Budapest University of Technology and Economics
Department of Measurement and Information Systems

RTF tokenizer for LibreOffice writerfilter (rewrite of the RTF import)

Vajna Miklós (AYU9RZ),
4th semester, (MSc) computer engineering student
Consultants: Cédric Bosdonnat, Máté András, Novell
Internal consultant: Horváth Ákos, MIT
Specialization in Dependable System Design

Internship Report
2011/12. 1st semester

Contents

1 Introduction

LibreOffice is a power-packed free, libre and open source personal productivity suite for Windows, Macintosh and GNU/Linux, that gives you six feature-rich applications for all your document production and data processing needs: Writer, Calc, Impress, Draw, Math and Base[1].

During my internship I worked on Writer. Writer has support for lossless import and export from/to the ODF file format (.odt). Additionally, several other filters are supported which are marked as “alien”: such filters may lose some information upon loading/saving. Such an alien RTF[2] filter was already implemented in LibreOffice 3.4, and my task was to improve it.

The source code of LibreOffice is divided to several (at the time of writing: 225) modules, two of them is *writerfilter*, which contains the import filter for the DOCX file format and *sw* (StarWriter), the module of Writer application.

At the begining, the writerfilter module already contained code to map Microsoft Office’s concepts to LibreOffice’s ones (for example Microsoft Office uses section breaks for different page settings, LibreOffice uses page styles for that purpose, etc.), and an internal RTF import filter within the sw module did something similar, but without less features.

In this paper I represent my work on re-writing the RTF filter in the writerfilter module, getting rid of code duplication and adding support for mappings which were missing in the old RTF filter.

The rest of this paper is structured as follows. First, I introduce necessary background knowledge, which was present before the current paper, but is needed to understand the rest of this work (Section 2). Section 3 describes the design of the solution, as well as relation with the underlying techniques. Next, I detail the implementation I created (Section 4). After that, I evaluate the created module (Section 5). Finally I give a summary, including future development directions (Section 6).

2 Background

Filters

Writer follows the Model – View – Controller principle. Once the user wants to open a document, the model of the document is loaded from serialized data. After loading the document model, the controller initializes the view, and the user can see the result of the import. Serialization is done similarly: the model of the document is exported to a file or stream, and the view is simply destroyed. Such conversion between the document model and serialized data is called a *filter*.

A filter in LibreOffice is a class, that implements given interfaces, and performs the import or the export (or both) of a file format to one of the six applications.

Registrations of such classes are usually handled via UNO (Universal Network Objects), the interface based component model of LibreOffice.[1] Each filter has a configuration file, which lists the file format detector service and the filter service.

I did not have to pay attention to the RTF file format detector service, as it was already working and I had no plans to improve it. The filter was an UNO one, but the import part of it was just a stub, that called the old RTF importer, that was a builtin one.

Implementing a filter via UNO has multiple advantages:

- The applications are written in C++, but the implementation of UNO services can be anything supported by UNO (e.g. Java or Python).
- The filter can be built once the UNO interfaces implemented by the application are available, even if the source code of the application it not available[2].

Implementing an RTF Reader

The already cited RTF specification defines the major tasks of an RTF reader:

- Separate text from RTF controls.
- Parse an RTF control.
- Dispatch an RTF control.

Separating text from RTF controls is relatively simple: all RTF controls begin with a backslash and can contain a defined set of characters only.

[1]Old filters are not using UNO and are bundled together with applications (e.g. Writer).

[2]This may be strange as a goal for a free software project, but given that LibreOffice is huge and building it is non-trivial, it's useful.

Parsing an RTF control is also relatively simple: RTF defines different type of control words – each of them is a simple rule of what type of parameters can they take (if any)[3].

Dispatching an RTF control, on the other hand, is relatively complicated: once needs having a really good knowledge of the API of the target application, since most of the RTF control words will result in an API call, passing the parameters of the control word to the document model.

Next to control words, RTF has the concept of groups. They start with a {character, end with a }one – and they should be threated like a stack: the scope of control words starts where they appear and ends at the end of the current group.

RTF Control Word Types

RTF defines the following control word types:

- flags: they take no parameter, for example \sbknone (continous section break)
- destinations: these appear at the start of a group and if they are ignored, the whole group should be ignored, for example \fonttbl (font table: list of fonts used in the document)
- symbols: they don't take parameters, either – special symbols like \tab (tabulator symbol) have this type
- toggles: they switch a property on or off, for example \b (bold characters are marked with this control word)
- value: they always take a parameter, for example \fs (font size)

Now that we are aware of basic concepts of RTF, the next section overviews the API requirements of Writer and explains the design of the filter that can parse and RTF document.

[3]For an exact list of control word types, see below.

3 Design

The design of the RTF filter is determined by the requirements from the RTF specification and the interfaces provided by the domain mapper.

The control flow has the following architecture:

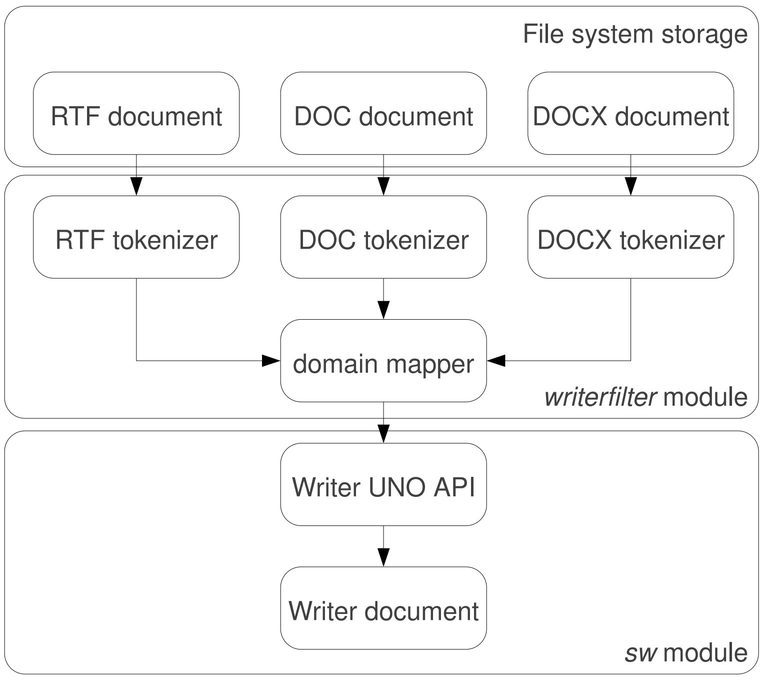

Figure 1: Architecture of the RTF import filter

As it can be seen, my task had two major parts:

- Implementing an RTF tokenizer.
- Improving the domain mapper as necessary.

The domain mapper provides the following interfaces, which has to be implemented by a tokenizer:

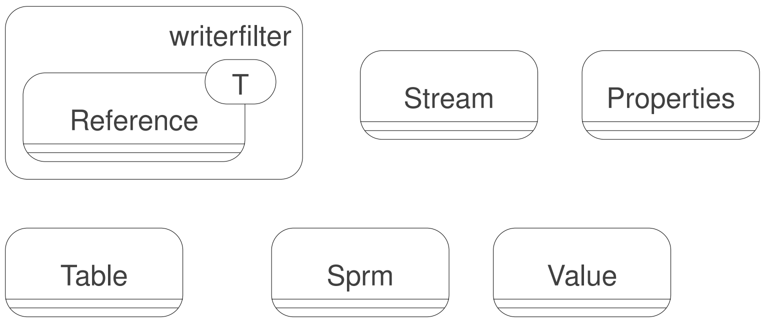

Figure 2: Interfaces provided by the domain mapper

- writerfilter::Reference<Stream> – represents a file stream, that can be tokenized
- writerfilter::Reference<Properties> – represents a set of properties that modify the behaviour of text (font size)
- writerfilter::Reference<Table> – represents tables of settings at the start of document (list of fonts, styles, etc.)
- Sprm – represents a section, paragraph, or run (set of characters) modifier
- Value – represents a parameter of an Sprm

Once these interfaces are implemented, a filter implementation has to:

- Create a domain mapper instance, and pass the Writer UNO API document model to it.
- Create a tokenizer instance, and pass file stream and the domain mapper instance to it.
- Call the tokenizer's resolve() method to alter the document model to represent the contents of the file stream.

The tokenizer then has the following tasks:

- Send initial tables to the domain mapper.
- Start/end sections, paragraphs, runs.
- Wrap the tokenized parameters to Value instances, tokens to Sprm instances.
- Send properties (set of sprms) at the start of sections, paragraphs or runs.

As a result, I designed the following classes inside the RTF tokenizer library:

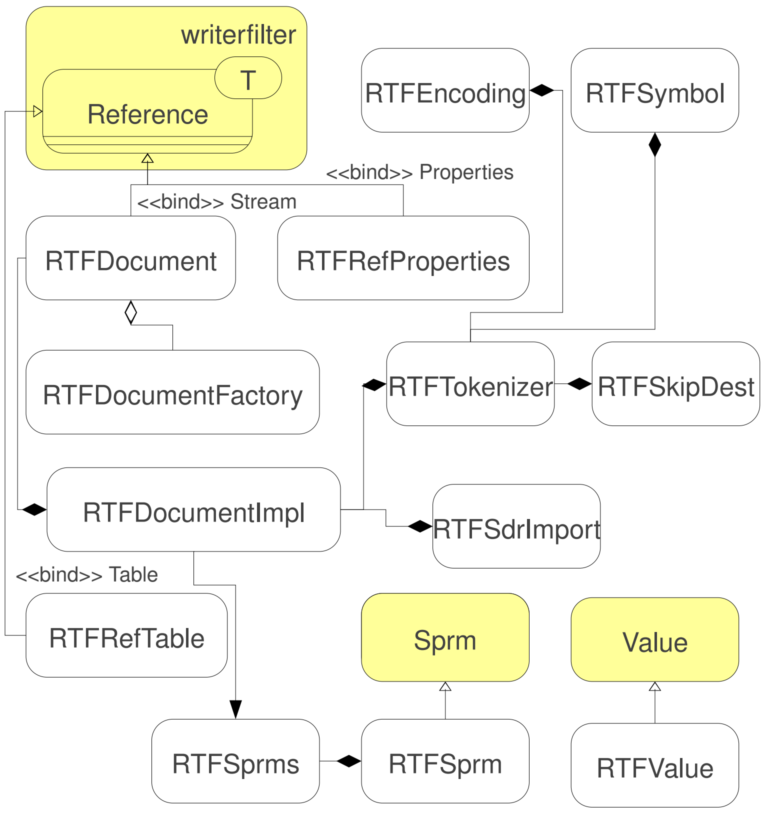

Figure 3: Classes of the RTF import filter

- RTFEncoding – Helper for RTF legacy charsets.
- RTFSymbol – Respresents an RTF Control Word.
- RTFDocumentFactory – Hides RTFDocumentImpl behind an RTFDocument interface.
- RTFDocument – Pure virtual class, the filter can access only this one.
- RTFDocumentImpl – The implementation of RTFDocument, invokes the domain mapper based tokens got from RTFTokenizer.
- RTFReferenceProperties – Sends RTFSprm instances to the domain mapper.
- RTFReferenceTable – Sends tables (e.g. font table) to the domain mapper.
- RTFSdrImport – Helper to import drawings, uses the (Star)Draw UNO API.
- RTFSkipDestination – Helper to skip entire destinations if the destination control word is unhandled.

- RTFSprms – Contains a set of RTFSprm, used by RTFReferenceProperties.
- RTFSprm – represents an RTF control word.
- RTFTokenizer – The core of the tokenizer, knows only how to separate control words (and their parameters) from text.
- RTFValue – represents a parameter to an RTF control word.

Note that a set of sprms can be a tree in fact, as a Value can point to a set of sprms as well, recursively.

4 Implementation

General

The majority of the work was about re-creating an RTF tokenizer that supports the same set of features as the old filter, except with a cleaner design, making it possible to get rid of ugly hacks. Two such example:

- RtfReader – the old filter – used to hardcode the type of the control words, RTFTokenizer now has this in a central table (originally generated from the specification), making it impossible to handle incorrectly the parameters of control words (e.g. handle the parameter of a value as a toggle).
- RTFTokenizer separates the task of separating control words and text, for example the meaning of the special {character is defined at a single place, while it was handled in 19 (!) different places in RtfReader.

Fixed bugs

My RTF improvements can be separated to bugfixes and features.

I fixed bugs in the following topics:

- Comments are not displayed upon saving and reopening RTF, and are lost on re-saving (fdo#36877)
- Group containing tabs deletes previous tabs (fdo#36922)
- Changing imported RTF tilts orientation of text (fdo#35985)
- Subscripts in RTF from LaTex file are missing (fdo#36089)
- Embedded picture invisible, rendering messed up (fdo#37691)

Details about each bug can be found in the Freedesktop Bugzilla[3].

New features

Once I was ready with an RTF filter that was as good as the old one, I worked on the following new features:

- character properties: blinking property, relative font size in superscript characters
- tables: nested tables, vertical merged cells

- footnotes/endnotes: all characters of the foot/endnote mark are in the field[4], the field is properly superscript
- sections: when line numbering is enabled in a section, it is no longer ignored by the importer
- the field type *postit comment* is now imported
- Drawing objects for Word 97 through Word 2007 (shapes) are now handled:
  - basic shapes (rectangle, ellipse, etc.)
  - lines, including free-form ones
  - texts, including vertical ones and their (paragraph and character) formatting
- form fields (used when filling out forms in a document): all types supported by the RTF format are handled
- OLE objects: Their result is imported as a picture - RtfReader did not import anything. When native is available, then it's handled as well, but no automatic conversion is done yet (for DOC files there is an automatic conversion from MathType to Writer formula).
- text frames:
  - anchor type is now parsed by RTFTokenizer (no longer always assume *to paragraph* but also handle *as character*)
  - handling of invalid nested frames now match the behaviour of Word

DOCX changes

Given that the domain mapper operates on tokens and the tokens are derived from the DOCX specification, the domain mapper is highly DOCX-specific. That means that when the developer documentation is lacking, usually the best is to see how the domain mapper works with DOCX documents, then other tokenizers can do the same. As a result, I also fixed bugs in the domain mapper, improving DOCX support in general. Here are a few items I added support for:

- double strikethrough character property used to have an effect till the end of the whole document (!)
- text-to-text alignment is now imported

[4]A field is a special set of characters which has an instruction and a result part. Examples: page number, date, footnote number.

- restart of footnote numbers
- extra paragraph at the end of footnotes is no longer inserted

5 Testing

LibreOffice supports two kind of automated test types:

- unit tests
- subsequent tests

Unit tests are always executed at the end of the build of a module, the failure of them is considered a kind of build failure. This makes them quite useful, however their scope is a bit limited: they can't use UNO services from module they don't build-depend on.

Subsequent tests are however executed manually, once the complete product is built. Given that most developers don't run these tests (as they are not forced to), it's less useful, but these tests can use all UNO services.

In case of RTF import, I decided to implement unit tests. I make the domain mapper handle the lack of Writer services handle gently (i.e. notice the lack of existence, but don't fail) and this way a unit test could make sure the import of an RTF document results in the expected return code, but it says little about the actual rendered layout.

LibreOffice uses the CppUnit framework for unit testing. Given that I wanted to be sure adding new test documents do not require code modifications, I defined the following document classes:

- pass: the document should be imported without any error
- fail: the import should return a proper error code (should not crash)
- indeterminate: the importer should just not crash

The idea is that directories can represent the given document classes, and each document in that directory is checked if the desired behaviour is achieved.

Creating test documents was the next step. During development, I followed a test-driven approach, so I already had documents which should be imported without errors in the following categories[5]:

- hello world: This document contains a single "Hello World!" string in a single paragraph.
- character properties: Contains testcases for properties which affect characters, so can differ within a paragraph as well. Examples: a character being bold, italic, underlined.

[5]The test documents were created by me in an ad-hoc way, then I checked with the QA engineers if they are complete. We had several iterations, this test set now covers all areas where the old RTF importer worked properly, plus a few more features.

- character styles: Contains testcases for character styles, for example a set of characters (but not the whole paragraph) being marked as a *quote*, then style can result in characters being italic or some other set of character properties.
- paragraph properties: Contains testcases for properties which should be the same within a paragraph. Examples: indentation (left, right, centered, justified), tab positions.
- unicode support: RTF supports both custom encodings and unicode, in which case the multibyte character has to be described using control words, so the physical document is still ASCII. This document contains samples for various strings, having accents in multiple languages.
- numberings: Contains numbered lists, numbering with bullets and other styles.
- pictures: Contains pictures with different anchoring types: as character, to character, to paragraph, to page.
- tables: Contains different table examples: tables with different row height and column width, cell background color, cell borders, nested tables.
- sections: Contains sections with different properties: borders, number of columns, continuous sections, sections with different page break types.
- headers and footers: first page headers, headers on even, odd or all pages. Same examples for footers.
- footnotes: footnotes at the end of the page, section or document; restart of footnote numbering, numbering type (Roman, Arabic, etc.)
- line numbering: this is an extended version of the sections document, containing a section with line numbering enabled.
- bookmarks: simple bookmarks, bookmarks across tables, references to bookmarks.
- red lines (change tracking): samples for added, deleted and moved text.
- fields: test cases for various field types: page number, date, references, URL's.
- table of contents: this is an extended version of the fields document, containing several headings and finally a table of contents.
- drawing objects: various shape types, including rectangles, ellipses, freeform lines.
- forms: a document containing form controls: label, inputbox, checkbox, radio buttons.

- OLE objects[6]: replacement graphics – this document contains an OLE object with native data missing (it's a chart object).

- OLE objects: load native data – this document contains an OLE object with native data, which can be edited (it's a math object).

For failing samples, I looked up what security advisories of the product referred to the RTF importer earlier: that provided test documents which should fail with a proper error.

Last, I needed a huge collection of RTF sample documents which I could throw in the indeterminate directory: for that I used an existing Python script that collected doc files from public bugzilla entries. With little modification I modified it to use crawl RTF samples.

[6]OLE objects are embedded to documents in two parts: first the native data, which is used during editing if the required application is available – and a replacement graphic, that can be shown even in case the editing application is missing.

6 Future work

At the end of the internship I did some measurements about how many percent of the control words are implemented from the specification. Of course that number does not mention how relevant a keyword is: e.g. a drawing keyword not used since Word 6.0 is way less important than a character property that sets the current run as bold.

Version 1.9.1 of the specification contains 1815 control words, while the RTF import filter at the time of writing parses 385 of them. That alone means that there are plenty of possibilities to improve the filter.

However, there are smaller tasks which could be implemented, once the domain mapper would support them:

- protected sections
- page breaks: even/odd ones
- make DomainMapper_Impl :: PushPageHeader() handle “left” and “all” headers separately for RTF
- NS_ooxml :: LN_EG_SectPrContents_pgNumType is completely ignored (page numbering style, restart)
- OLE objects: convert native data (mathtype to mathml, etc), like the doc import does
- crop of images is ignored

Additionally a performance idea: currently each control word is converted to a numeric constant, which is then parsed by based on the type of the control word. Converting to a constant means reading lines of a static array, comparing the string value of the control word and the current entry, and then reading the numeric value, if they match. This can be quite slow for large documents, using gperf instead would be a better solution.[7]

[7] It’s a code generator that generates perfect hashes: so looking up the numeric constant in a table based on string value would cost a single string comparison only.

References

[1] Libreoffice, http://libreoffice.org/

[2] Word 2007: Rich Text Format (RTF) Specification, version 1.9.1, http://bit.ly/qD1CTV

[3] https://bugs.freedesktop.org/