LibreOffice: Code Structure

Hermenutical keys to a complex code-base

Michael Meeks
General Manager at Collabora Productivity
michael.meeks@collabora.com

mmeeks, #libreoffice-dev, irc.freenode.net

“Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls...” - Jeremiah 6:16
Overview / Agenda ... Chunk #1

- Codebase overview
  - Internal core modules, internal leaf, (ignore externals)
- Build + package: gnumake + scp2
- Code organisation / git bits
- Bear in mind: this is a 20 year old codebase
  - The code-base is rather better than can be expected, and things continue to improving over time.
Module overview – lowest level
Internal non-leaf modules: UNO modules

- Top-level directory names:
  - `make dump-deps.png`
    # needs graphviz
- Each module has a README file:
  - eg. `sal/README`
- `sal`: - at the bottom
  - The system abstraction layer
  - 'tools' is an obsolete internal ~duplication of this module
- `salhelper`: - wrapper code around `sal` – also part of URE
What is the Uno Runtime Environment (URE)

- We'll come onto UNO later in detail but … for now:
  - “Uno Runtime Environment”
  - cf. JRE → Java Runtime Env.
  - Belongs to the pipe-dream of a world where UNO was re-used by other applications.
- Provides an ABI / API stable abstraction layer for the suite
  - So you can write C++ plugins
- Careful:
  - We have to watch our ABI here.
  - ABI control via C symbol map files
UNO module dissection

- **store**: obsolete & irrelevant.
- **registry**: used to keep interface descriptions
- **unoidl**: used to create / compile interface descriptions: an IDL compiler.
- **cppu**: C++ UNO
  - Implements basic UNO types, and infrastructure for C++
- **xmlreader**: very simple XML pull parser
- **cppuhelper**:
  - luggage to bootstrap UNO, create UNO components etc.
More associated modules

**ucbhelper** – Universal Content Broker (ucb) C++ wrapper / helper classes

- **ucb** - Provides an unusual Virtual Filing System abstraction

**i18nlangtag:** – module that handles BCP47: a powerful way to represent subtle language / locales

**jvmfwk:** Java / UNO integration

**comphelper:** lots of good C++ stuff for using UNO – not stable enough to go into the URE
Module overview – middle level
More associated modules

- **basegfx** – algorithms / graphic etc. for basic graphics.
- **tools**: – more basic types:
  - SvStream – internal stream
    - vs. UCB vs. sal/ file pi
  - Color COL_RED etc.
  - INetURLObject – canonic URL handling
  - SolarMutex (the big lock)
  - Resources, translation
  - Polygon / PolyPolygon
  - Date / Time classes
  - A total grab-bag of things
Unit testing pieces:

- **cppunit**: - ~all our tests are ultimately cppunit tests though this is an external module.

- **unotest**: low level testing of simpler / UNO infrastructural pieces. Bootstrap UNO enough to be able to test filters, components etc.
  - All of that requires types / services, configuration etc.

- **test**: helpers for testing standard interfaces, more advanced tests: brings UCB bootstrap (for streams), VCL initialization, graphic filter pieces etc.

- CppUnit*_.mk files in directories
Other non-graphical bits ...

- **i18nutil**: 'honest C++ code' wrapping UNO l10n madness eg. 'bool isUpper (sal_Unicode c);'

- **unotools**: C++ helpers for using UCB eg. SvStream *CreateStream( OUString &rPath);
  - Misc. font & config handling

- **sot**: handles OLE2 / compound file storage for binary documents

- **svl**: non-graphical (no VCL dependency) pieces originally from svtools/ or sfx/ eg. **SfxItemSet** – the key C++ property-bag class
  - Undo/Redo, and more ...
  - 'tools' but higher up ...
• **vcl**: - Visual Class Libraries – the LibreOffice graphical toolkit, on this – more later.

• **toolkit**: - a particularly thin & horrible UNO API wrapper with Model/View flavour on top of vcl.

• **canvas**: - alpha transparent, anti-aliased UNO rendering API – more modern rendering than VCL, primarily used by *slideshow*
  - DirectX, Cairo & VCL impls.

• **cppcanvas**: - C++ wrappers to make using the canvas less bad.
Non-Graphical grab-bag ...

- **basic**: - the StarBasic parser / interpreter & run-time.

- **xmlscript**: XML serialisation of (orrible) basic dialogs which wrap the toolkit pieces for in-document scripting / macro dialogs.

- **connectivity**: - UNO implemented database drivers for all manner of backends:
  - Postgresql, MySQL, Mozilla addressbook, Evolution, JDBC, ODBC etc. etc.

- **sax**: - wrapper of libxml2 – providing an UNO sax API for parsing XML files, and an XFastParser for tokenising them.
Graphical grab-bag

- **svtools**: - lots of pieces
  - tree / list controls
  - table control
  - dialog helpers
  - accessibility helpers
  - options wrappers
  - print dialogs
  - filedialog helpers
  - imagemaps
  - wizard helpers etc.
Module overview – upper level
• **framework**: manages docking, toolbars, menus, status bar, sidebars, task-panes
  - 'new' (over-engineered) code with heavy UNO logic

• **sfx2**: works closely with framework, core of the app.
  - load / save logic: SfxMedium
  - manage views on top of framework
  - 'Help' pieces, quick-starter,
  - Dialog helpers: tab dialogs
  - Document meta-data dialogs
  - Template management
  - Shared style pieces.
Miscellaneous pieces

- **formula**: - nominally shared code extracted from calc (sc) for use in *reportdesign*

- **avmedia**: - Audio / Video media – multimedia abstraction over DirectX, quicktime, gstreamer

- **linguistic**: - implements UNO services for spell / hyphenator & thesaurus.

- **xmlsecurity**: - XML document encryption and signing used for ODF.

- **vbahelper**: - helper code for implementing VBA / macro interoperability with MS Office
Load / save / filter logic ... 

- **package:** - ZIP file compress / de-compress, also handles manifest files in the .zip with UNO stream / storage interfaces

- **xmlloff:** - ODF file filters and helpers to load / save our model to/from ODF.
  - Often working in conjunction with eg. sw/source/filter/xml/

- **filter:** - meta-data to manage, register and auto-detect filters
  - Also flat-ODF, XSLT filters, graphic filters, flash + svg export & more.

- **oox:** - shared MS Office Open XML (import) filter pieces.
Applications ...  

- **desktop**: - legacy name, StarOffice 5 had a 'desktop' complete with 'Start' menu etc.
  - here lives the real 'main'
  - desktop/source/app/app.cxx
- **sd**: - Star Draw (Impress)
  - Drawings + Presentations
- **sw**: - 'Star Writer'
  - Word processor
- **sc**: - Star Calc
  - Spreadsheet
Caveats: this is a simplified picture

- That was just the non-leaf nodes.
- This is a linking dependency graph
  - UNO component use is hard to graph / grok.
    - fundamentally a dependency breaking technology.
- other important bits:
  - *cui*: - a big bag of dialogs – split to avoid loading
  - *ucb*: - Universal Content Broker
  - *chart2*: - embedded chart rendering and model
  - *slideshow*: - the piece that renders your slideshow.
  - *solenv*: - where build infrastructure lives.
Build + Package
Build: configure etc.

- autoconf / configure reasonably sane
  - autogen.sh – a wrapper around autotools
    - builds & runs configure script etc.
    - keep your parameters in autogen.input
  - Builds:
    - config_host.mk from config_host.mk.in
      - This contains all the variables we need.
    - config_host/*.h – from templates
      - containing the build configuration.
Android / Online build

- **Android**
  - Normal core.git, configure nicely:
    - `--with-android-ndk, --with-android-sdk` etc.
    - Checkout README.android
  - Binaries end up in android/ as APK files.

- **Online**
  - Normal autotools style configure / make / make run.
  - Ensure you use:
    - `--with-lo-path=core.git/instdir`
    - `--enable-debug`
  - To get working unit tests
Build: gnumake ...

- gnumake used in some odd ways
  - code is in solenv/gbuild/
  - Each module has it's own Makefile
    - You can build each independently after a full-build.
    - All rules are built by $(call Function,...) magic, we don't use generic / built-in rules.
      => if something is compiled – we have an explicit rule for it (somewhere)
  - Following the rules is not trivial: $(1) → $(7)
We build a working image into 'instdir/'

- instdir/program
  - Contains a runnable image post 'make'
    - The authoritative location for libraries
      - make && instdir/program/soffice.exe

- workdir/*
  - object files, and build intermediates here
  - generated headers
  - unpacked external source code etc.
Finally – key modules in build...

- **postprocess**
- **packimages/**
  - Using solenv/bin/packimages.pl – build icon theme .zip and sort it by access pattern
- **CustomTarget_registry.mk**
  - Build configuration files from officecfg/
- **Rdb_Services.mk**
  - Build services.rdb file from .components
- **officecfg/**
  - Home of all defaults / office configuration / settings
Internal module organisation ...

- `include/`
  - All global includes live in `include/<module>/`
- `sfx2/inc`
  - includes local to module
- `source/*`
  - source code for module
- `source/inc/`
  - other includes local to module
- `uiconfig/`
  - new-style XML UI descriptions
- `sdi/`
  - descriptions of slots / actions
- `qa/`
  - unit tests, test file data etc.

- Lots of things moved over time:
  - `git log -u --follow --include/sfx2/new.hxx`
    - Only works for one file
Questions / conclusions

- Are you still alive?
- That was very dense and high-level
- Hopefully it's useful.
- We have a lot of modules
  - You can safely not know about the vast majority of them.

Oh, that my words were recorded, that they were written on a scroll, that they were inscribed with an iron tool on lead, or engraved in rock for ever! I know that my Redeemer lives, and that in the end he will stand upon the earth. And though this body has been destroyed yet in my flesh I will see God, I myself will see him, with my own eyes - I and not another. How my heart yearns within me. - Job 19: 23-27