# the wiz book

Torben Schinke

Version 1.0, 2017-12-05

# Table of Contents

# Dedication

For the family.

# Contents

# Preface

The idea of a robust, simple and scalable storage format superseeding the lowest denomiator filesystems, fascinated me already 15 years ago, however I never had the opportunity to actually start implementing such a project. When the time came, I started to design a paper based specification in 2015 which performs well for deduplicating large files, nested directory trees and continues snapshots. To solve the typical problems of a *multi file based document format* at work, I created a proprietary java based implementation from it, called wiz - which is just the opposite of a git, similarities are purely coincidental. For the original intention, it worked pretty well. But as requirements changed, the performance for a lot of additional use cases was disappointing. The main performance issues are caused by both, inherent format decisions and the necessity of a complex virtual machine. In practice, the latter caused also penalties on the probably most successful mobile platform of our time. To solve all of these issues I started to design an entirely new specification which addresses all of the new additional scenarios (and even more). Hereafter this new specification is actually *wiz version 3* or simply *wiz*. Therefore the proprietary existing wiz implementation is called *legacy wiz* and is not only implemented in a different language but also does a lot of things differently to improve performance, storage usage, reliability and system complexity. Today, the market for closed source commercial software libraries is nearly dead and gaining money or finding acceptance is not easy. Usually large companies dominate the market with a lot (but definity not all) high quality products.

# Format specification

Wiz is both, an implementation and a specification. In this chapter only the specification matters and is described in a way that it can be implemented in any language or ecosystem.

A node always starts with a byte identifier and is otherwise undefined. Most lengths uses a varuint so that it has an adaptive overhead which increases dynamically as the payload size increases. Overhead also depends on the used compression algorithm, if a node supports that at all. The payload of a node should not exceed something reasonable, e.g. ZFS (see also [zfs-spec]) uses 128KiB but you may even go into the range of MiB to increase efficiency. The UTF8 type is always prefixed with a varuint length to increase efficiency for short strings but allowing also more than the typical 64k bytes. All numbers are treated as big endian to match network byte order. As a side note, even if most operating systems are little endian today, one cannot ignore BE systems, so any code must be endian independent anyway. It is not expected that wiz may profit from a system specific endianess, like ZFS does.

*ditaa Diagram Example*

```
+--------+   +-------+    +-------+
|        |---+ ditaa +--->|       |
|  Text  |   +-------+    |diagram|
|Document|   |!magic!|    |       |
|     {d}|   |       |    |       |
+---+----+   +-------+    +-------+
    :                         ^
    |       Lots of work      |
    +-------------------------+
```

Chapters can contain sub-sections nested up to three deep. [1: An example footnote.]

Chapters can have their own bibliography, glossary and index.

And now for something completely different: monkeys, lions and tigers (Bengal and Siberian) using the alternative syntax index entries.  Note that multi-entry terms generate separate index entries.

Here are a couple of image examples: an [smallnew] example inline image followed by an example block image:

[Tiger image] | *images/tiger.png*

*Figure 1. Tiger block image*

Followed by an example table:

*Table 1. An example table*

| Option | Description |
|---|---|
| -a *USER GROUP* | Add *USER* to *GROUP*. |
| -R *GROUP* | Disables access to *GROUP*. |

*Example 1. An example example*

> Lorum ipum…

# Sub-section with Anchor

Sub-section at level 2.

## Chapter Sub-section

Sub-section at level 3.

### Chapter Sub-section

Sub-section at level 4.

This is the maximum sub-section depth supported by the distributed AsciiDoc configuration. [2: A second example footnote.]

# The Second Chapter

An example link to anchor at start of the first sub-section.

An example link to a bibliography entry [taoup].

# The Third Chapter

Book chapters are at level 1 and can contain sub-sections.

# Appendix A: Example Appendix

One or more optional appendixes go here at section level 1.

## Appendix Sub-section

Sub-section body.

# Example Bibliography

The bibliography list is a style of AsciiDoc bulleted list.

*Books*


- [taoup] Eric Steven Raymond. *The Art of Unix Programming*. Addison-Wesley. ISBN 0-13-142901-9.
- [walsh-muellner] Norman Walsh & Leonard Muellner. *DocBook - The Definitive Guide*. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.
- [zfs-spec] http://www.giis.co.in/Zfs_ondiskformat.pdf


*Articles*


- [abc2003] Gall Anonim. *An article*, Whatever. 2003.

# Example Glossary

Glossaries are optional. Glossaries entries are an example of a style of AsciiDoc labeled lists.

**A glossary term**

The corresponding (indented) definition.

**A second glossary term**

The corresponding (indented) definition.

# Example Colophon

Text at the end of a book describing facts about its production.

# Example Index