ODF Support in KWord 2

Girish Ramakrishnan
girish@forwardbias.in

About me

- Girish Ramakrishnan
- Ex-Troll
- ForwardBias

Agenda

- Whirlwind tour of ODF
- KWord 2 ODF Text Design
- ODF Testing Framework
- What's cooking
- Questions

What is ODF?

- XML Format for spreadsheets, presentations, word documents
- Brief history
  - StarOffice (Sun) → OpenOffice.org → OASIS
- The spec is huge (~800 pages)
  - Cross references many other specs like XML, SVG, XSLT, XFORMS.
  - Surprisingly readable (damn...)
- For this preso, we care only about text documents

ODT

- Text documents are stored as .odt
- Can contain
  - Paragraphs
  - Lists
  - Tables
  - Frames
  - Sections
  - Graphical elements
  - Notes, annotations, ruby text, references, bookmarks, TOC, change log, ...
- Imagine lots of XML element names and attributes for the above

ODT File Contents

- Zip file contains the following
  - META-INF/manifest.xml (Contents of the zip file relevant to ODT and the mimetype)
  - meta.xml (Author, Creation date and so on)
  - settings.xml (Application settings)
  - content.xml (The content)
  - styles.xml (The styles i.e how the content looks)
- Content is kept separate from the style
  - Easy to extract contents
  - Styles can be reused

ODT Styles

- Classified by type/family
  - Paragraph styles
  - Character styles
  - List styles
  - Outline styles
  - Conditional styles
  - Section styles, Frame styles, ruby styles, note styles, annotation styles...

Styles

- Classified by concept
  - Default style
    - Used when an entity does not state a style
  - Named style
    - Styles that have ”names” (style manager)
  - Master style
    - Styles for pages, header/footer
  - Automatic style
    - Styles created on the fly
- Styles can have a parent style
  - Attributes cascade
  - Automatic styles usually have a parent

ODT File contents

- styles.xml
  - Contains master styles, named styles, default styles
  - Contains automatic styles too...
- content.xml
  - Contains all the content
  - Automatic styles

KWord – KoStore

- Task: Unpacking the odt
- libs/store – Read/Write archives
- Also handles encrypted stores
  - Queries user for password, if need be
- Task: Extract contents of the archive

```
KoStore *store = KoStore::createStrore(
            ”test.odt”, KoStore::Read);
store->open(”content.xml”);
QIODevice *contentDevice = store->device();
```

KWord – Parsing XML

- Task: Parsing XML
- We have our own DOM parser written for speed & memory usage
- API identical to QDom (meant to be a drop-in for Qt Dom)
- Main differences
  - Nodes are loaded on demand
  - Read only
  - No DTD handling

ODF Library - libs/odf

- Task: Process ODF documents
- Understands the ODF XML Schema
- Goal is to make it a library that can be used for processing ODF documents without requiring it to be rendered
- Use case: document processors, report generators

ODF Library - libs/odf

- Query style information of the document
- Push a style and its parents into a stack and query its properties (KoOdfStyleStack)
- Holds information about a style and query for existing style of the same format (KoGenStyle, KoGenStyles)
- Far from complete
  - Input/Output is mostly XML. User needs to know ODF

KWord - ODF Text

- Task: Load the text into a structure that can be drawn
- Qt 4 brings Qt Scribe (i.e QTextDocument and friends)
- ODF has lots of features compared to Qt Scribe
  - Extend Qt Scribe

Qt Scribe

- Central class is QTextDocument
- A paragraph is a ”block”
- Blocks are made of fragments
- A list is set of non-contiguous blocks
- Blocks, fragments, lists can have formats
  - QTextBlockFormat, QTextCharFormat, QTextListFormat
- QTextCursor can be used to manipulate text
  - cursor.insertText()

Extending Qt Scribe

- The formats are nothing but a QMap<int, QVariant>
- One can store arbitrary values in the formats
  - Store ODF specific properties in the format using custom keys in the QMap

Loading ODF

- Task: Load ODF into the QTextDocument
- All styles are parsed
  - Named and default styles are put in Style manager
  - Style manager generates a unique id for each named style

Loading the document (Text)

- Use QTextCursor to load text
- Applying styles
  - Convert ODF style to QTextFormat
  - Save the style id into the format's QMap
- That's it!

Editing the document

- Edit/make changes to the document using QTextCursor (just like QTextEdit)
  - We throw away the DOM
- When applying a named style, overwrite the existing format
- When applying an automatic style, merge with the existing format

Saving the Document

- Write blocks one after the other
- Generate automatic styles for each block by
  - Diff'ing against the ”StyleId” style

Lists

- Lists can be nested and have styles for different levels
  - QTextList does not support nested lists
- KoListStyle
  - Maintains a QTextList for each level
- At layout time, we generate the numbering for the list items
  - Complex stuff

Flake

- Text documents can have embedded content aka frames
  - Frames can be non-text
- Flake is a ”shape” framework (Very similar to GraphicsView)
- A flake contains
  - shape(s) that can be placed on a canvas
  - Provides tools to modify the shapes
  - Shapes can load/save ODF
- It's plugin based
  - Textshape plugin for text handling

Layout

- Qt Scribe can be provided a custom layouter
- KWord has a custom layout to accomodate ODF features
- Allows run around of text (any shape for that matter) along painter paths

ODF Testing Framework

- Goal: Testsuite that can check if documents are loaded and saved correctly by ODF
- Test cases from OpenDocument Fellowship
- Uses the QtScript binding generator (TT labs)
- Strategy
  - Loading Document using kotext
  - Load document created by a script
  - Compare both documents
- Does not check if the document is actually rendered correctly

Current Status

- 1.6 already had good ODT support
  - Qt 3 Text document was forked and extended
- 2.0, lots of rewritten code
  - Moving to Scribe
  - Flake architecture
  - Lots and lots of code, with very little testing (like KDE 4.0, playing catch up with 1.6. We need help!
  - No plans to keep API BC after 2.0

What's cooking

- Style manager (Thomas)
- Headers, lists (Girish, Roop)
- TOC, Page layout (Peirre, Sebastien)
- Paragraph tool (Florian)

- There is a LOT of work to be done
  - Usable but not ready for user testing. There are lots of small bugs. We need developers who can test and fix
  - Tables
  - Many ODF features have no UI
  - KWord Audit has list of tasks TODO.

KWord and ODF spec

- Constant feedback to Oasis
  - Unfortunately, one needs to pay to be a member of Oasis
  - Channelled through David Faure and Thomas Zander

Questions?

Thank you!
(With special thanks to NLNet.nl,
Thomas, Thorsten, David)