L04 Configuration Sources

Markus Raab

Institute of Information Systems Engineering, TU Wien

14.04.2021

Configuration File Formats

1. Configuration File Formats
2. Command-line Arguments
3. Environment Variables
4. Abstractions
5. Complexity
   - Trend
   - Calculation
6. Meeting

Learning Outcomes

Students will be able to

- differentiate between configuration sources
- unify configuration sources via specifications
- (calculate complexity of configuration settings)

Definition

A ***configuration file*** is a file containing configuration settings.

Definition

A ***configuration file*** is a file containing configuration settings.
A Web server configuration file:

```
port=80␣;␣comment
address=127.0.0.1
```

Question

What are keys? What are configuration values? What is metadata?

Definition

A ***configuration file*** is a file containing configuration settings.
A Web server configuration file:

```
port=80␣;␣comment
address=127.0.0.1
```

The configuration values are 80 and `127.0.0.1`, respectively.
Other information in the configuration file is metadata for the configuration settings (such as the comment).

Configuration File Formats

- CSV (comma-separated values)
- semi-structured
- programming language
- literate

CSV formats

- passwd: 3$^{\text{rd}}$ November, 1971
- passwd and group use : as separator
- are difficult to extend (e.g., GECOS)
- today mostly used for legacy reasons
- are replaced one-by-one (e.g., inetd, crontab)

Programming Language

- + trivial for developers (source the file)
- + above-overage quality of error message
- − makes automatic change of individual values harder
- − very hard to use for people who do not know the programming language
- − does not separate code and data

Trends

- away from CSV
- towards general-purpose serialization formats (INI, JSON)
- human-read/writable (YAML, TOML)
- programming language as configuration file

Method

What do FLOSS developers say?

Q: survey with 672 persons visiting, 162 persons completing the survey [4]

S: source code analysis of 16 applications, comprising 50 million lines of code [4]

Why are so many formats present?

Q: "In which way have you used or contributed to the configuration system/library/API in your previously mentioned FLOSS project(s)?" [4]

- 19 % persons ($n = 251$) have introduced a configuration file format.
- 29 % implemented a configuration file parser.
- 15 % introduced a configuration system/library/API.
- 34 % used external configuration access APIs.

Multitude of Formats

- on every system a multitude of (legacy) configuration file formats exist
- the number grows fast
- thus applications usually have to deal with some legacy formats

Requirement

A configuration library must be able to integrate (legacy) systems and must fully support (legacy) configuration files.

L04 Configuration Sources

Markus Raab

Institute of Information Systems Engineering, TU Wien

14.04.2021

Command-line Arguments

1 Configuration File Formats

2 Command-line Arguments

3 Environment Variables

4 Abstractions

5 Complexity
- Trend
- Calculation

6 Meeting

Is there something else?

- configuration files are the most researched of all configuration sources [2]
- but it is neither the most used nor most popular [4]

Q: "Which configuration systems/libraries/APIs have you already used or would like to use in one of your FLOSS project(s)?"

- command-line arguments (92 %, $n = 222$)
- environment variables (79 %, $n = 218$)
- *S:* API `getenv` is used omnipresently with 2,683 occurrences
- configuration files (74 %, $n = 218$))

Q: "What is your experience with the following configuration systems/libraries/APIs?"

- `getenv` (10 %, $n = 198$)
- configuration files (6 %, $n = 190$)
- command-line options (4 %, $n = 210$)
- X/Q/GSettings (41 %, 14 %, 35 %)
- KConfig (21 %)
- dconf (42 %)
- plist (32 %)
- Windows Registry (69 %)

Semantics

- passed by main for a new process via
  `(int argc, char ** argv)`
- visible from other processes (e.g., via `ps aux`)
- could be passed along to subprocesses but hardly done
- need to be parsed by process
- portability: differences in parsing
- cannot be changed from outside (requires restart, no IPC)

L04 Configuration Sources

Markus Raab

Institute of Information Systems Engineering, TU Wien

14.04.2021

Environment Variables

1. Configuration File Formats
2. Command-line Arguments
3. Environment Variables
4. Abstractions
5. Complexity
    - Trend
    - Calculation
6. Meeting

Usage

1. bypassing other configuration accesses (*Q:* 45 %)
2. locating configuration files
3. debugging and testing (*Q:* 55 %, *S:* 1,152, i. e. 43 %)
4. sharing configuration settings across applications (*Q:* 53 %, *S:* 716, i. e. 47 %)
5. for configuration settings unlikely to be changed by a user (*Q:* 20 %)
6. *"even when it is used inside a loop"* (*Q:* 2 %)

Semantics

- are also per-process (`/proc/self/environ`)
- are not visible from other processes
- are automatically inherited by subprocesses
- need to be parsed by process (`[extern] char **environ`) but API is provided (`getenv`)
- cannot be changed from outside (requires restart or an additional IPC mechanism)

getenv

- is widely standardized, including SVr4, POSIX.1-2001, 4.3BSD, C89, C99 [1],
- is supported by many programming languages, and
- enforces `key=value` convention.

Portability

- no separators for values defined
- case sensitivity problems
- often many environment variables for the same purpose: TMP, TEMP, or TMPDIR
- sometimes one environment variable for different purposes: PATH

L04 Configuration Sources

Markus Raab

Institute of Information Systems Engineering, TU Wien

14.04.2021

Abstractions

1. Configuration File Formats
2. Command-line Arguments
3. Environment Variables
4. Abstractions
5. Complexity
   - Trend
   - Calculation
6. Meeting

User View

- command-line for trying out configuration settings
- environment variables for configuration settings within a shell
- configuration files for persistent configuration settings

Abstraction

Requirement

A configuration library must be able to integrate (legacy) systems and must fully support (legacy) configuration files.

How can we deal with the many formats?

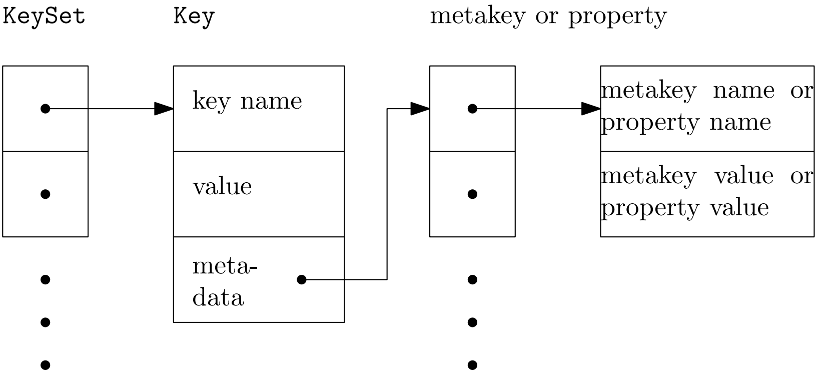
Key-Value

A key-value pair is the simplest generic data structure [7]. While all these formats above have many differences, all of them represent configuration settings as ***key-value pairs*** [2, 3, 6, 8].

For configuration as program you need to execute them first.

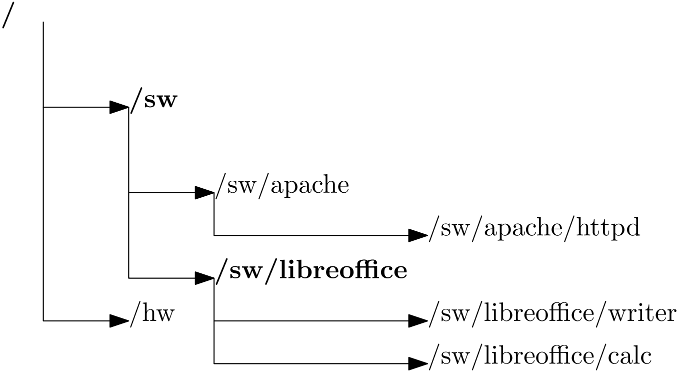
KeySet (Recapitulation)

The common data structure between plugins:


KeySet
Key
metakey or property
key name
value
meta-data
metakey name or property name
metakey value or property value

Mounting

Mounting integrates a backend into the key database [5]. Hence, Elektra allows several backends to deal with configuration files at the same time. Each backend is responsible for its own subtree of the key database.



Elektra

```
[kdb/printversion]
description = "print version information"
opt = v
opt/long = version
opt/arg = none
```

- gopts puts Keys in the proc namespace
- https://www.libelektra.org/tutorials/command-line-options

```
kdb -v    kdb --version    VERSION=1 kdb
```

How can we deal with the many sources?

Requirement

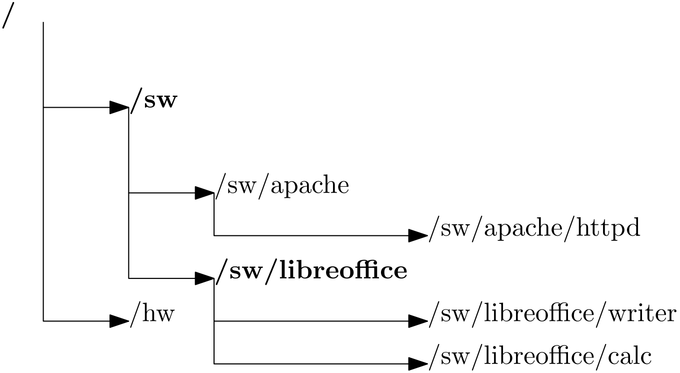
A configuration library must support all three popular ways for configuration access: configuration files, command-line options, and environment variables.

Plugins

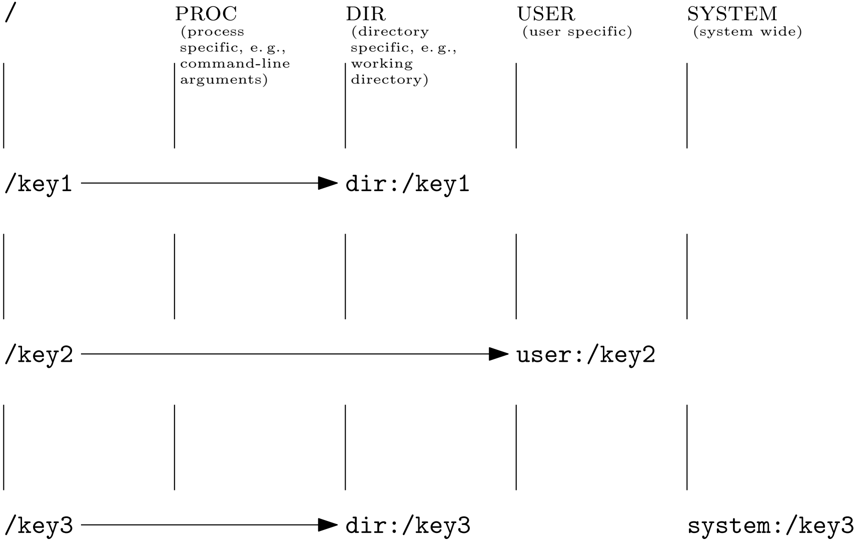
Different backends can use different plugins:

`/sw` in the INI file config.ini

`/sw/libreoffice` in the XML file libreoffice.xml



Cascading



/

PROC
(process specific, e. g., command-line arguments)

DIR
(directory specific, e. g., working directory)

USER
(user specific)

SYSTEM
(system wide)

/key1 → dir:/key1

/key2 → user:/key2

/key3 → dir:/key3

system:/key3

Conclusion

- three different configuration sources widely used
- all three used for different reasons but often for the same configuration settings
- many different configuration file formats
- abstractions: key-value, mounting, and cascading

L04 Configuration Sources

Markus Raab

Institute of Information Systems Engineering, TU Wien

14.04.2021

Complexity

1 Configuration File Formats

2 Command-line Arguments

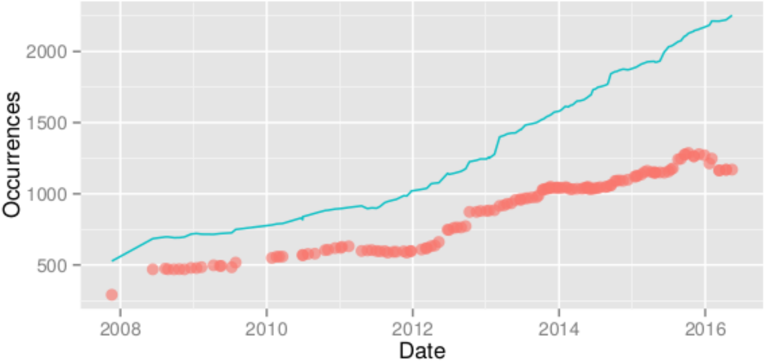
3 Environment Variables

4 Abstractions

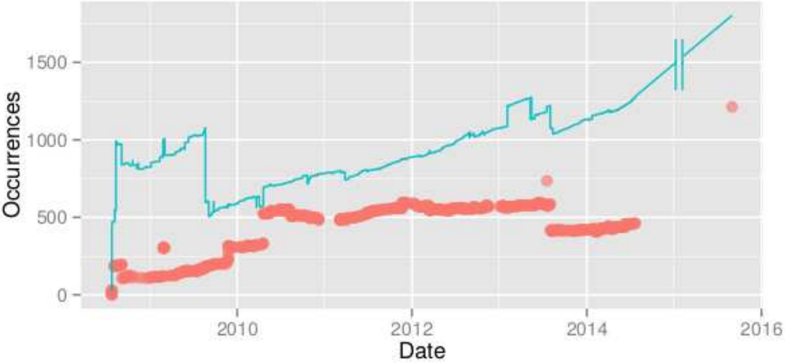
5 Complexity
- Trend
- Calculation

6 Meeting

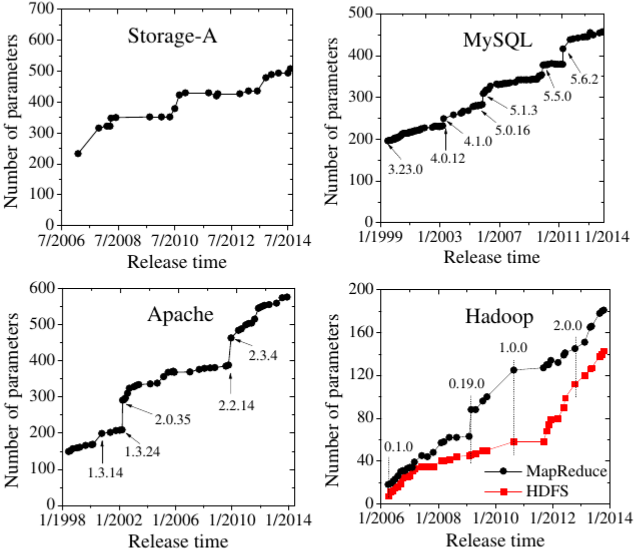
Trend Firefox



Trend Chromium



Trend Configuration Files



Xu et al. [9]

Types of Complexity

- complexity in access:
  - many different formats
  - non-uniformity
  - transformations
- configuration settings
  - number of settings s
  - number of values n
  - dependences between settings

Calculation of Complexity

Using enumerative combinatorics:

- number of configurations: n^s
- for N groups of different n and s (i.e., $n_1 \ldots n_N$ with $s_1 \ldots s_N$ occurrences):

$$\prod_{i=1}^{N} n_i^{s_i}$$

- more difficult to calculate (or unbounded) for dependences, module instantiations, arrays, . . .

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd (let us assume $n = 2$):

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd (let us assume $n = 2$): $2^{600} \approx 10^{180}$
- 19 integer settings:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd (let us assume $n = 2$): $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- for 20 boolean and 20 enums with 5 possibilities:

Calculation of Complexity

Examples:

- 600 boolean settings in Apache httpd (let us assume $n = 2$): $2^{600} \approx 10^{180}$
- 19 integer settings: $2^{32^{19}} = 2^{32 \cdot 19} = 2^{609} \approx 10^{183}$
- for 20 boolean and 20 enums with 5 possibilities:

$$2^{20} * 5^{20} = 10^{20}$$

Calculation of Complexity (cont.)

Examples:

- an array with $1 - 20$ boolean settings:

[1] `https://downloads.mysql.com/docs/refman-5.7-en.pdf`

Calculation of Complexity (cont.)

Examples:

- an array with $1 - 20$ boolean settings: 2^{20}
- MySQL has 461 settings, of which 216 are non-simple types [9]
  (let us assume $n = \{3, 20\}$):

[1] https://downloads.mysql.com/docs/refman-5.7-en.pdf

Calculation of Complexity (cont.)

Examples:

- an array with $1 - 20$ boolean settings: 2^{20}
- MySQL has 461 settings, of which 216 are non-simple types [9]
  (let us assume $n = \{3, 20\}$): $3^{245} * 20^{216} \approx 10^{397}$
  (settings are explained in 5560 pages[1])

[1] `https://downloads.mysql.com/docs/refman-5.7-en.pdf`

Calculation of Complexity (cont.) [2]

Examples:

- in Firefox resulting in 846 boolean options and 1,111 options of either integer or string, each with three values

Calculation of Complexity (cont.) [2]

Examples:

- in Firefox resulting in 846 boolean options and 1,111 options of either integer or string, each with three values

$$2^{846} * 3^{1111} \approx 6.46 * 10^{259}$$

- LibreOffice

Calculation of Complexity (cont.) [2]

Examples:

- in Firefox resulting in 846 boolean options and 1,111 options of either integer or string, each with three values

$$2^{846} * 3^{1111} \approx 6.46 * 10^{259}$$

- LibreOffice

$$2^{4433} * 3^{31889}$$

L04 Configuration Sources

Markus Raab

Institute of Information Systems Engineering, TU Wien

14.04.2021

Meeting

1. Configuration File Formats
2. Command-line Arguments
3. Environment Variables
4. Abstractions
5. Complexity
    - Trend
    - Calculation
6. Meeting

[1] *getenv(3) Linux User's Manual*, March 2017.

[2] Dongpu Jin, Xiao Qu, Myra B. Cohen, and Brian Robinson. Configurations everywhere: Implications for testing and debugging in practice. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 215–224, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2768-8. doi: 10.1145/2591062.2591191. URL http://dx.doi.org/10.1145/2591062.2591191.

[3] Neal Lathia, Kiran Rachuri, Cecilia Mascolo, and George Roussos. Open source smartphone libraries for computational social science. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, pages 911–920, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2215-7. doi:

10.1145/2494091.2497345. URL http://dx.doi.org/10.1145/2494091.2497345.

[4] Markus Raab and Gergö Barany. *Challenges in Validating FLOSS Configuration*, pages 101–114. Springer International Publishing, Cham, 2017. ISBN 978-3-319-57735-7. doi: 10.1007/978-3-319-57735-7_11. URL http://dx.doi.org/10.1007/978-3-319-57735-7_11.

[5] Markus Raab and Patrick Sabin. Implementation of Multiple Key Databases for Shared Configuration. ftp://www.markus-raab.org/elektra.pdf, March 2008. Accessed February 2014.

[6] Ariel Rabkin and Randy Katz. Static extraction of program configuration options. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 131–140. IEEE, 2011.

[7] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004*, September 2004. URL `http://elib.dlr.de/7444/`.

[8] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.

[9] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs! Understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 307–319, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3675-8. doi:

10.1145/2786805.2786852. URL
http://dx.doi.org/10.1145/2786805.2786852.