BERLIN 2012 CONFERENCE

17th-19th October

How to add a new Writer feature?

- Miklós Vajna
- 18 October 2012

LibreOffice

Introduction

Not only features count

How a member of the community can help

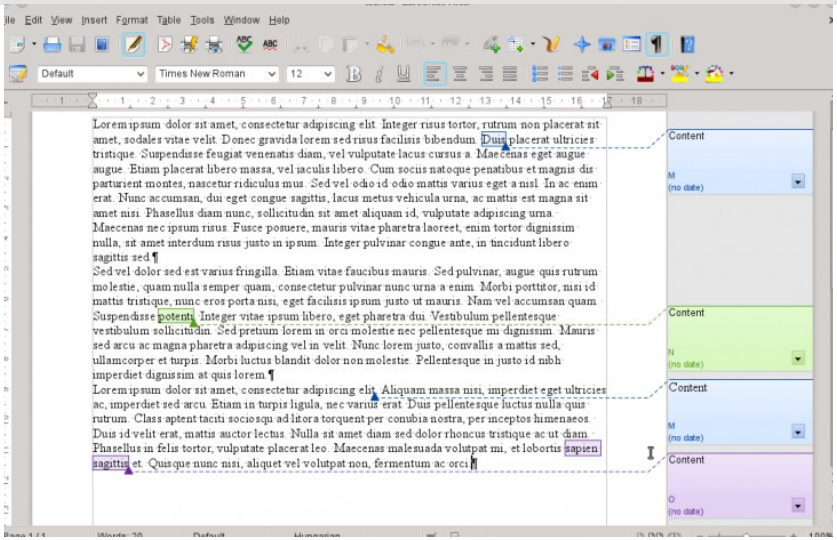
- Helping users (mailing lists, forums, IRC)
- QA (reporting, confirming issues)
- Development
- Writing guides
- Design
- Operating the infrastructure

How a developer can help

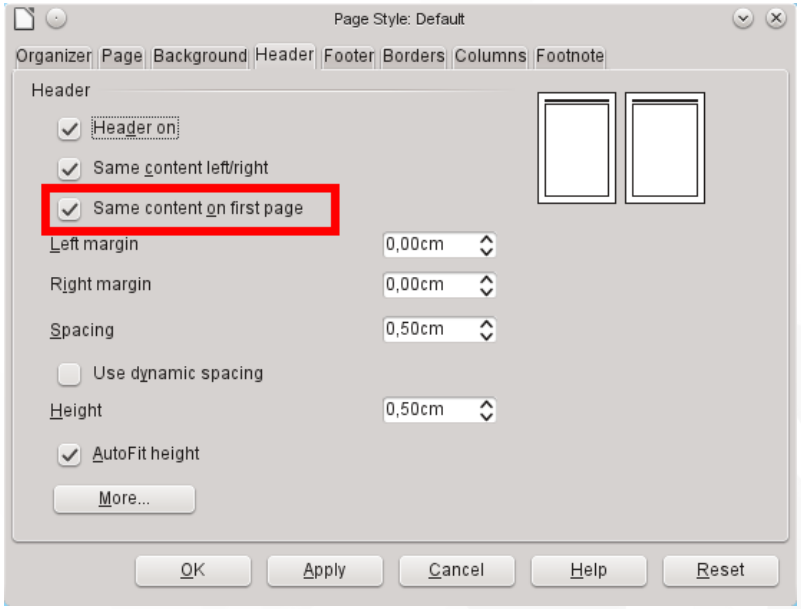
- Fixing issues
- Testing
- Implementing new features
- Refactoring
- Mentoring
- Documenting

Examples: new Writer features in LibreOffice 3.7

- Commenting a text range



- Different header / footer on first page



Steps for a new feature

- Document model
- UNO API
- Layout
- Filters
- UI
- Tests
- Documentation
- Specification

Document model

You would think this is the only necessary part

What is the document model

- Classic Model-View-Controller
- One open document ↔ SwDoc
  - Basic building block: paragraphs ↔ SwNode
  - Code lives under sw/source/core/
  - No separate node for text runs, SwpHints instead
    - SwTxtAttr inside for each property: start/end/what
  - Lots of other collections, for example:
    - SwPageDesc: page styles
- Added for “first page header” feature:
  - SwPageDesc::aFirst

How to explore the document model

- XML dump:

```
$ SW_DEBUG=1 ./soffice.bin
```

- Shift-F12 creates nodes.xml
- Through UNO:

```
enum = ThisComponent.Text.createEnumeration
para = enum.NextElement
para = enum.NextElement
xray para
```

- Same for runs

UNO API

To be able to manipulate it from macros

Purpose of the UNO API

- Macros can read and write the new feature
  - Handy: you can test it before messing with the UI
- UNO filters use this as well
- Makes writing testcases easy as well
- Where it lives: sw/source/core/unocore/

How to play with macros / UNO API

- Turn on contextual spacing for a paragraph:

```
enum = ThisComponent.Text.createEnumeration
para = enum.NextElement
xray para.ParaContextMargin
para.ParaContextMargin = True
```

- Enable different header and footer on first pages:

```
oDefault = ThisComponent.StyleFamilies.PageStyles.Default
oDefault.HeaderIsOn = True
oDefault.HeaderIsShared = False
oDefault.FirstIsShared = False
```

- Attach comment to a text range:

```
oTextField =
oDoc.createInstance("com.sun.star.text.TextField.Annotation")
oTextField.TextRange.String = "Content"
oDoc.Text.insertTextContent(oCurs, oTextField, True)
```

Layout

Display what was hidden so far

Layout

- The “View” from MVC
- In case of Writer, it has its own model as well:
  - In the document model, it's just paragraphs
  - In the layout, it's “just” frames:
    - One opened document ↔ SwRootFrm
    - One page ↔ SwPageFrm
    - One paragraph ↔ SwTxtFrm
- One model element ↔ multiple layout elements
  - Headers on each page
  - Paragraphs splitting across pages
- Linked with SwClient / SwModify
- Dump: same as document model, but F12, not Shift-F12

Filters

Let's try to survive the restart

Filters

- Export filters: save SwDoc to a stream
- Import filters: open the stream and populate SwDoc again
- ~All filters are “alien”
  - It's known that they loose some info during conversion
  - Exception: ODF filters (a.k.a. “own”)
- Type:
  - UNO: DOCX import, RTF import
  - Builtin – uses the internal sw API: DOC import, Word export
  - Mixed: ODF (mostly UNO, but some bits are internal)
- Code: sw/source/filter/, writerfilter/, xmloff/

User Interface

Let users start to notice your work

Old / new user interface

- Old: let's not use mouse for building a UI
  - Typically a .src, .hrc and a .cxx file for each dialog
  - Each element has fixed position / size:

```
CheckBox CB_SHARED_FIRST
{
    HelpID ="svx:CheckBox:RID_SVXPAGE_HEADER:CB_SHARED_FIRST";
    Pos = MAP_APPFONT ( 12 , 46 ) ;
    Size = MAP_APPFONT ( 152 , 10 ) ;
    Text [ en-US ] = "Same content on first page" ;
};
```

  - Add one element in the middle of the dialog
    - And you'll have to move all the remaining controls down
- New: Glade-powered
  - See Caolán's talk

UI ↔ document model interaction

- UI is often shared, can't be specific to applications
- Invented solution: SfxItemSet
- Checkbox → SfxItemSet:

```
aSet.Put(SfxBoolItem(nWSharedFirst,aCntSharedFirstBox.IsChecked()));
```

- SfxItemSet → document model (SwPageDesc):

```
rPageDesc.ChgFirstShare(((const SfxBoolItem&)
 rHeaderSet.Get(SID_ATTR_PAGE_SHARED_FIRST)).GetValue());
```

- Similar in the other direction
- Shared code is under svx/source/dialog/, cui/
- Writer-specific code is under sw/source/ui/

Testcases

Why repeat old bugs, if there are so many new ones?

Testcases

- Test types in LibreOffice:
  - Unitcheck: at the end of every partial module build
    - Should be super-fast (CVE tests, for example)
  - Slowcheck: at the end of a toplevel build
    - Filter tests typically go here
  - Subsequentcheck: old Java tests
    - Slowly migrated to native code
- What to test:
  - Document model (UNO API)
  - Layout (XML DUMP + assert XPath expression):

```
CPPUNIT_ASSERT_EQUAL(OUString("First header"),
 parseDump("/root/page[1]/header/txt/text()"));
```

Import / export tests

- Create minimal reproducer
- Import or import → export → import
  - You know what you want to test
- Check the resulting document model using UNO API:

```
CPPUNIT_ASSERT_EQUAL(6, getLength());
```

- Make sure it passes
- Revert the bugfix / feature
- Make sure it fails
- Drop the revert, squash the two commits
- Code: sw/qa/

Documentation

When the obvious is not that obvious

User help

- You know, the thing nobody cares about
- Comes up every time the user presses F1
- Where to add it?
  - Use the UI
  - Find existing text after/before you would insert new content
  - Use git grep under helpcontent2/
  - Add new content to the XML files
- Problems
  - No linkoo, so run make dev-install to test
  - Individual ID for each paragraph
    - Don't worry, should be individual inside a file only
    - Copy an existing one and increment

Developer help

- Use doxygen
- Document each class – at least a one-liner mission statement
  - Preferably each public method
  - Use bin/find-undocumented-classes
- Old code uses German comments
  - If you figured out what they mean anyway…
  - … then spend one minute on translating it
  - Use bin/find-german-comments

Specification

We should have started with this… or not?

Specification

- We serialize everything to ODF
  - ODF is an open standard, takes years till it's accepted
  - Catch-22
- Solution: we implement first, call it “ODF Extended”
  - Then propose a modification to the standard
  - Ideally: using extension namespaces before it's accepted

ODF JIRA ticket example

- Rationale: what's the motivation
- Requested changes to the standard
  - Text changes
  - Schema changes
- Impacts
  - Backwards compatibility
  - Easy if the new element is optional

BERLIN 2012 CONFERENCE

17th-19th October

Thank you ...

- Any questions?
- Slides: http://vmiklos.hu/odp/