(empty left first page)

LibreOffice & ZML plain source text internals

Dr. Hartmut Schorrig
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

1.1 Read content.xml from the odt file to internal data

The org.vishia.xmlReader.XmlJzReader (www) contained in the `vishiaBase.jar` is used to read the XML data. This class can select determined parts from the XML file, read not all. Therefore a configuration file is used. The data are stored in a common or a specific data class. For the `content.xml` of the odt the common class org.vishia.xmlReader.XmlDataNode (www) is used. It reads all data in a tree representation.

The class to read the XML file is org.vishia.xmlReader.XmlJzReader (www) in the `vishiaBase.jar` file. The operation readZipXml(zipfile, "content.xml", data) (www) reads the “content.xml” from the given odt file which is a zip file.

The following code snippet shows how the `XmlJzReader` is invoked:

```
include:../java/org/vishia/odt/readOdt/
ReadOdt.java::readXml::45
```

```
 /**Reads completely the content.xml from...
  * and stores the data in the data insta...
  * @param data The root node for all XML...
  * @param fInOdt the odt file is a zip file
  * @return null or an error message
  * @throws IOException for file operation
  */
 private String readXml (XmlDataNode data...
   String sFileOdt = fInOdt.getName();
   XmlJzReader xmlRd = new XmlJzReader();
   xmlRd.setNamespaceEntry("xml", "XML");...
   xmlRd.setCfg(this.xmlCfgOdt);
   //xmlReader.setDebugStopTag("text:span");
   xmlRd.openXmlTestOut( new File(this.cm...
   String error = xmlRd.readZipXml(...<data>
              fInOdt, "content.xml", data );
   return error;
 } //
```

The `<data>` are of type `XmlDataNode`. The given instance stores the root node, and subsequently all children nodes. There are evaluated by the following algorithm.

The environment of call is in ReadOdt.execute():

```
include:../java/org/vishia/odt/readOdt/ReadOdt.java::callReadXml::92
```

```
    if(this.cmdArgs.fIn !=null) {  //======================== read the given odt file
      XmlDataNode data = new XmlDataNode(null, "root", null); ...<new root>
      String error = readXml(data, this.cmdArgs.fIn); ...<readXml>
      if(error !=null) {
        this.log.writeError("\nERROR reading xml file %s: %s", this.cmdArgs.fIn.getAbsolu...
        err = 4;
      } else {
        writeBackupFile(this.cmdArgs);
        this.wra = this.cmdArgs.fAdoc == null ? null :new OutputStreamWriter(new FileOutp...
        this.wr = new java.io.OutputStreamWriter(new FileOutputStream(this.cmdArgs.fMarku...
        this.wrRep = this.cmdArgs.fReport == null ? null :new OutputStreamWriter(new File...
        //this.wr = new java.io.FileWriter(this.cmdArgs.fOut);
        //======>>>>                   =====================>> convert to Asciidoc
        try {
          readOdgxmlWriteZmL(data);  ...<prc>
          wrClose();
          writeLabelRef();
          System.out.printf("\n*** finished %s @%s", this.cmdArgs.fMarkup.getName(), this...
          err = 0;         //
```

This shows the top level after the `main()`. The Instance for the root node is created and referenced only locally `<new root>`,. filled from the `<readXml>`, and processed in the line `<prc>`. You can here see also behavior on errors.

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.amain(...) (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