LibreOffice: 
Hermenutical keys to a complex code-base #2

Michael Meeks <michael.meeks@suse.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 #2

- 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
Chunk #2 – grokking the code
Sal pieces
Strings ... include/rtl/ustring.hxx ...

- Three important string classes – but converging on two
- New sal / immutable strings:
  - ref-counted
  - OUString – UTF16, 32bit lengths
  - OString – unspecified 8bit encoding, 32bit length
- String ('UniString')
  - Internally re-uses OUString data structures
  - old 'mutable' string – but uniquifies on change
  - 16bit length enforcement on underlying 32bit len
  - UTF16 – (if you're lucky)
- Writer's limit of 64k chars per paragraph: is here.
Strings … constructing & mutating

- OUStringBuffer
- `include/rtl/ustrbuf.hxx`
- Used to construct strings, concatenate them etc.
- steal to an OUString with 'makeStringAndClear()'
- Conversion OUString ↔ String is fast
- Construction from const char foo[N] is implicit
- String – required for translation:
  - `String(ResId(STR_FOO))`
  - ResId etc. lives in `tools/` ie. high above `sal/`
- `include/comphelper/string.hxx` – lots of good helpers.
- Debugging … - python needed …
Strings … 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 US val:
  ```c
  String STR_ACCESS_DOC_NAME
  {
      Text [ en-US ] = "Document view";
  }
  ```
- `sw/source/core/access/accdoc.cxx`:
  ```c
  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
- operator overloads for `<<` and `>>` [urgh!]
- `udkapi/com/sun/star/io/XinputStream.idl`
- UNO stream impl. - as implemented by UCB, and package code.
- `include/unotools/streamwrap.hxx`
- Convert SvStream ↔ UNO
VCL ...
Visual Class Libraries (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
LibreOffice is fundamentally single threaded

“the” one big lock: is the 'SolarMutex'

This is recursive

'Application::Yield' VCL (or Application::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

Unfortunately VCL is explicitly lifecycle managed:
- New / delete – which causes some problems ...
VCL event emission ...

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

`vcl/inc/salframe.hxx`

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

- Call the callback:
  ```c
  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`

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

**User does:**

```cpp
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

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

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

- Invalidate → Idle → re-render
  - Wait for the app to be ready to render

```cpp
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.hxx` `ImplUpdateDisabledBmpEx`
- 'Image' – class wraps this – giving a cut-out of an image-strip (obsolete)
- All Image/Bitmap/BitmapEx primitives are pImpl + ref-counted
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
VCL: Layout / graphical look ...

- Well documented elsewhere
- Recommend presentations from Caolan on this and/or his write-ups:
- Finally we have a UI dialog / toolkit layout approach.
- Hopefully in the end this will lead to smart pointers everywhere and an end to lifecycle issues.
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