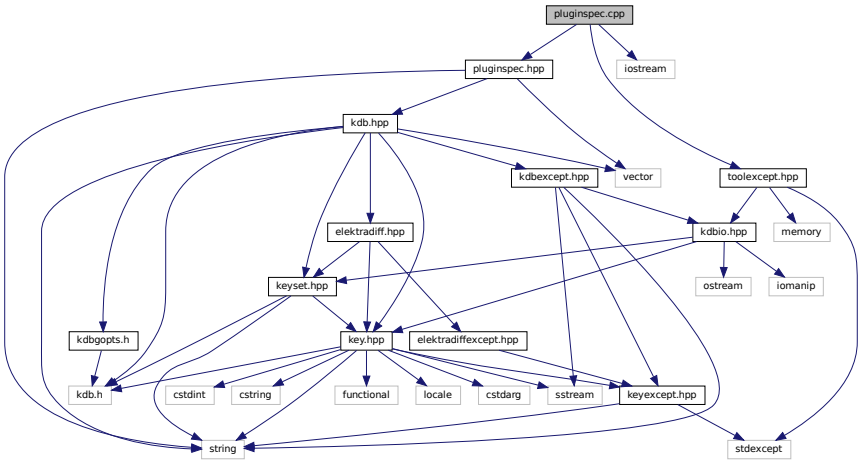
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.146 pluginspec.cpp File Reference

Implementation of plugin spec.

```
#include <pluginspec.hpp>
#include <toolexcept.hpp>
#include <iostream>
```

Include dependency graph for pluginspec.cpp:




## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

## Functions

- bool kdb::tools::operator== (PluginSpec const &self, PluginSpec const &other)

  *Compare two pluginspec if their value is equal.*
- bool kdb::tools::operator!= (PluginSpec const &self, PluginSpec const &other)

  *Compare two pluginspec if their value is not equal.*
- std::ostream & kdb::tools::operator<< (std::ostream &os, PluginSpec const &spec)

  *Output the name, refname and size of config.*

### 573.146.1 Detailed Description

Implementation of plugin spec.

**Copyright**

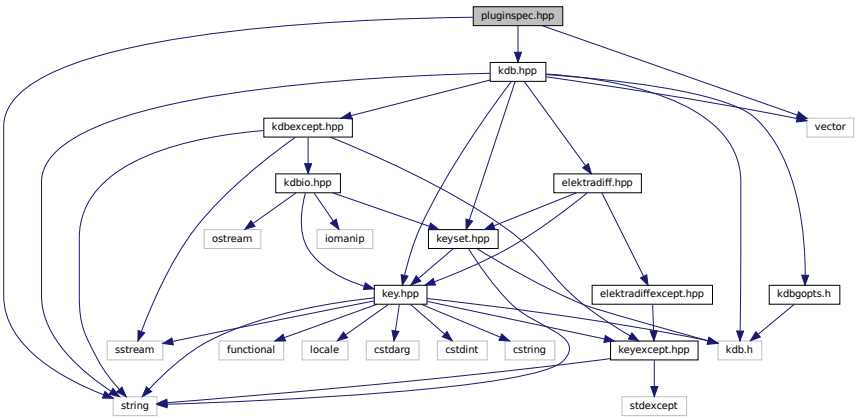
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.147 pluginspec.hpp File Reference

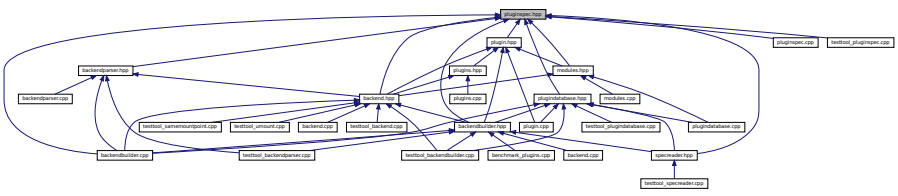
Interface to specify which plugin is meant.

```
#include <string>
#include <vector>
#include <kdb.hpp>
```

Include dependency graph for pluginspec.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- class kdb::tools::PluginSpec

  *Specifies a plugin by its name and configuration.*

- struct kdb::tools::PluginSpecHash

  *Only to be used with PluginSpecName!*

### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### Functions

- std::ostream & kdb::tools::operator<< (std::ostream &os, PluginSpec const &spec)

  *Output the name, refname and size of config.*

### 573.147.1 Detailed Description

Interface to specify which plugin is meant.

**Copyright**

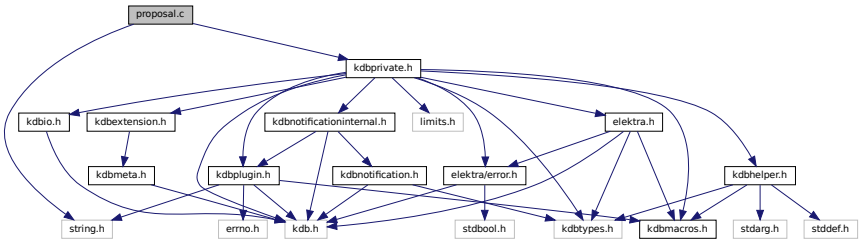
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.148 proposal.c File Reference

Implementation of proposed API enhancements.

```
#include <string.h>
#include <kdbprivate.h>
```

Include dependency graph for proposal.c:




### Functions

- Key * elektraKsPopAtCursor (KeySet *ks, elektraCursor pos)

  *Pop key at given cursor position.*

### 573.148.1 Detailed Description

Implementation of proposed API enhancements.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.148.2 Function Documentation

#### 573.148.2.1 elektraKsPopAtCursor()

```
Key* elektraKsPopAtCursor (
            KeySet * ks,
            elektraCursor pos )
```

Pop key at given cursor position.

**Parameters**

| | |
|---|---|
| *ks* | the keyset to pop key from |
| *c* | where to pop |

**Returns**

the popped key

**Return values**

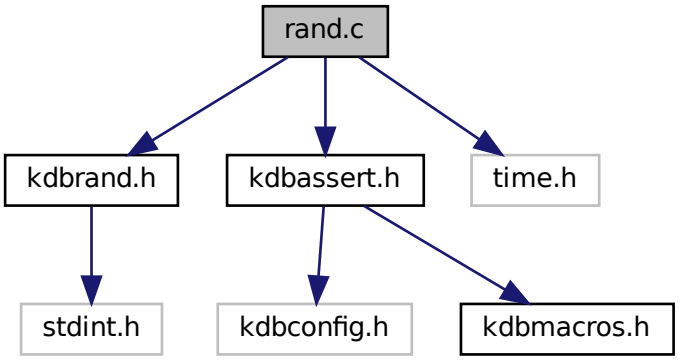
| | |
|---|---|
| *0* | if ks is 0 |

## 573.149 rand.c File Reference

Rand for Elektra.

```
#include "kdbrand.h"
#include <kdbassert.h>
#include <time.h>
```

Include dependency graph for rand.c:




### Functions

- void elektraRand (int32_t *seed)

  *Non cryptographic pseudo random number generator.*
- int32_t elektraRandGetInitSeed (void)

  *Random initial seed generator.*

### 573.149.1 Detailed Description

Rand for Elektra.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.149.2 Function Documentation

#### 573.149.2.1 elektraRand()

```
void elektraRand (
            int32_t * seed )
```

Non cryptographic pseudo random number generator.
By Ray Gardner www8.cs.umu.se/~isak/snippets/rg_rand.c
based on "Random Number Generators: Good Ones Are Hard to Find", S.K. Park and K.W. Miller, Communications of the ACM 31:10 (Oct 1988), and "Two Fast Implementations of the 'Minimal Standard' Random Number Generator", David G. Carta, Comm. ACM 33, 1 (Jan 1990), p. 87-88
linear congruential generator f(z) = 16807 z mod (2 ** 31 - 1)
uses L. Schrage's method to avoid overflow problems
Make sure the initial seed is: 0 < seed < ELEKTRARANDMAX

**Parameters**

| | |
|---|---|
| *seed* | a pointer to the seed |

#### 573.149.2.2 elektraRandGetInitSeed()

```
int32_t elektraRandGetInitSeed (
            void  )
```

Random initial seed generator.
Generates a random initial seed for the `elektraRand (...)` function. Two invocations in the same second return the same random initial seed, due to the usage of `time (0)`.

**Return values**

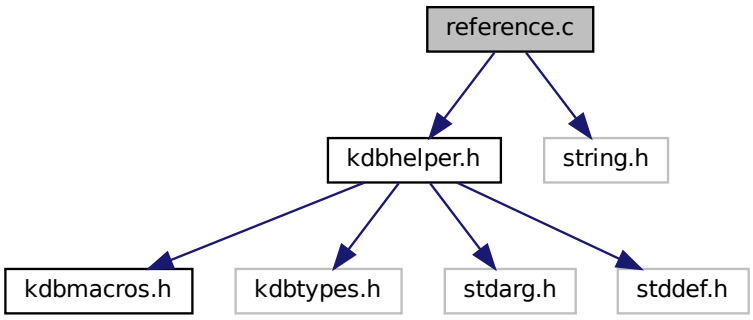
| | |
|---|---|
| *random* | initial seed |

## 573.150 reference.c File Reference

Reference methods.

```
#include "kdbease.h"
#include "kdbhelper.h"
#include <string.h>
```

Include dependency graph for reference.c:




## Functions

- int elektraIsReferenceRedundant (const char ∗reference)

  *Check whether a reference is redundant (i.e.*
- char ∗ elektraResolveReference (const char ∗reference, const Key ∗baseKey, const Key ∗parentKey)

  *Resolve reference into a full keyname.*

### 573.150.1 Detailed Description

Reference methods.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.150.2 Function Documentation

#### 573.150.2.1 elektraIsReferenceRedundant()

```
int elektraIsReferenceRedundant (
            const char * reference )
```

Check whether a reference is redundant (i.e.
it can be expressed in less characters) or not.
This can be used to give a warning to users, because redundant references are often mistakes. Using "../some/key/../path" instead of "../some/path" may indicate, that either a mistake was made while editing, or the concept of references was misunderstood.

**Parameters**

| | |
|---|---|
| *reference* | the reference to check |

**Return values**

| | |
|---|---|
| *1* | if the reference is redundant |
| *0* | otherwise |

#### 573.150.2.2 elektraResolveReference()

```
char* elektraResolveReference (
            const char * reference,
            const Key * baseKey,
            const Key * parentKey )
```

Resolve reference into a full keyname.
References operate like UNIX paths, with some additions:

- '.' refers to the current key
- '..' refers to the parent of the current key
- '@' refers to the parentKey of KDB
- references starting with anything but './', '../' and '@/' as absolute, only embedded '/./' and '/../' as well as '/.' and '/..' at the end will be resolved (like in any keySetName call)

**Parameters**

| | |
|---|---|
| *reference* | The reference to resolve |
| *baseKey* | The key identified by the reference "./" |
| *parentKey* | The key identified by the reference "@/" |

**Returns**

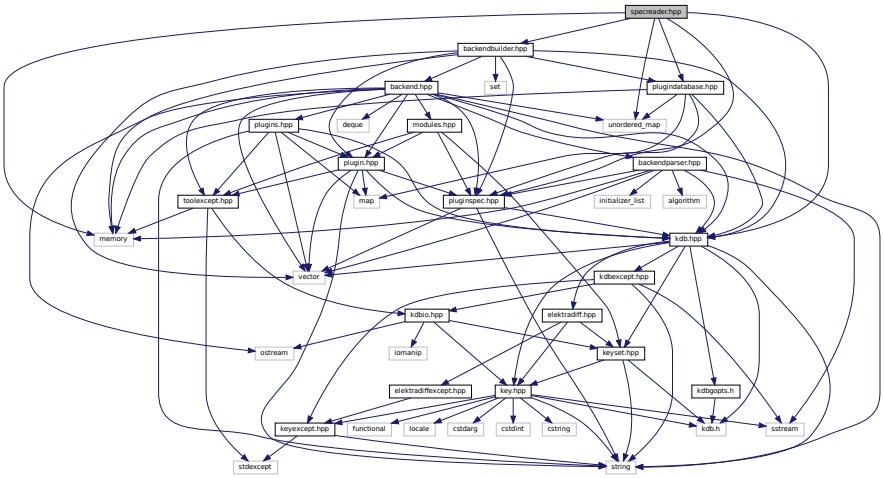
a newly allocated string, containing the full keyname; has to be disposed with elektraFree()

## 573.151 specreader.hpp File Reference

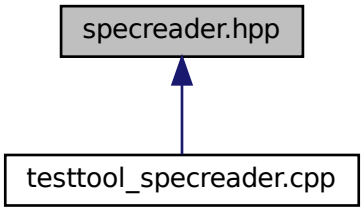
Implements a way to read spec for mounting purposes.

```
#include <kdb.hpp>
#include <backendbuilder.hpp>
#include <plugindatabase.hpp>
#include <pluginspec.hpp>
#include <memory>
#include <unordered_map>
```

Include dependency graph for specreader.hpp:



This graph shows which files directly or indirectly include this file:




## Classes

- class kdb::tools::SpecBackendBuilder

  *Build individual backend while reading specification.*
- class kdb::tools::SpecReader

  *Highlevel interface to build a backend from specification.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.151.1 Detailed Description

Implements a way to read spec for mounting purposes.

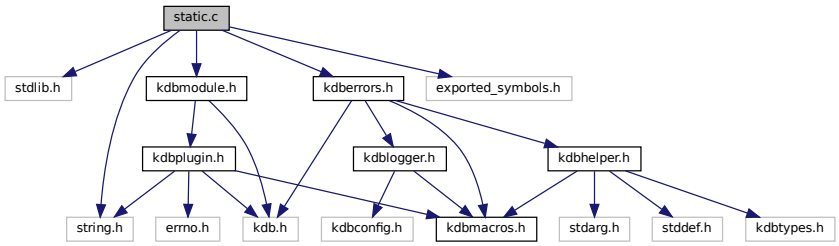
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.152 static.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <exported_symbols.h>
#include <kdberrors.h>
#include <kdbmodule.h>
```

Include dependency graph for static.c:




## Functions

- int elektraModulesInit (KeySet *modules, Key *error)

  *Initialises the module loading system.*
- elektraPluginFactory elektraModulesLoad (KeySet *modules, const char *name, Key *error)

  *Load a library with the given name.*
- int elektraModulesClose (KeySet *modules, Key *error)

  *Close all modules.*

### 573.152.1 Detailed Description

**Copyright**

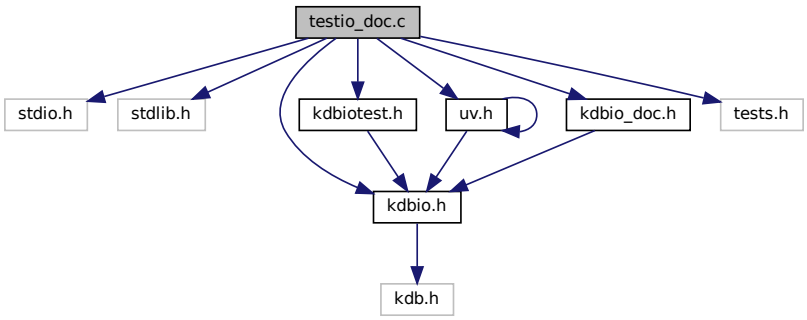
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.153 testio_doc.c File Reference

Tests for I/O doc binding.

```
#include <stdio.h>
#include <stdlib.h>
#include <kdbio.h>
#include <kdbiotest.h>
#include <uv.h>
#include "kdbio_doc.h"
#include <tests.h>
```

Include dependency graph for testio_doc.c:



### Functions

- int main (int argc, char **argv)

  *[kdbio testsuite main]*

### 573.153.1 Detailed Description

Tests for I/O doc binding.

**Copyright**

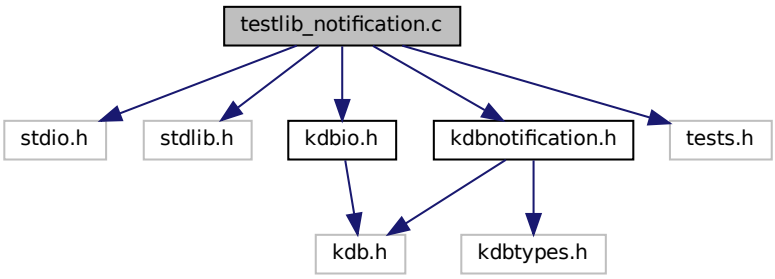
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.154 testlib_notification.c File Reference

Tests for notification library.

```
#include <stdio.h>
#include <stdlib.h>
#include <kdbio.h>
#include <kdbnotification.h>
#include <tests.h>
```

Include dependency graph for testlib_notification.c:




### 573.154.1 Detailed Description

Tests for notification library.

**Copyright**

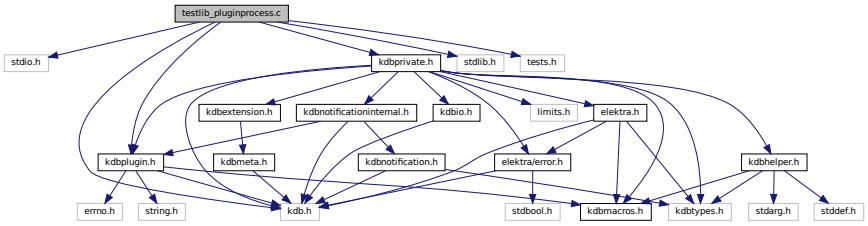
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.155 testlib_pluginprocess.c File Reference

Tests for pluginprocess library.

```
#include <stdio.h>
#include <kdbpluginprocess.h>
#include <kdb.h>
#include <kdbplugin.h>
#include <kdbprivate.h>
#include <stdlib.h>
```

```
#include <tests.h>
```
Include dependency graph for testlib_pluginprocess.c:




### 573.155.1 Detailed Description

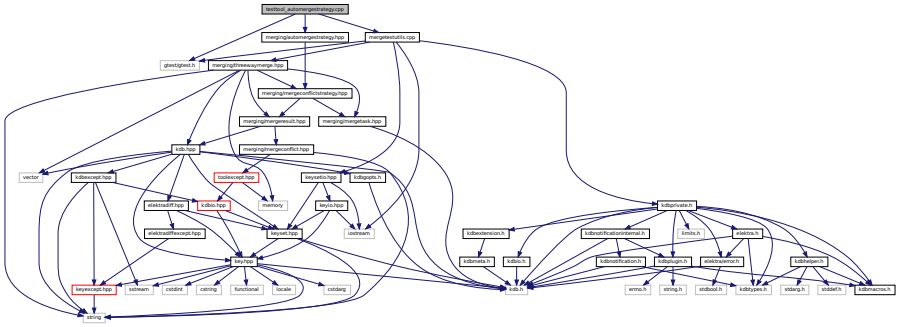
Tests for pluginprocess library.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.156 testtool_automergestrategy.cpp File Reference

Tests for the AutoMergeStrategy.
```
#include "mergetestutils.cpp"
#include <gtest/gtest.h>
#include <merging/automergestrategy.hpp>
```
Include dependency graph for testtool_automergestrategy.cpp:



### 573.156.1 Detailed Description

Tests for the AutoMergeStrategy.

**Copyright**

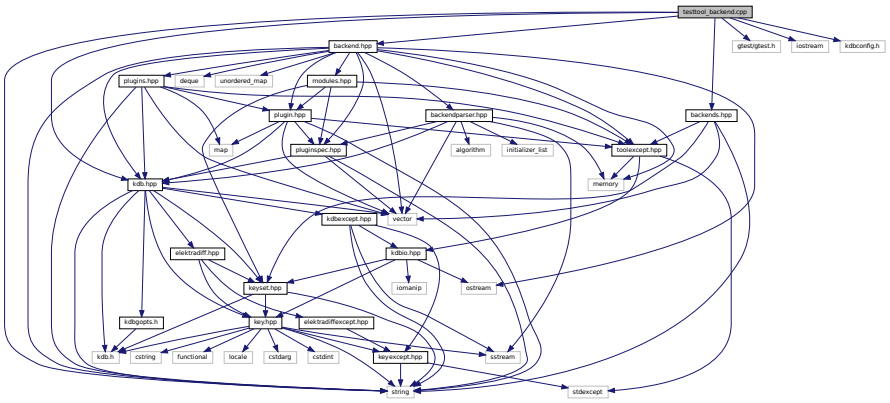
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.157 testtool_backend.cpp File Reference

Tests for the Backend class.
```
#include <backend.hpp>
#include <backends.hpp>
#include <gtest/gtest.h>
#include <iostream>
```

```
#include <kdb.hpp>
#include <kdbconfig.h>
#include <string>
```

Include dependency graph for testtool_backend.cpp:




## Functions

- void outputGTest (kdb::KeySet tocheck, std::string name)

  *Easily allows one to generate regression tests for keysets.*

### 573.157.1 Detailed Description

Tests for the Backend class.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.157.2 Function Documentation

#### 573.157.2.1 outputGTest()

```
void outputGTest (
            kdb::KeySet tocheck,
            std::string name )
```

Easily allows one to generate regression tests for keysets.

**Parameters**

| | |
|---|---|
| *tocheck* | the keyset to check (name + string) |
| *name* | the name of the keyset |

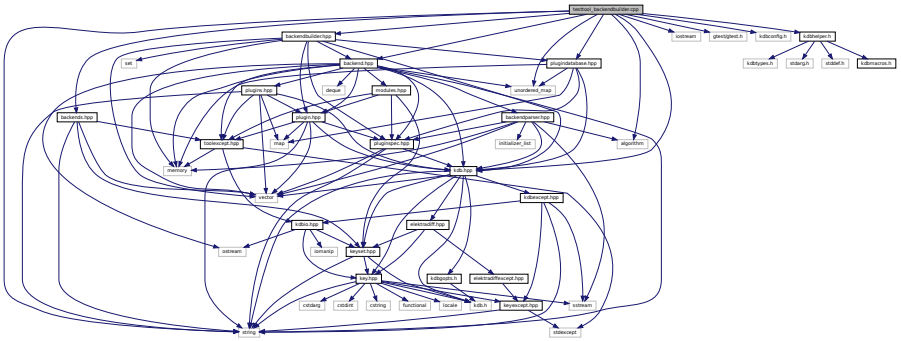
## 573.158 testtool_backendbuilder.cpp File Reference

Tests for the Backend builder class.

```
#include <backendbuilder.hpp>
#include <backend.hpp>
#include <backends.hpp>
#include <plugindatabase.hpp>
```

```
#include <algorithm>
#include <iostream>
#include <string>
#include <unordered_map>
#include <gtest/gtest.h>
#include <kdb.hpp>
#include <kdbconfig.h>
#include <kdbhelper.h>
```

Include dependency graph for testtool_backendbuilder.cpp:



### 573.158.1 Detailed Description

Tests for the Backend builder class.

**Copyright**

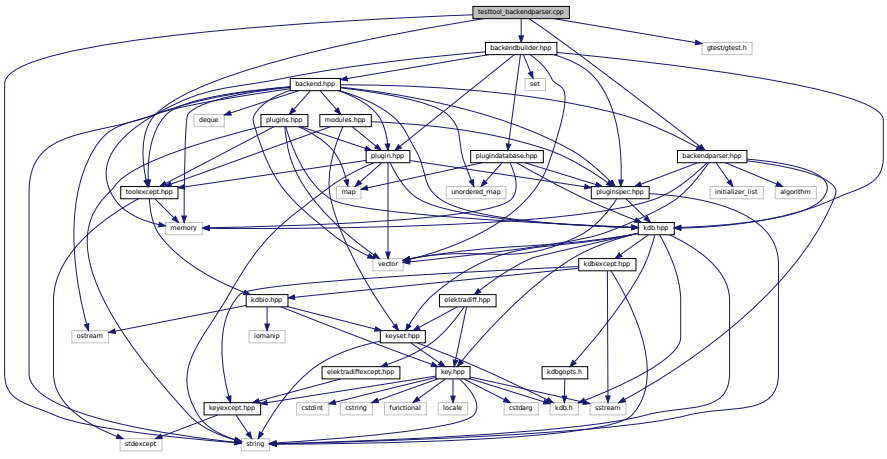
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.159 testtool_backendparser.cpp File Reference

Tests for the Backend parser class.

```
#include <backendparser.hpp>
#include <backendbuilder.hpp>
#include <string>
#include <toolexcept.hpp>
#include <gtest/gtest.h>
```

Include dependency graph for testtool_backendparser.cpp:



### 573.159.1 Detailed Description

Tests for the Backend parser class.

**Copyright**

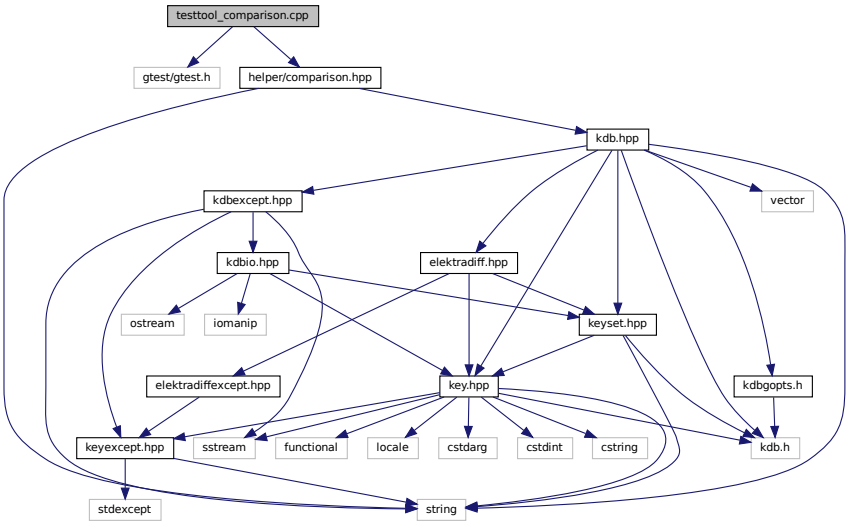
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.160 testtool_comparison.cpp File Reference

Tests for the comparison helper.

```
#include <gtest/gtest.h>
#include <helper/comparison.hpp>
```

Include dependency graph for testtool_comparison.cpp:




### 573.160.1 Detailed Description

Tests for the comparison helper.

**Copyright**

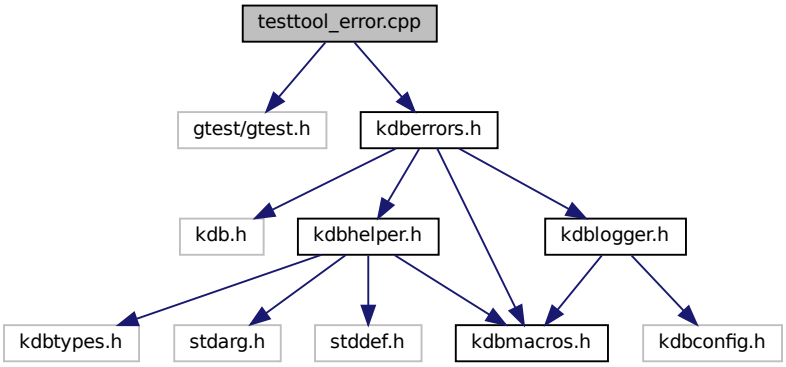
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.161 testtool_error.cpp File Reference

Tests for the errors and warnings.

```
#include <errors/errorFactory.hpp>
#include <errors/errorTypes.hpp>
#include <errors/warningTypes.hpp>
#include <gtest/gtest.h>
#include <kdberrors.h>
```

Include dependency graph for testtool_error.cpp:




### 573.161.1 Detailed Description

Tests for the errors and warnings.

**Copyright**

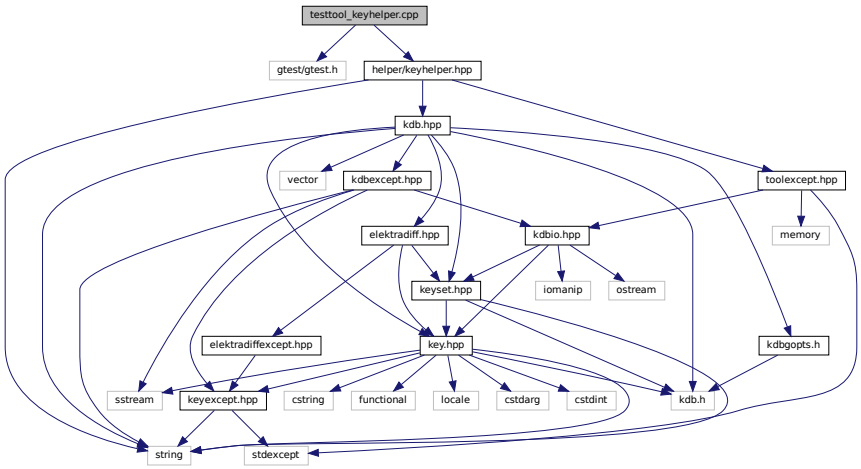
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.162 testtool_keyhelper.cpp File Reference

Tests for the key helper.

```
#include <gtest/gtest.h>
#include <helper/keyhelper.hpp>
```

Include dependency graph for testtool_keyhelper.cpp:




### 573.162.1 Detailed Description

Tests for the key helper.

**Copyright**

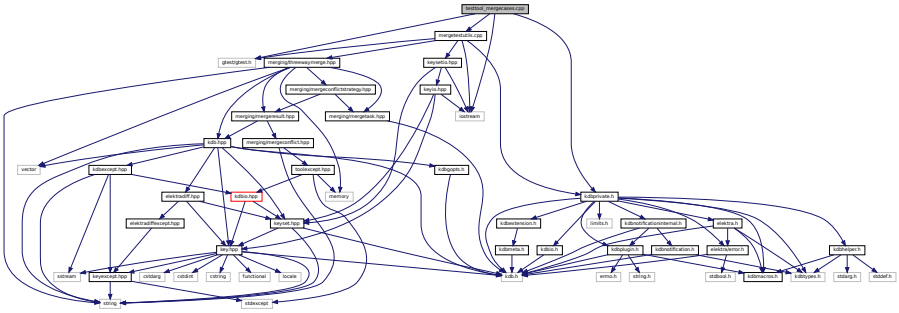
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.163 testtool_mergecases.cpp File Reference

Tests for the ThreeWayMerge.

```
#include "mergetestutils.cpp"
#include <gtest/gtest.h>
#include <iostream>
#include <kdbprivate.h>
```

Include dependency graph for testtool_mergecases.cpp:



## 573.163.1 Detailed Description

Tests for the ThreeWayMerge.

**Copyright**

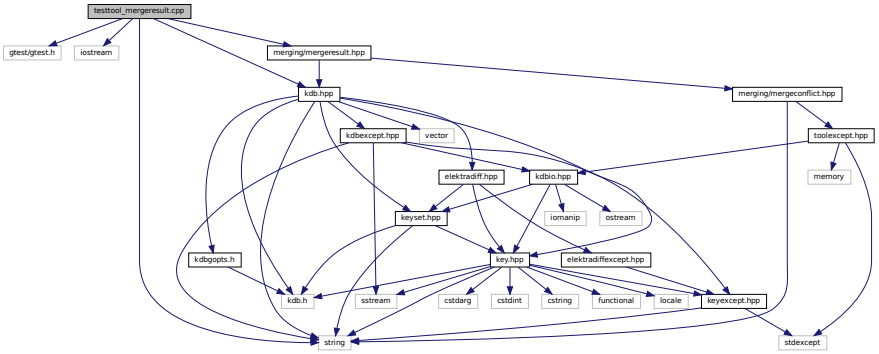
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.164 testtool_mergeresult.cpp File Reference

Tests for the Mergeresult class.

```
#include <gtest/gtest.h>
#include <iostream>
#include <kdb.hpp>
#include <merging/mergeresult.hpp>
#include <string>
```

Include dependency graph for testtool_mergeresult.cpp:



### 573.164.1 Detailed Description

Tests for the Mergeresult class.

**Copyright**

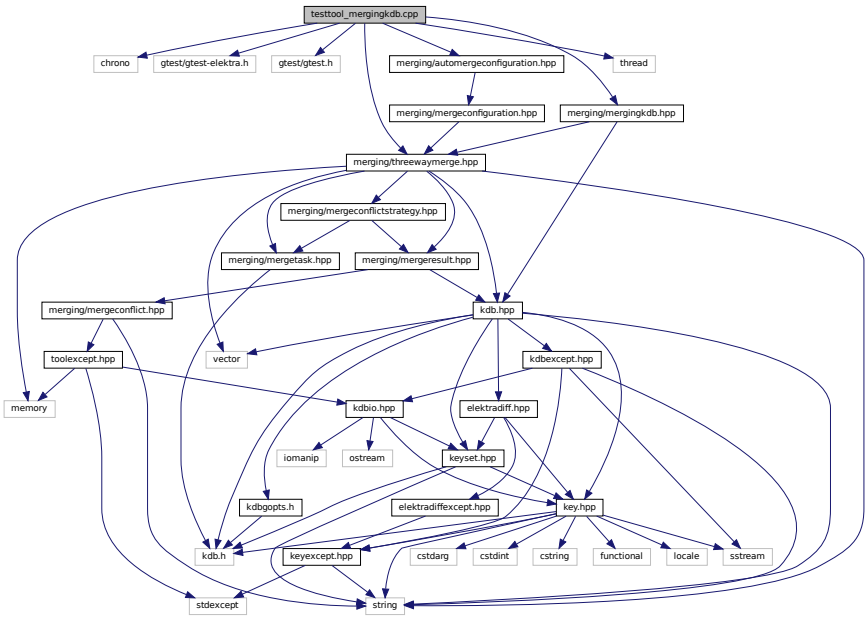
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.165 testtool_mergingkdb.cpp File Reference

Tests for MergingKDB.

```
#include <chrono>
#include <gtest/gtest-elektra.h>
#include <gtest/gtest.h>
#include <merging/automergeconfiguration.hpp>
#include <merging/mergingkdb.hpp>
#include <merging/threewaymerge.hpp>
#include <thread>
```

Include dependency graph for testtool_mergingkdb.cpp:




### 573.165.1 Detailed Description

Tests for MergingKDB.

**Copyright**

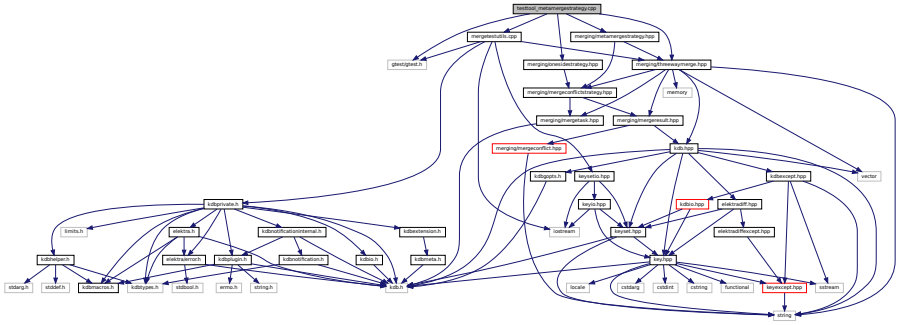
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.166 testtool_metamergestrategy.cpp File Reference

Tests for the MetaMergeStrategy.

```
#include "mergetestutils.cpp"
#include <gtest/gtest.h>
#include <merging/metamergestrategy.hpp>
#include <merging/onesidestrategy.hpp>
```

```
#include <merging/threewaymerge.hpp>
```
Include dependency graph for testtool_metamergestrategy.cpp:



### 573.166.1 Detailed Description

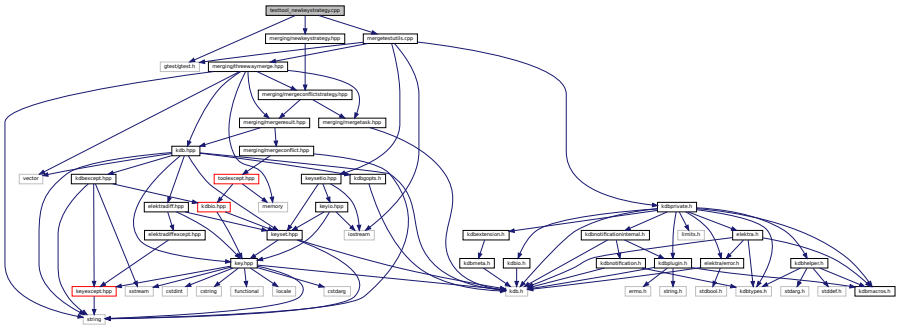
Tests for the MetaMergeStrategy.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.167 testtool_newkeystrategy.cpp File Reference

Tests for the NewKeyStrategy.
```
#include "mergetestutils.cpp"
#include <gtest/gtest.h>
#include <merging/newkeystrategy.hpp>
```
Include dependency graph for testtool_newkeystrategy.cpp:



### 573.167.1 Detailed Description

Tests for the NewKeyStrategy.

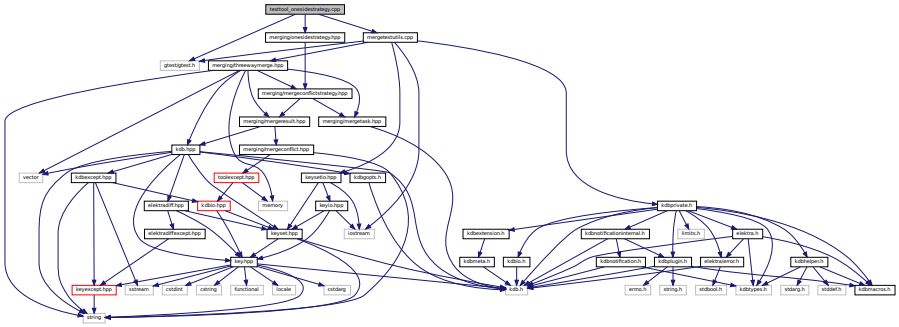
**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.168 testtool_onesidestrategy.cpp File Reference

Tests for the OneSideStrategy.
```
#include "mergetestutils.cpp"
#include <gtest/gtest.h>
```

```
#include <merging/onesidestrategy.hpp>
```
Include dependency graph for testtool_onesidestrategy.cpp:



### 573.168.1 Detailed Description

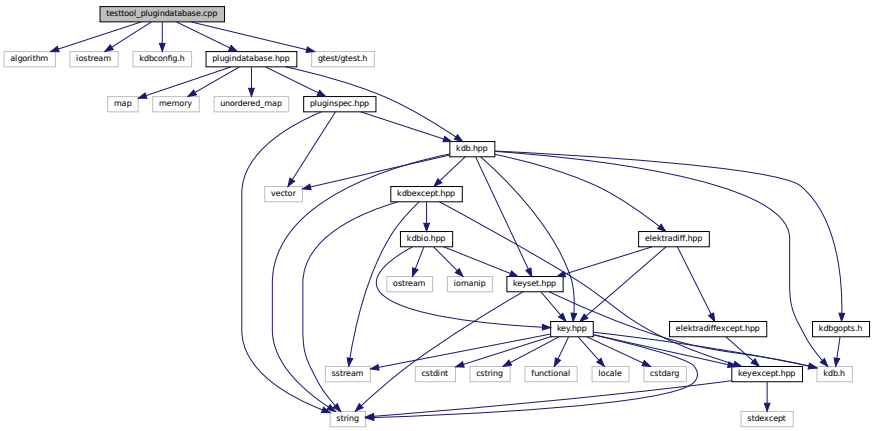
Tests for the OneSideStrategy.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.169 testtool_plugindatabase.cpp File Reference

Tests for the plugindatabase class and implementations of it.
```
#include <algorithm>
#include <iostream>
#include <kdbconfig.h>
#include <plugindatabase.hpp>
#include <gtest/gtest.h>
```
Include dependency graph for testtool_plugindatabase.cpp:




### 573.169.1 Detailed Description

Tests for the plugindatabase class and implementations of it.

**Copyright**

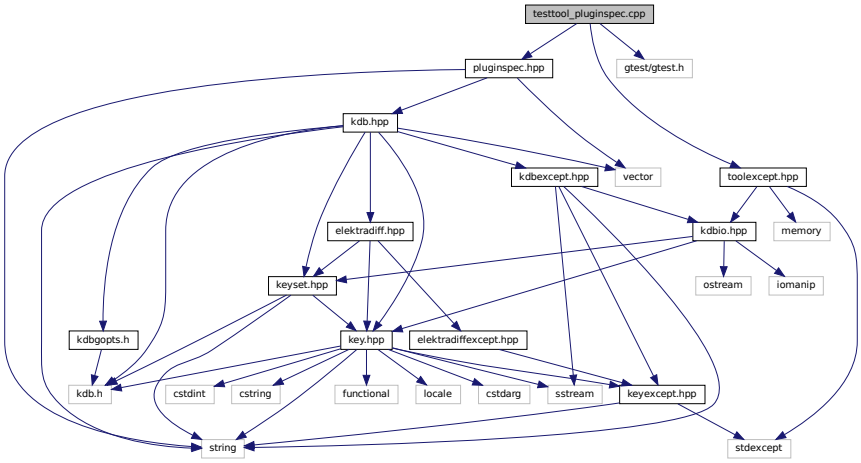
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.170 testtool_pluginspec.cpp File Reference

Tests for the pluginspec class.

```
#include <pluginspec.hpp>
#include <toolexcept.hpp>
#include <gtest/gtest.h>
```

Include dependency graph for testtool_pluginspec.cpp:




## 573.170.1 Detailed Description

Tests for the pluginspec class.

### Copyright

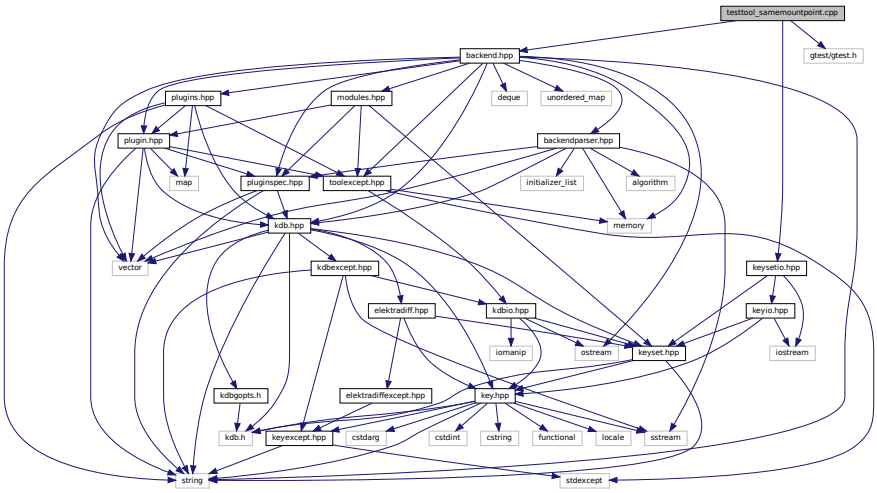
BSD License (see LICENSE.md or https://www.libelektra.org)

# 573.171 testtool_samemountpoint.cpp File Reference

Tests for the Backend class.

```
#include <backend.hpp>
#include <keysetio.hpp>
#include <gtest/gtest.h>
```

Include dependency graph for testtool_samemountpoint.cpp:




### 573.171.1 Detailed Description

Tests for the Backend class.

**Copyright**

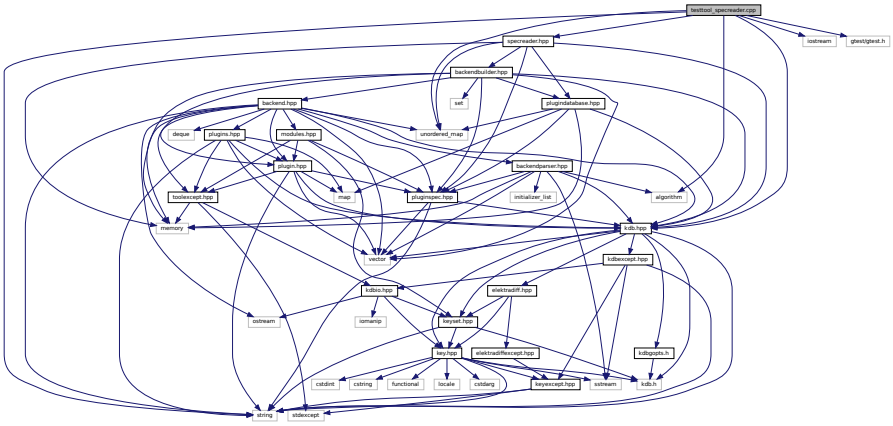
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.172 testtool_specreader.cpp File Reference

Tests for the spec readerclass.

```
#include <specreader.hpp>
#include <algorithm>
#include <iostream>
#include <string>
#include <unordered_map>
#include <gtest/gtest.h>
#include <kdb.hpp>
```

Include dependency graph for testtool_specreader.cpp:



### 573.172.1 Detailed Description

Tests for the spec readerclass.

**Copyright**

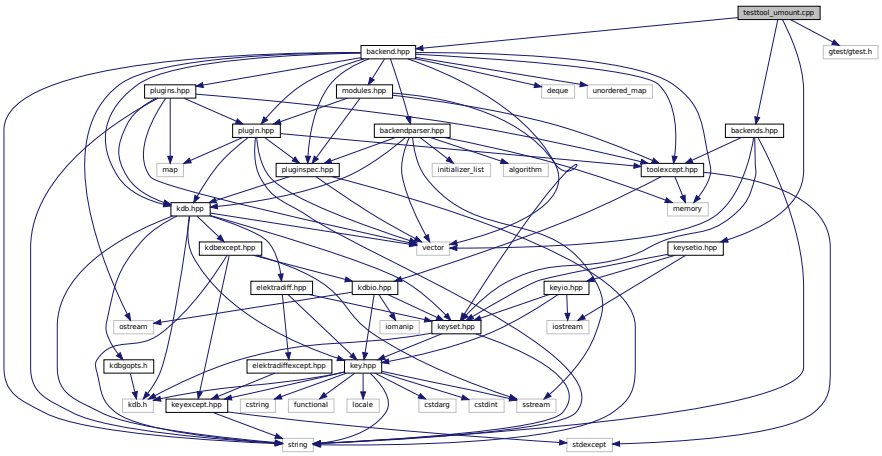
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.173 testtool_umount.cpp File Reference

Tests for the umount.

```
#include <backend.hpp>
#include <backends.hpp>
#include <keysetio.hpp>
#include <gtest/gtest.h>
```

Include dependency graph for testtool_umount.cpp:




### 573.173.1 Detailed Description

Tests for the umount.

**Copyright**

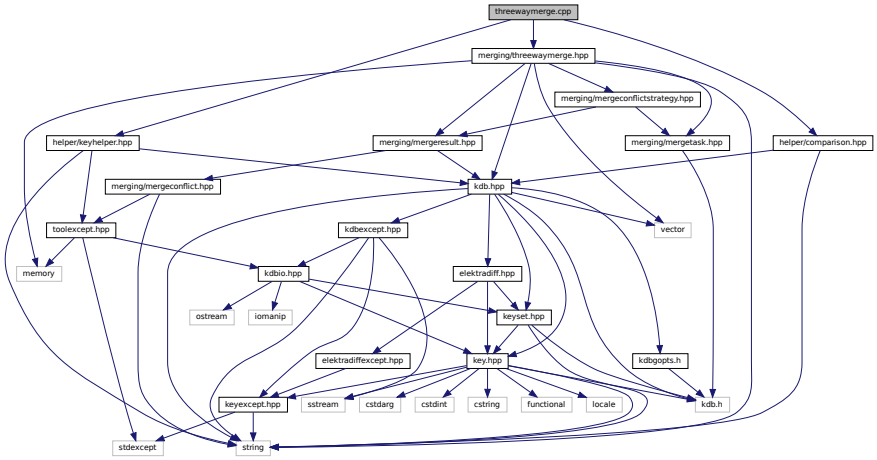
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.174 threewaymerge.cpp File Reference

Implementation of ThreeWayMerge.

```
#include <helper/comparison.hpp>
#include <helper/keyhelper.hpp>
#include <merging/threewaymerge.hpp>
```

Include dependency graph for threewaymerge.cpp:




### Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*

- kdb::tools

  *This namespace is for the libtool library.*

### 573.174.1 Detailed Description

Implementation of ThreeWayMerge.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

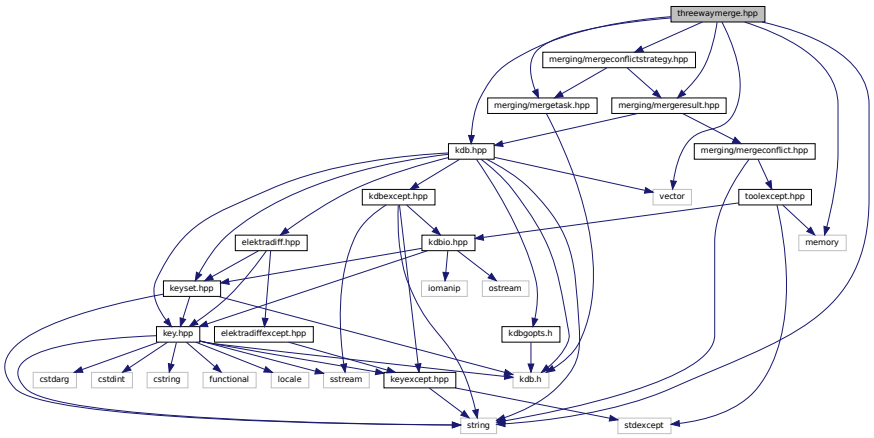
## 573.175 threewaymerge.hpp File Reference

Implements a way to build and deal with a backend.

```
#include <kdb.hpp>
#include <memory>
#include <merging/mergeconflictstrategy.hpp>
#include <merging/mergeresult.hpp>
#include <merging/mergetask.hpp>
#include <string>
```

```
#include <vector>
```

Include dependency graph for threewaymerge.hpp:




This graph shows which files directly or indirectly include this file:

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.175.1 Detailed Description

Implements a way to build and deal with a backend.

**Copyright**

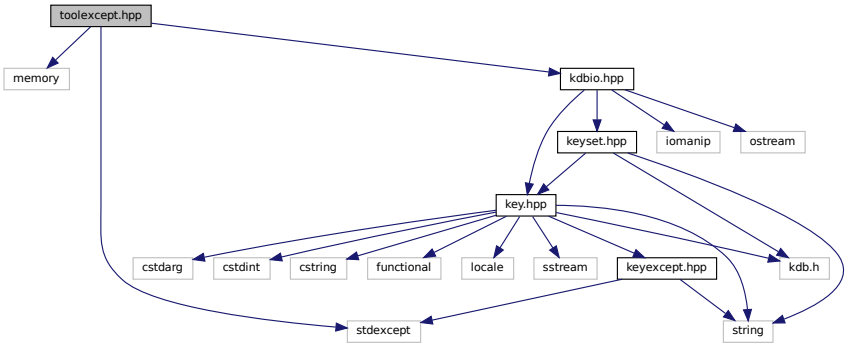
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.176 toolexcept.hpp File Reference

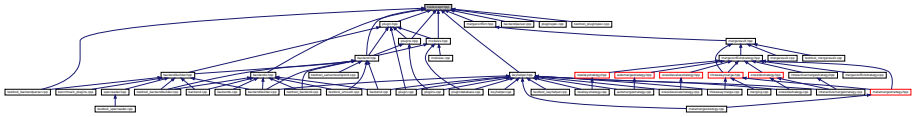
Implementation of all exceptions elektratools library might throw.

```
#include <memory>
#include <stdexcept>
#include <kdbio.hpp>
```

Include dependency graph for toolexcept.hpp:




This graph shows which files directly or indirectly include this file:



## Classes

- struct kdb::tools::ToolException

  *All exceptions from the elektratools library are derived from this exception.*

## Namespaces

- kdb

  *This is the main namespace for the C++ binding and libraries.*
- kdb::tools

  *This namespace is for the libtool library.*

### 573.176.1 Detailed Description

Implementation of all exceptions elektratools library might throw.

**Copyright**

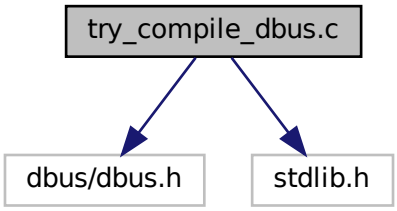
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.177 try_compile_dbus.c File Reference

Compilation test for D-Bus.

```
#include <dbus/dbus.h>
#include <stdlib.h>
```

Include dependency graph for try_compile_dbus.c:




### 573.177.1 Detailed Description

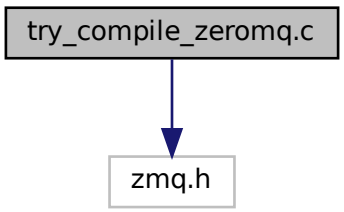
Compilation test for D-Bus.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.178 try_compile_zeromq.c File Reference

Compilation test for ZeroMQ.
`#include <zmq.h>`
Include dependency graph for try_compile_zeromq.c:




### 573.178.1 Detailed Description

Compilation test for ZeroMQ.

**Copyright**

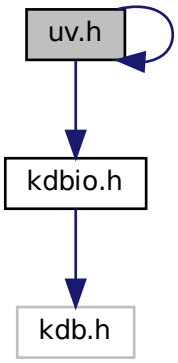
BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.179 uv.h File Reference

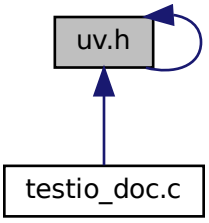
Declarations for the uv I/O binding.

```
#include <kdbio.h>
#include <uv.h>
```

Include dependency graph for uv.h:




This graph shows which files directly or indirectly include this file:




## Functions

- ElektraIoInterface * elektraIoUvNew (uv_loop_t *loop)

  *Create and initialize a new I/O binding.*

### 573.179.1 Detailed Description

Declarations for the uv I/O binding.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.179.2 Function Documentation

#### 573.179.2.1 elektraIoUvNew()

```
ElektraIoInterface* elektraIoUvNew (
            uv_loop_t * loop )
```

Create and initialize a new I/O binding.

**Parameters**

| | |
|---|---|
| *loop* | Loop to use for I/O operations |

**Returns**

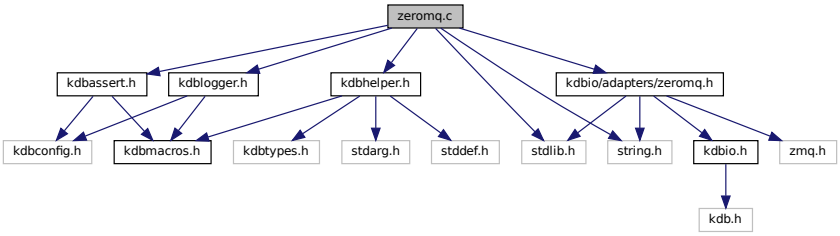
Populated I/O interface

## 573.180 zeromq.c File Reference

I/O Adapter for D-Bus.

```
#include <kdbassert.h>
#include <kdbhelper.h>
#include <kdbio/adapters/zeromq.h>
#include <kdblogger.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for zeromq.c:




### Functions

- ElektraIoAdapterZeroMqHandle * elektraIoAdapterZeroMqAttach (void *socket, ElektraIoInterface *io↩ Binding, ElektraIoAdapterZeroMqCallbackType type, ElektraIoAdapterZeroMqCallback callback, void *context)

  *Attach to ZeroMq socket.*
- void elektraIoAdapterZeroMqSetContext (ElektraIoAdapterZeroMqHandle *handle, void *context)

  *Set the callback context for a ZeroMq adapter handle.*
- int elektraIoAdapterZeroMqDetach (ElektraIoAdapterZeroMqHandle *handle)

  *Remove ZeroMq socket from I/O binding.*

### 573.180.1 Detailed Description

I/O Adapter for D-Bus.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

### 573.180.2 Function Documentation

#### 573.180.2.1 elektraIoAdapterZeroMqAttach()

```
ElektraIoAdapterZeroMqHandle* elektraIoAdapterZeroMqAttach (
            void * socket,
            ElektraIoInterface * ioBinding,
            ElektraIoAdapterZeroMqCallbackType type,
            ElektraIoAdapterZeroMqCallback callback,
            void * context )
```

Attach to ZeroMq socket.
The callback is called whenever socket is readable or writable (depending on type) and data can be processed. The callback should not do blocking calls (e.g. use ZMQ_DONTWAIT for zmq_*recv()). Since ZeroMq guarantees that multipart messages arrive at once, data will be available. New adapter instances are enabled.

**Parameters**

| | |
|---|---|
| *socket* | ZeroMq socket |
| *ioBinding* | I/O binding |
| *type* | callback type ( |

**See also**

ElektraIoAdapterZeroMqCallbackType)

**Parameters**

| | |
|---|---|
| *callback* | callback |
| *context* | callback context (gets passed to callback) |

**Returns**

handle to use with elektraIoZeroMqAdapterDetach() or NULL on error

#### 573.180.2.2 elektraIoAdapterZeroMqDetach()

```
int elektraIoAdapterZeroMqDetach (
            ElektraIoAdapterZeroMqHandle * handle )
```

Remove ZeroMq socket from I/O binding.
This function frees the passed handle.

**Parameters**

| | |
|---|---|
| *handle* | adapter handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

#### 573.180.2.3 elektraIoAdapterZeroMqSetContext()

```
void elektraIoAdapterZeroMqSetContext (
            ElektraIoAdapterZeroMqHandle * handle,
```

```
void * context )
```

Set the callback context for a ZeroMq adapter handle.
The previous context is replaced and not freed.

**Parameters**

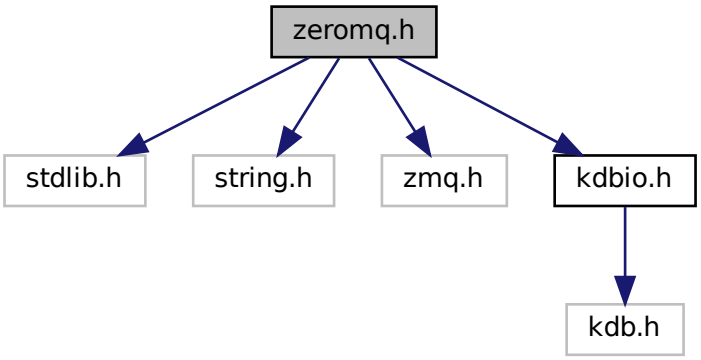
| | |
|---|---|
| *handle* | adapter handle |
| *context* | new callback context |

## 573.181 zeromq.h File Reference

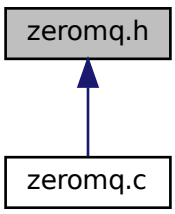
I/O Adapter for D-Bus.

```
#include <stdlib.h>
#include <string.h>
#include <zmq.h>
#include <kdbio.h>
```

Include dependency graph for zeromq.h:




This graph shows which files directly or indirectly include this file:



## Typedefs

- typedef void(* ElektraIoAdapterZeroMqCallback) (void *socket, void *context)

  *Callback for ZeroMq adapter callbacks.*
- typedef struct _ElektraIoAdapterZeroMqHandle ElektraIoAdapterZeroMqHandle

  *D-Bus Adapter Handle.*

## Enumerations

- enum ElektraIoAdapterZeroMqCallbackType { ELEKTRA_IO_ADAPTER_ZEROMQCB_READ = 1 << 0 , ELEKTRA_IO_ADAPTER_ZEROMQCB_WRITE = 1 << 1 }

  *callback types*

## Functions

- ElektraIoAdapterZeroMqHandle * elektraIoAdapterZeroMqAttach (void *socket, ElektraIoInterface *io↩ Binding, ElektraIoAdapterZeroMqCallbackType type, ElektraIoAdapterZeroMqCallback callback, void *context)

  *Attach to ZeroMq socket.*
- void elektraIoAdapterZeroMqSetContext (ElektraIoAdapterZeroMqHandle *handle, void *context)

  *Set the callback context for a ZeroMq adapter handle.*
- int elektraIoAdapterZeroMqDetach (ElektraIoAdapterZeroMqHandle *handle)

  *Remove ZeroMq socket from I/O binding.*

## 573.181.1 Detailed Description

I/O Adapter for D-Bus.

**Copyright**

BSD License (see LICENSE.md or https://www.libelektra.org)

## 573.181.2 Typedef Documentation

### 573.181.2.1 ElektraIoAdapterZeroMqCallback

```
typedef void(* ElektraIoAdapterZeroMqCallback) (void *socket, void *context)
```

Callback for ZeroMq adapter callbacks.

**Parameters**

| | |
|---|---|
| *socket* | zeromq socket |
| *context* | callback context supplied to elektraIoZeroMqAdapterAttach() |

### 573.181.2.2 ElektraIoAdapterZeroMqHandle

```
typedef struct _ElektraIoAdapterZeroMqHandle ElektraIoAdapterZeroMqHandle
```

D-Bus Adapter Handle.
Returned by elektraIoAdapterDbusAttach().

## 573.181.3 Enumeration Type Documentation

#### 573.181.3.1 ElektraIoAdapterZeroMqCallbackType

```
enum ElektraIoAdapterZeroMqCallbackType
```
callback types

**Enumerator**

| | |
|---|---|
| ELEKTRA_IO_ADAPTER_ZEROMQCB_READ | callback is called when socket is readable |
| ELEKTRA_IO_ADAPTER_ZEROMQCB_WRITE | callback is called when socket is writable |

### 573.181.4 Function Documentation

#### 573.181.4.1 elektraIoAdapterZeroMqAttach()

```
ElektraIoAdapterZeroMqHandle* elektraIoAdapterZeroMqAttach (
            void * socket,
            ElektraIoInterface * ioBinding,
            ElektraIoAdapterZeroMqCallbackType type,
            ElektraIoAdapterZeroMqCallback callback,
            void * context )
```
Attach to ZeroMq socket.
The callback is called whenever socket is readable or writable (depending on type) and data can be processed. The callback should not do blocking calls (e.g. use ZMQ_DONTWAIT for zmq_*recv()). Since ZeroMq guarantees that multipart messages arrive at once, data will be available. New adapter instances are enabled.

**Parameters**

| | |
|---|---|
| *socket* | ZeroMq socket |
| *ioBinding* | I/O binding |
| *type* | callback type ( |

**See also**

ElektraIoAdapterZeroMqCallbackType)

**Parameters**

| | |
|---|---|
| *callback* | callback |
| *context* | callback context (gets passed to callback) |

**Returns**

handle to use with elektraIoZeroMqAdapterDetach() or NULL on error

#### 573.181.4.2 elektraIoAdapterZeroMqDetach()

```
int elektraIoAdapterZeroMqDetach (
            ElektraIoAdapterZeroMqHandle * handle )
```
Remove ZeroMq socket from I/O binding.
This function frees the passed handle.

**Parameters**

| | |
|---|---|
| *handle* | adapter handle |

**Return values**

| | |
|---|---|
| *1* | on success |
| *0* | on error |

### 573.181.4.3 elektraIoAdapterZeroMqSetContext()

```
void elektraIoAdapterZeroMqSetContext (
            ElektraIoAdapterZeroMqHandle * handle,
            void * context )
```

Set the callback context for a ZeroMq adapter handle.
The previous context is replaced and not freed.

**Parameters**

| | |
|---|---|
| *handle* | adapter handle |
| *context* | new callback context |