LibreOffice: Core Classes *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 ... Core

- System abstractions, basic types
  - sal / tools
  - strings, translations
- Rendering / GUI pieces
  - Vcl
    - Widget layout / old-style dialogs
    - Main-loop & thread / locking
    - Images
  - basebmp, basegfx, canvas, cppcanvas, drawinglayer

System Abstraction Layer (SAL) pieces

Strings ... include/rtl/ustring.hxx ...

- Two important string classes
  - sal / immutable strings:
    - ref-counted
    - OUString – UTF16, 32bit lengths
      - The predominant string type
      - Used for UNO calls, and ~all internal storage
    - OString – unspecified 8bit encoding, 32bit length
      - Used in a few corner cases where needed.
- *include/comphelper/string.hxx*
  - lots of useful helpers.

Strings ... constructing & mutating

- OUStringBuffer
  - *include/rtl/ustrbuf.hxx*
  - Used to efficiently construct strings, concatenate them etc.
  - steal to an OUString with 'makeStringAndClear()'
  - OUString a(“foo”); a += “baa”
    - 3 allocations, 2 frees.
  - OUStringBuffer – can help ...
- Construction from const char foo[N] is implicit
- OUString translation:
  - ResId(STR_FOO).toString()
  - ResId etc. lives in *tools/* ie. high above *sal/*

OUStrings ... Translation ...

- Translated resources keyed from a unique integer ID
  - This is scoped to the module / resource file eg.
  - *sw/inc/access.hrc* – shared between .src and .cxx
    - `#define STR_ACCESS_DOC_NAME (RC_ACCESS_BEGIN + 1)`
  - *sw/source/ui/docvw/access.src* – define the en-US value:

```
String STR_ACCESS_DOC_NAME
{
    Text [ en-US ] = "Document view";
};
```

  - *sw/source/core/access/accdoc.cxx:*

```
SetName( GetResource( STR_ACCESS_DOC_NAME ) );
```

  - Should be easy to extend ...
- Resource files compiled by *rsc/* code to a binary .res file eg.
  - *program/resource/swen-US.res* – in the install

Stream APIs ... - all URL based

- *include/osl/file.hxx* – (from *sal/osl*)
  - C++ Volume / File / DirectoryItem API
- *include/tools/stream.hxx* – (SvStream)
  - C++ more traditional stream object
    - lots of variants, buffering
- *udkapi/com/sun/star/io/XinputStream.idl*
  - UNO stream impl. - as implemented by UCB, and package code.
- *include/unotools/streamwrap.hxx*
  - Converts SvStream ↔ UNO

Handling URLs ...

- UNO wrapper is somewhat grim
- Includes/tools/urlobj.hxx
  - INetURLObject
    - Also used for File URLs.
  - Does most of what you would expect from a URL API.
    - Escaping: encode / decode etc.
    - Manipulate components
    - Get user/protocol/path information etc.

Visual Class Libaries (VCL) …

VCL – a bit about it vcl/

- The LibreOffice toolkit
  - Lots of backends:
    - *headless/* - ie. No display pixel-bashing
      - *android/ & quartz/* - for Android /iOS
      - both ultimately 'headless' sub-classes.
    - *unx/*
      - pluggable backends for gtk2, gtk3, KDE3, KDE4
    - *win/ & aqua/* - Windows / Mac backends
  - *generic/*
    - shared code between unx-like backends

VCL main-loop / mutex / events ...

- LibreOffice is fundamentally single threaded
  - “the” one big lock: is the 'SolarMutex'
  - This is recursive and ~complex.

    Application::Yield / Reschedule
      - releases the lock while we wait
        - for input / timeout
  - code in *vcl/source/* defers to backends for this

  eg. *vcl/headless/svpinst.cxx* Yield / DoReleaseYield
- SolarMutexGuard aSolarGuard;
  - Takes and holds the Solar Mutex
    - Usually used in UNO implementations

VCL: Idle & Timer

- Include/vcl/scheduler.hxx, idle.hxx, timer.hxx

```
class MyIdle : public Idle {
    virtual void Invoke() override
    {
            print (“Hello World”);
    }
};
```

- To defer work:
  - Prioritized: used for
  - Background document layout, spell-check, word-count etc. etc.

VCL event emission ...

- main-loop dispatches timeouts, user events
  - input events – associated with a SalFrame sub-class

    vcl/inc/salframe.hxx

```
class SalFrame { …
  // Callbacks (independent part in
  //     vcl/source/window/winproc.cxx)
  // for default message handling return 0
  void SetCallback( Window* pWindow, SALFRAMEPROC pProc )
     { m_pWindow = pWindow; m_pProc = pProc; }
  long CallCallback( sal_uInt16 nEvent,
                     const void* pEvent ) const
  {
      return m_pProc ? m_pProc( m_pWindow,
       const_cast<SalFrame*>(this), nEvent, pEvent ) : 0;
  }
```

VCL event emission ...

- After mapping the input:
  - eg. *vcl/unx/gtk/window/gtksalframe.cxx*

```
SalWheelMouseEvent aEvent;
aEvent.mnTime = pSEvent->time;
aEvent.mnX    = (sal_uLong)pSEvent->x;
aEvent.mnY    = (sal_uLong)pSEvent->y;
```

  - Call the callback:

```
pThis->CallCallback( SALEVENT_WHEELMOUSE,
                     &aEvent );
```

  - This enters: *vcl/source/window/winproc.cxx*
    - Multiplexed outwards to the VCL / Window internals & listeners.

Tools / links – wrapping a fn. Ptr ...

- **ImplCallEventListenersAndHandler**
  - **Uses *include/tools/link.hxx***
- *include/vcl/button.hxx*

```
class Button {
    Link    maClickHdl; ...
    void SetClickHdl( const Link& rLink )
      { maClickHdl = rLink; }
```

- User does:

```
    Button maButton;
    maButton.SetClickHdl( LINK(this, NewObjectDialog,
                               OkButtonHandler) );
    ...
IMPL_LINK_NOARG(NewObjectDialog, OkButtonHandler)
{
     SAL_DEBUG( “ok pressed” );
```

VCL event emission ... a control ...

- eg. Button ... *vcl/source/control/button.cxx*

```
void PushButton::MouseButtonDown(
                    const MouseEvent& rMEvt )
{ ...
    if ( … )
        Click();
}
…
void Button::Click()
{
    ImplCallEventListenersAndHandler(
          VCLEVENT_BUTTON_CLICK,
          maClickHdl, this );
}
```

VCL: Rendering model ...

- Unlike modern toolkits VCL has two rendering models:
  - Immediate rendering:
    - Render anything, at any time on your Window.
    - All Windows – are an 'OutputDevice' sub-class

```
void  DrawLine( const Point& rStartPt,
                const Point& rEndPt );
```

  - Invalidate → Idle → re-render
    - Wait for the app to be ready to render
    - `Window::Invalidate( const Rectangle& rRect,`
      `sal_uInt16 nFlags = 0 );`
- This causes some issues.
  - cf. *basebmp/source/bitmapdevice.cxx* (setDamageTracker)

VCL: Images ... split Alpha ...

- *include/vcl/bitmapex.hxx / bitmap.hxx*
- Unfortunately VCL was started 20+ years ago
  - No full alpha transparency then.
  - separate 'mask' – with a different bit-depth (1bit) was.
- In consequence:
  - Bitmap – is a non-alpha transparent bitmap (or mask)
  - BitmapEx – combines two Bitmaps: a Bitmap + an AlphaMask
  - This makes pixel operations somewhat complicated
- Bitmaps have different platform representations:
  - BitmapReadAccess / BitmapWriteAccess – to access the underlying pixels
  - eg. *vcl/source/gdi/impimage.cxx* ImplUpdateDisabledBmpEx
- 'Image' – class wraps this – giving a cut-out of an image-strip (obsolete)
- All Image/Bitmap/BitmapEx primitives are pImpl + ref-counted
- This often **doubles** our rendering: render to Bitmap & render to AlphaMask ...

VCL: Bitmaps ... getting stock images

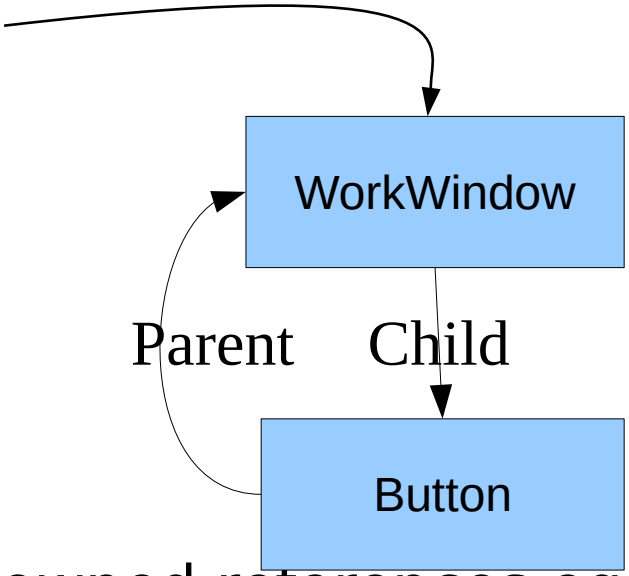
- *vcl/source/gdi/bitmapex.cxx (BitmapEx::BitmapEx (ResId ...)*
  - gets string name from resource
  - loads image from 'image tree' singleton.
- *vcl/source/gdi/impimagetree.cxx*
- Some nice sample code to read through
  - Used to load themed images.
  - Look for /.zip/
  - Notice the SvStream vs. XinputStream

VclPtr – initial lifecycle cleanup

- vcl/README.lifecycle
- VclPtr<Foo> xFoo
  - a smart reference counted ptr to widget.
- VclPtr<Foo>::Create(<*args*>)
  - returns a smart reference counted ptr.
- VclPtr<Foo> xFoo(new Foo());
  - Not used – due to unfortunate referencing in constructors ...
- Instead of delete:
  - xFoo.disposeAndClear();
  - dispose can be called multiple times ...

VclPtr & reference loops

- VclPtr<WorkWindow>
- disposeAndClear
  - breaks reference cycles
  - disposeAndClear called on all child references.
- WorkWindow::dispose
  - Implementation – clears all un-owned references eg. mxParent.
  - disposeAndClear's owned references eg. mxChild
- Post-dispose widgets should still 'work'.



Questions / conclusions

- VCL is a 20+ year old toolkit
- The code-base is no worse than can be expected
- Everything needs some love & understanding
- No reason why we can' t do radical things with the API
- Things are improving over time

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