Viability of TOML parsing for configuration

Jakob Fischer

November 26, 2019

Abstract

TODO: Abstract

Chapter 1

Introduction

1.1 TOML

Tom's Obvious, Minimal Language (TOML) is a configuration file format developed since February, 2013 [1]. It claims to be a format, that is easily to read and parse [2] and finds use in many different projects [3]. For instance, it is used with Rust's project manager `cargo` [4], as well as Python's package installer `pip` [5].

1.2 Elektra

Elektra is a configuration framework for accessing configuration settings in a global key database [6]. An application using Elektra can read and write key/value pairs via calls to the Elektra library, making the implementation of an own configuration system obsolete. Furthermore, elektrified applications can access configuration settings of other elektrified applications, to provide better application interoperability.

Although Elektra is mainly written in C, there are bindings for other languages like java, python or ruby [7].

Elektra can also read from and write to different configuration file formats, like JSON, XML or ini [8]. If the need arises, applications can easily switch to a different file format, because configuration access is done with the Elektra API. Developers or Administrators no longer need to commit to one file format for configuration.

Support for different languages and file formats is implemented by the use

of plugins, which can be enabled if needed. With this modularity, Elektra can provide a wide range of functionality, while simultaneously avoiding being a bloated library. Developers can compile Elektra with the exact set of needed functionality. If they don't need bindings for java, they can just disable this feature.

1.3 LCDproc

LCDproc is an open source project for displaying stats like CPU/RAM usage of a system on different kinds of LCD (Liquid Crystal Display) devices [9] [10]. It has a client/server architecture, where the clients provide their system stats to the server, which can display them on a LCD device.

1.4 Research Question

As part of this thesis, we wrote an Elektra storage plugin for reading and writing TOML files. We will test this newly written TOML plugin on ElektraInitiative's fork of LCDproc, where LCDproc is currently in the process of using Elektra for it's handling of configuration. The TOML plugin will be used as the storage plugin for the LCDproc configuration. For evaluation, TOML configuration files for each tested part of LCDproc were created.

There are three questions we want to answer for the TOML plugin:

RQ1 Does the plugin correctly read the LCDproc configuration?

RQ2 Does the plugin correctly write the LCDproc configuration without any loss of information?

RQ3 Does the plugin put an unreasonable overhead on LCDproc loading times?

Ad RQ1 we must ensure, that were are really reading the configuration from our supplied file, and not the default values supplied by the Elektra specification, which each application of LCDproc has.

Ad RQ2 we also have to check, if comments and newlines - which don't affect the LCDproc execution practically, but are only for the user to read - are preserved.

Chapter 2

Implementation

The plugin is implemented in C, using the C99 standard. It requires POSIX for some checks using regular expressions.

2.1 Plugin

The toml parser is realized as an Elektra storage plugin. The plugin exposes two functions, **ElektraTomlGet** for reading, and **ElektraTomlSet** for writing, as expected of a Elektra storage plugin.

2.2 Tools

It uses two external tools for generating a parser for the toml file format: **Flex** [11] for lexical analysis and **bison** [12] for parsing. The key generation is done in the bison parser with the use of actions.

In order to generate the appropriate Elektra keys, the parser has access to a driver which contains all needed Elektra logic. We oriented on the existing Elektra plugin **yambi** [13] in key generation via driver. The driver contains functions of the style `driverEnterKey`/`driverExitTable`, to make it more clear at what point of parsing a certain function shall be invoked.

2.3 Grammar

Since bison generates LR (Left-to-Right, Rightmost derivation) parsers, grammar rules should be be left-recursive. Otherwise, we could run out of stack memory, if recursing too deep with a right-recursive rule. The created parser only contains left-recursive rules. TODO: Maybe explain why/examples/sources

The parser makes use of so-called midrule-actions, which are parser actions that are not at the end of a grammar rule. This can affect the resulting grammar, since after executing a mid-rule action, the parser has to commit to the chosen parse branch [14]. We tried to minimize the usage of midrule-actions, to make the grammar as independent of actions as possible. However, especially for possibly nested structures, like arrays or inline tables, they were unavoidable.

Chapter 3

Evaluation

3.1 Hardware Setup

The evaluation was executed on the following hardware setup:

```
Processor: Intel i7-4790K @ 4.20 GHz
RAM: 16 GB (2 x 4GB, 1 x 8GB), DDR-3 @ 1600 MHz
HDD: Seagate Desktop ST2000DM001, 2TB, 6 GB/s
```

3.2 Software Setup

The following versions of Elektra and LCDproc were used for evaluation:

```
Elektra:
repository: github.com/bauhaus93/libelektra.git
branch: 'plugin_toml'
commit: #e75d60f8699d8ea80f0f099364b3b8f03ec4413f
LCDproc:
repository: github.com/bauhaus93/lcdproc.git
branch: 'toml_test'
commit: #70bf70135190c270c6a75e0b975f591df323bb65
```

The evaluation was done within a docker container, which was built as follows.

```
cd libelektra
docker build -t buildElektra-buster \
    --build-arg JENKINS_USERID=‘id -u‘ \
    --build-arg JENKINS_GROUPID=‘id -g‘ \
    -f scripts/docker/debian/buster/Dockerfile \
    scripts/docker/debian/buster/
```

We extended this image by building another image on top of it. This dockerfile mainly installs/sets up sudo for installing Elektra and LCDproc. Dockerfile:

```
FROM buildelektra-buster

USER root
RUN apt-get update && apt-get -y install sudo vim tmux
RUN adduser jenkins sudo
RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
USER jenkins
WORKDIR /home/jenkins
```

Finally, a setup script get invoked. It installs Elektra and LCDdproc, and then copies LCDproc TOML files to the `./config` directory.

```
./lcdproc/toml_setup_install.sh
```

This script also invokes the `post-install.sh` script, which will mount the configuration scripts at the point where expected by LCDproc.

3.3 Methodology

In this evaluation, we will check four parts of LCDproc: The server daemon `LCDd` and three clients, namely `lcdproc`, `lcdexec`, `lcdvc`. To check, if the TOML plugin correctly reads/writes configuration, we will force a roundtrip of the file by changing a value within that file with the help of the Elektra command `kdb set`. When we call `kdb set`, the configuration first gets read from the file, the value to set is updated, and then the keys are written back into the file. After we roundtripped a file, we compare it with it's original file by the help of the `diff` command. The `diff` command is always invoked

with the `-w` switch, so that whitespace differences ignored. The TOML plugin currently preserves whitespace only before comments (tabs get converted into four spaces).

The TOML files copied during installation either differ from the default configuration loaded by the specification, or we will alter the keys of the file by a call of `kdb set` before we execute the program. During execution, we check if the program starts without any errors concerning the reading of the TOML file. We also check, if we can observe the expected difference from the default configuration, to ensure that we really are using the values from our configuration file.

3.4 LCDd

The LCDd server displays the stats sent to it be the client applications. Normally we would display that stats on a LCD device, but for our purposes, we configured it to use curses instead. Since the default configuration LCDd file is very minimal, and most of the configuration is loaded by the specification, we created a LCDd TOML file for evaluation purposes. It is based on the old default configuration file of LCDd [15]. We converted this old configuration from it's ini like format into TOML. All driver configuration settings, which are not referenced in the specification [16], were removed from it.

The resulting TOML file has 606 lines including 369 lines of comments and 143 empty lines. It contains one TOML table and a nested array. It contains multiple TOML table arrays - one for each driver.

We force a full read/write roundtrip of the configuration file, by setting the currently used driver. This driver is already the active driver in the file, but we just set it again for the roundtrip. Afterwards, we check if there are any differences between the original file and the newly written file.

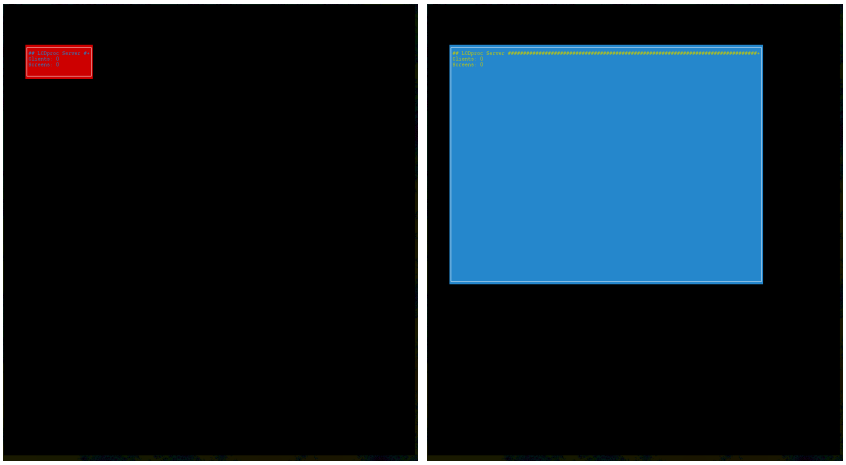
```
kdb set '/sw/lcdproc/lcdd/#0/current/server/drivers/#0' '@/curses/#0'
diff -ws ./lcdproc/LCDd.toml .config/LCDd.toml
Files ./lcdproc/LCDd.toml and .config/LCDd.toml are identical
```

After the roundtrip, the resulting LCDd.toml is identical to the original file under disregard of whitespace.

We start and stop the LCDd daemon once, change some parameters of the curses driver, and start the daemon again.

```
LCDd -f
kdb set '/sw/lcdproc/lcdd/#0/current/curses/#0/foreground' 'yellow'
kdb set '/sw/lcdproc/lcdd/#0/current/curses/#0/background' 'blue'
kdb set '/sw/lcdproc/lcdd/#0/current/curses/#0/backlight' 'blue'
kdb set '/sw/lcdproc/lcdd/#0/current/curses/#0/size' '100x40'
LCDd -f
```

The resulting output of LCDd is:



(a) Before parameter change (b) After parameter change

Figure 3.1: LCDd output

We observe, that we successfuly changed the foreground/background color and window size of the daemon.

3.5 lcdproc

The lcdproc client is the main client for LCDproc. It sends the stats of it's host to the connect LCDd server. The configuration file used for the evaluation of lcdproc contains has 156 lines, including 58 lines of comment and 46 empty lines. The lcdproc configuration gets roundtripped by changing the used port from the default value of 13666 to 12345. Afterwards the file difference without whitespace gets checked.

```
kdb set 'user/sw/lcdproc/lcdproc/#0/current/lcdproc/port' '12345'
diff -w ./lcdproc/lcdproc.toml .config/lcdproc.toml
```

```
9c9
< port=13666
---
> port = 12345
```

The only difference found is the line with the port number, for which we changed the value. The rest of the file remained unchanged.

Then, the lcdproc client gets started without an active LCDd-daemon, expeciting it to not start fully, since it can't connect to the server.

```
lcdproc -f
sock_connect: connect failed
Error connecting to LCD server localhost on port 12345.
Check to see that the server is running and operating normally.
```

In the error message, we see that the client tried to connect to port 12345, and not the default 13666.

3.6 lcdexec

With lcdexec, the LCDd server can execute preconfigured commands on the machine where lcdexec is running. We can add menus to the lcdexec configuration, which can contain further menus or the aforementioned commands, which can be any line of shell code. For the evaluation, e created a configuration file containing one menu with two options. The menu options execute `'echo a'` and `'echo b'`, respectively. The lcdexec configuration file has 125 lines, including 36 lines of comment and 25 empty lines. We first roundtrip the configuration file, by resetting the main menu key (which already contains the now set value). Afterwards, the file is checked without whitespaces.

```
kdb set 'user/sw/lcdproc/lcdexec/#0/current/menu/main' \
'@/menu/menu/#0'
diff -ws ./lcdproc/lcdexec.toml .config/lcdexec.toml
Files ./lcdproc/lcdexec.toml and .config/lcdexec.toml are identical
```

We see there are no differences between the original and the roundtripped configuration file.

In a tmux (Terminal Multiplexer) session, LCDd and lcdexec get executed simultaneously. In LCDd, we navigate into our configured lcdexec menu and execute the first option to execute `echo a`.

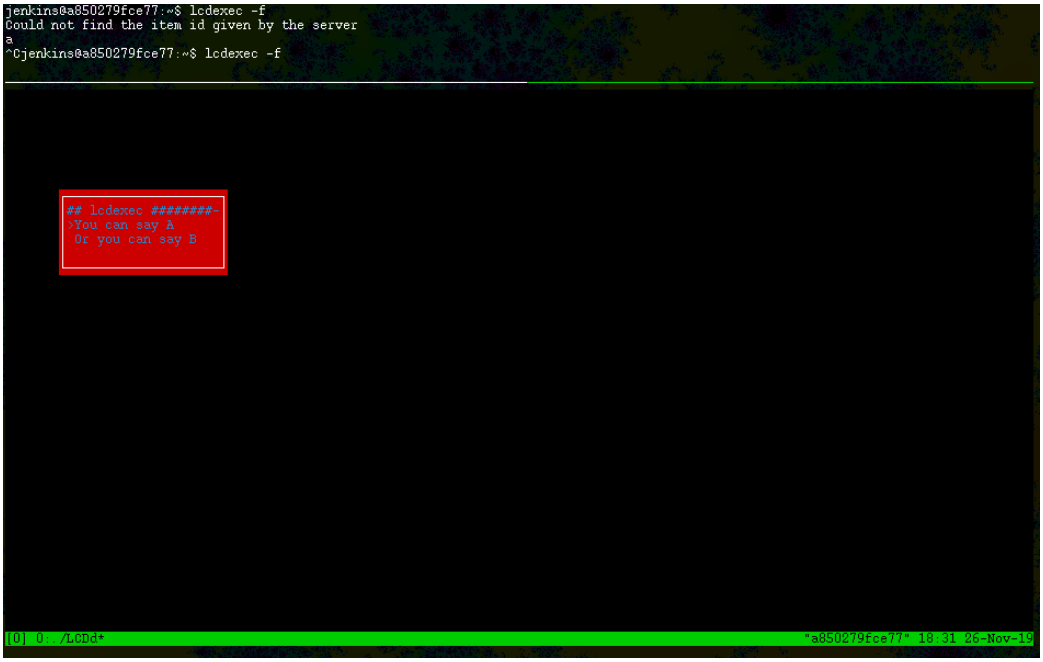


Figure 3.2: Menu navigation in LCDd while lcdexec is connected

We see both of our configured menus, and, on selecting the first menu option, we also see that `a` gets printed in the lcdexec window. We also see a warning message, that an item id could not be found, which gets printed, when the server has sent an unknown command.

3.7 lcdvc

The lcdvc client can display a system console on a display which is controlled by LCDd. Our configuration file only contains 2 assignments, the rest of the 33 lines consist of comments and empty lines. In the created TOML configuration we set the devices to read from to `/dev/urandom`.

```
kdb set user/sw/lcdproc/lcdvc/#0/current/lcdvc/vcsdevice /dev/urandom
kdb set user/sw/lcdproc/lcdvc/#0/current/lcdvc/vcsadevice /dev/urandom
diff -ws lcdproc/lcdvc.toml .config/lcdvc.toml
31,32c31,32
< vcsdevice="/dev/null"
< vcsadevice="/dev/null"
```

```
---
> vcsdevice = "/dev/urandom"
> vcsadevice = "/dev/urandom"
```

We see, that the only file differences are the values we changed by `kdb set` before.

When we start LCDd and lcdexec in a tmux session, we see that lcdexec forwards the stream of `/dev/urandom` to LCDd.

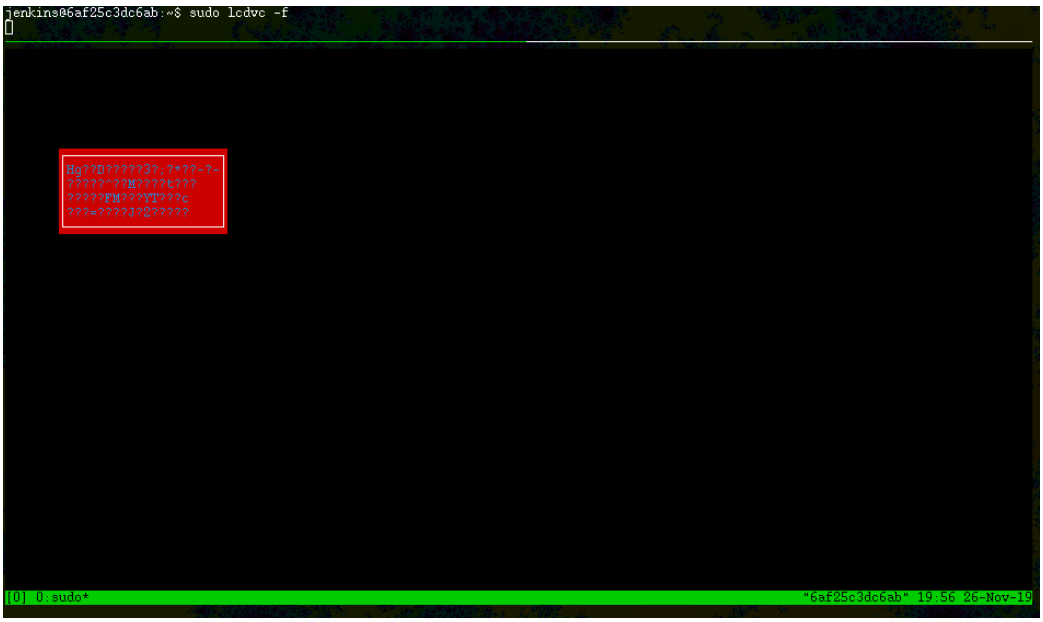


Figure 3.3: LCDd receiving `/dev/random` from lcdexec

3.8 Benchmarks

3.9 Discussion (or place it into Conclusion?)

We could observe, that in all four programs of LCDproc, we would get identical files after read/writing them, when ignoring whitespace. This also means, that no comments got lost or misordered during roundtripping a file. We tested not only basic key/value assignments, but also TOML features like

tables and table arrays, and even nested arrays, which all could be restored correctly. Only during lcdexec evaluation we got a warning for a missing menu id, but since our configured command got executed anyway, we assumed success. There is problably an oversight on our side, and we didn't create a file conforming to the specification perfectly. (TODO: Can I write that?)

Chapter 4

Related Work

Chapter 5

Conclusion

Chapter 6

Glossary

Acronyms

Flex Fast Lexical Analyzer. 3

LCD Liquid Crystal Display. 2, 7

LR Left-to-Right, Rightmost derivation. 4

POSIX Portable Operating System Interface. 3

tmux Terminal Multiplexer. 9, 11

TOML Tom's Obvious, Minimal Language. 1, 2, 6, 7, 10, 11

Bibliography

[1] Toml git contributions. https://github.com/toml-lang/toml/graphs/contributors. Accessed 2019-10-23.

[2] Toml readme. https://github.com/toml-lang/toml/blob/master/README.md. Accessed 2019-10-23.

[3] Toml wiki. https://github.com/toml-lang/toml/wiki. Accessed 2019-10-23.

[4] Cargo github. https://github.com/rust-lang/cargo/. Accessed 2019-10-23.

[5] Pip reference guide. https://pip.pypa.io/en/stable/reference/pip/. Accessed 2019-10-23.

[6] libelektra. https://www.libelektra.org/home. Accessed 2019-10-23.

[7] Elektra bindings. https://www.libelektra.org/bindings/readme. Accessed 2019-10-23.

[8] Elektra storage plugins. https://www.libelektra.org/plugins/readme#storage. Accessed 2019-10-23.

[9] Lcdproc website. http://lcdproc.omnipotent.net. Accessed 2019-11-19.

[10] Lcdproc github. https://github.com/lcdproc/lcdproc. Accessed 2019-11-19.

[11] Flex github. https://github.com/westes/flex. Accessed 2019-11-19.

[12] Bison website. https://www.gnu.org/software/bison. Accessed 2019-11-19.

[13] Elektra yambi plugin. https://github.com/ElektraInitiative/libelektra/tree/master/src/plugins/yambi. Accessed 2019-11-19.

[14] Bison midrule conflicts. https://www.gnu.org/software/bison/manual/html_node/Midrule-Conflicts.html. Accessed 2019-11-19.

[15] Lcdproc.conf. https://github.com/lcdproc/lcdproc/blob/master/LCDd.conf. Commit #d68c7e4da03362795edef12373cc80b3e553e29c.

[16] Lcdd spec. https://github.com/bauhaus93/lcdproc/blob/master/server/specification/LCDd-spec.ini. Commit #3e2b9d3f5cade3bbe21fdb5a9ea1c9ffc1c994fa.