

(empty left first page)

# LibreOffice & ZML plain source text internals

Dr. Hartmut Schorrig

[www.vishia.org](http://www.vishia.org)

2024-09-30

LibreOffice odt content is held parallel and also editable and convertible in a plain text, the Format is named ZmL (Z markup Language). Also working with AsciiDoc is supported.

This document shows internal implementation concepts and hints.

It is in the moment not complete.

## Table of Contents

1 Internals.....	4
------------------	---



## 1.2 ReadOdt

todo

## 1.3 Write content.xml to the odt file from internal data

todo

## 1.4 WriteOdt

The main source for the writer can be found in [org.vishia.odt.readOdt.WriteOdt \(www\)](#). It contains the [WriteOdt.main\(...\)](#) ([www](#)) to start from command line. Parsing all command line arguments is done with the class [org.vishia.util.Arguments](#) ([www](#)) from the used base library vishiaBase, The main calls [WriteOdt.smain\(...\)](#) ([www](#)) and then [WriteOdt.ainmain\(...\)](#) ([www](#)) and then with already outside prepared arguments to support calling from a superior tool (for example a GUI). The `execute(...)` does the work.

The [WriteOdt.execute\(...\)](#) ([www](#)) reads firstly the content from a maybe given `-cfg:file` for some settings.

The `-odt:file.odt` should be first opened as zip file, to read out its given style.xml for checks. This should be done in Version-2, yet not.

The `-oxml:content.xml` is opened for writing. If this argument is not given, it is supplemented by a `content.xml` file either `-in dbgDir:dir` or in the `-i:input` directory. The [org.vishia.xmlSimple.XmlSequWriter](#) ([www](#)) is used for writing XML. This is a simple sequential writer which does not build a tree of XML data, instead writing as coming. Hence the order of elements for output is well proper.

First the name space and head information are written to the `content.xml`. Then [WriteOdt.parseAdocWriteOdt\(...\)](#) ([www](#)) is

called which does the internal work. Then XML writing is finished and at last `content.xml` in the zip file `-odt:file.odt` is replaced.

[WriteOdt.parseAdocWriteOdt\(...\)](#) ([www](#)) works in the following kind:

It reads line per line the textual input file `-ifile.vml.adoc`. For any line [parseAdocM\(...\)](#) ([www](#)) is called. This checks the beginning of each trimmed line (left spaces are ignored). Note that lines of the elements before which continues the text are processed already, it means this operation sees the start of a new item of the text. This line starts are:

- `* ?` [parseList\(...\)](#) ([www](#))
- else, if the line does not start with `*` ([www](#)) , then a currently list is closed.
- `= ?` [writeHeaderLine\(...\)](#) ([www](#)) : The line should contain with `=== ChapterTitle <: #Label>`
- `### ?` Line is ignored, it's a comment.

Note, this are the only one simple character on start of a line which triggers. All other are the `<:xxx` Designation.

- `<:p: ?` [parseWriteParagrStyleLabel\(...\)](#) ([www](#)) The paragraph starts with a style definition in form `<:p:style.>`. The following text and following text lines builds the text of the paragraph. See also [parseWriteText\(...\)](#) ([www](#))

- `<:Code:` ? [parseWriteCodeBlock\(...\)](#) ([www](#)) and the text till `<.code>` is parsed and translated.
- `<:table` ? [parseWriteTable\(...\)](#) ([www](#))
- `<:pageBreak.>` ? It sets only the flag [bPageBreakBefore](#) ([www](#)) , recognized for the next style as modification (chapter title, paragraph).
- `<:columnBreak.>` ? It sets adequate only the flag [bColumnBreakBefore](#) ([www](#))
- `<:Section:` ? [parseWriteSection\(...\)](#) ([www](#))
- `<.Section>:` [writeSectionEnd\(...\)](#) ([www](#))
- `<:TOC:` [writeTableOfContents\(...\)](#) ([www](#))
- else ? [parseWriteParagr\(...\)](#) ([www](#)) If non of this Designations met, the following text is a standard paragraph.

### **Docu file: LibreOffcZMarkup**

- 1 Approaches page 4 (#approach)
  - 1.1 LibreOffice beside the plain text of content page 5 (#Test2Label)
  - 1.2 Why another markup format instead and beside AsciiDoc? page 6 (#Appr-whyZmL)
  - 1.3 Using only indirect styles page 7 (#Appr-onlyIndirectStyles)
  - 1.4 What you see is what you have page 8 (#appr-wysiwyh)
  - 1.5 Working in the document in LibreOffice and similar in ZmL page 8 (#appr-editLOffcZmL)
  - 1.6 Software docu including generated docu and source code page 9 (#\$Label\_1)
- 2 Some decisions how to write a technical documentation page 10 (#WrDocu)
  - 2.2 Writing style in columns for each (sub) chapter page 11 (#WrDocu-Columns)
  - 2.3 Manual column or page breaks and positions of images page 11 (#WrDocu-PgImage)
    - 2.3.1 Page breaks and reserve space on page end page 11 (#WrDocu-PgImage-PgReserve)
    - 2.3.2 How to insert a page or column break page 11 (#WrDocu-PgImage-break)
    - 2.3.3 Position of images page 13 (#WrDocu-PgImage-PosImg)
  - 2.4 Using a real small set of format styles and less direct formatting page 14 (#WrDocu-LessStyles)
    - 2.4.1 Is a free styled document design proper? page 14 (#WrDocu-LessStyles-FreeStyles)
    - 2.4.2 List appearances page 14 (#WrDocu-LessStyles-List)
    - 2.4.3 Code snippets also possible to include from sources page 15 (#WrDocu-LessStyles-Code)
    - 2.4.4 Character styles page 16 (#WrDocu-LessStyles-Char)
  - 2.5 Character set and special characters page 17 (#WrDocu-SpecChars)
  - 2.6 Internal links, bookmarks page 18 (#WrDocu-Refs)
  - 2.7 External links to Javadoc local files and the internet page 19 (#WrDocu-HrefSw)
    - 2.7.1 Software documentation page 19 (#WrDocu-HrefSw-SwDoc)
    - 2.7.2 Relative local links and supplement www link with same path page 19 (#WrDocu-HrefSw-wwwSuppl)
    - 2.7.3 Supplement argument types of intern operation links (anchors in html) page 20 (#WrDocu-HrefSw-OpArgSuppl)
  - 2.8 Exchange and maintain the styles of the document page 21 (#WrDocu-editStyles)
- 3 Z markup Language page 23 (#ZmL)
  - 3.1 Basic Considerations page 23 (#ZmL-Base)

- 3.1.1 Plain source text page 23 (#ZmL-Base-PlainText)
- 3.1.2 Comment lines page 24 (#ZmL-Base-Comment)
- 3.1.3 Section, chapter and paragraph structure near AsciiDoc page 24 (#ZmL-Base-ChaptStruct)
- 3.1.4 Text structure (syntax) similar AsciiDoc but other designations page 24 (#ZmL-Base-Syntax)
- 3.10 Cross references inside the document, how to deal with interrelated documents page 35 (#ZmL-ChRef)
  - 3.10.1 Use a proper name for bookmark labels page 35 (#ZmL-NameBookmark)
  - 3.10.2 How to write a reference page 35 (#ZmL-ChRef-Syntay)
  - 3.10.3 Cross reference to other documents of the same suite page 36 (#ZmL-ChRef-ext)
- 3.11.2 Relative local links and supplement www link with same path page 38 (#ZmL-Href-wwwSuppl)
- 3.11.3 Supplement argument types of intern operation links (anchors in html) page 38 (#ZmL-Href-OpArgsSuppl)
- 3.12 Transliteration of specific characters page 40 (#ZmL-TranscrChars)
- 3.13 Using Character styles, semantic text span page 41 (#ZmL-cStyle)
- 3.2 Syntax overview by examples page 25 (#ZmL-SyntaxOverview)
  - 3.2.1 Third level chapter page 26 (#Label1)
- 3.3 Chapter designation and content page 27 (#ZmL-chapter)
- 3.4 Writing style of paragraphs page 27 (#ZmL-paragr)
- 3.5 Sections page 27 (#ZmL-section)
- 3.6 Lists page 27 (#ZmL-list)
- 3.7 Code snippets page 28 (#ZmL-code)
  - 3.7.1 Syntax and styles page 28 (#\$Label\_2)
  - 3.7.2 Shortened code lines page 28 (#\$Label\_3)
  - 3.7.3 Lines can have character styles page 28 (#\$Label\_4)
  - 3.7.4 Code lines can contain special characters page 28 (#\$Label\_5)
  - 3.7.5 Include of code snippets from sources page 29 (#ZmL-code-fromSrc)
  - 3.7.6 Include code snippets with labels and off/on in include line, Syntax of the code include line page 30 (#\$Label\_6)
- 3.8 Images page 31 (#ZmL-Image)
  - 3.8.1 Some remarks to size of images page 32 (#ZmL-ImgSize)
- 3.9 Possibility of include and dispersion page 34 (#ZmL-include)
- 4 Handling Writing and Converting page 42 (#Handling)
  - 4.1 Directory tree structure in the working area page 42 (#Handling-Dirtree)
  - 4.2 Daily work on the documents page 43 (#Handling-Dwork)
  - 4.3 Convert from given vml to a new odt page 44 (#Handling-Genvml2odt)
  - 4.4 Convert from LibreOffice odt to a vml page 45 (#Handling-GenOdt2ZmL)
- 5 Implementation page 46 (#impl)
  - 5.1 Explanation of the scripts to call the converter page 46 (#Impl-call)
    - 5.1.1 Script to clean and create a build (temporary output) location page 46 (#Impl-call-cleanBuild)
    - 5.1.2 callZmL2odt.bat and callZmL2odt.sh page 48 (#Impl-call-ZmL2odt)
    - 5.1.3 Macro =>ZmL and script -callOdt2ZmL.bat and .callOdt2ZmL.sh page 50 (#Impl-call-Odt2ZmL)
  - 5.2 Get the tools page 52 (#Impl-tools)
  - 5.3 Test ref page 52 (#\$Label\_7)