DrawingLayer:

What Should You Know about It

Jan Holesovsky <kendy@collabora.com>

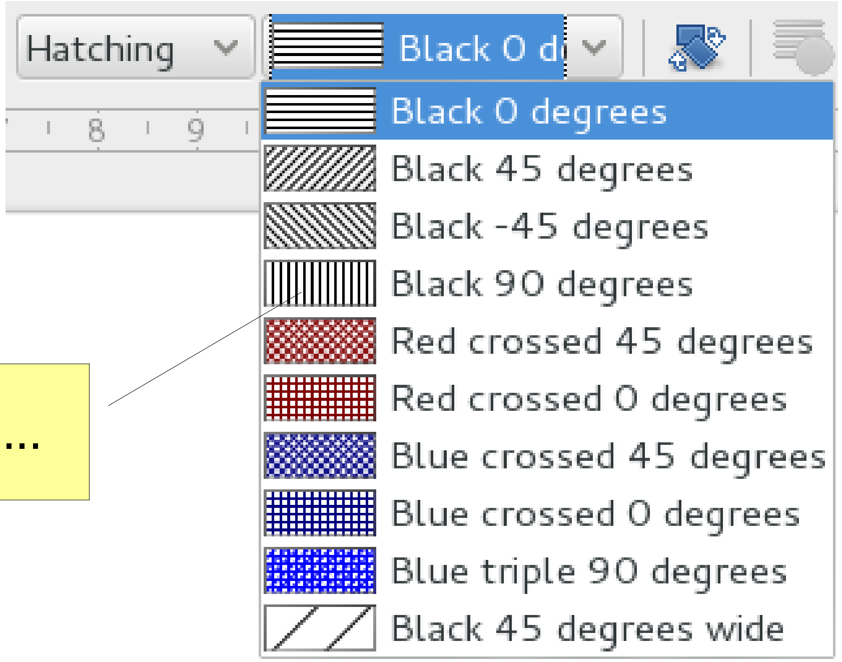
kendy, #libreoffice-dev, irc.freenode.net

What is DrawingLayer

- (Another) drawing infrastructure in LibreOffice
- One man show: Armin Le Grand
- No unit tests
- Yet, important part of LibreOffice
    - Images, drawing objects, antialiasing, ...

Example of DrawingLayer Use

- Draw previews of hatching
  - The hatching itself
  - And a black rectangle around it



Hatching

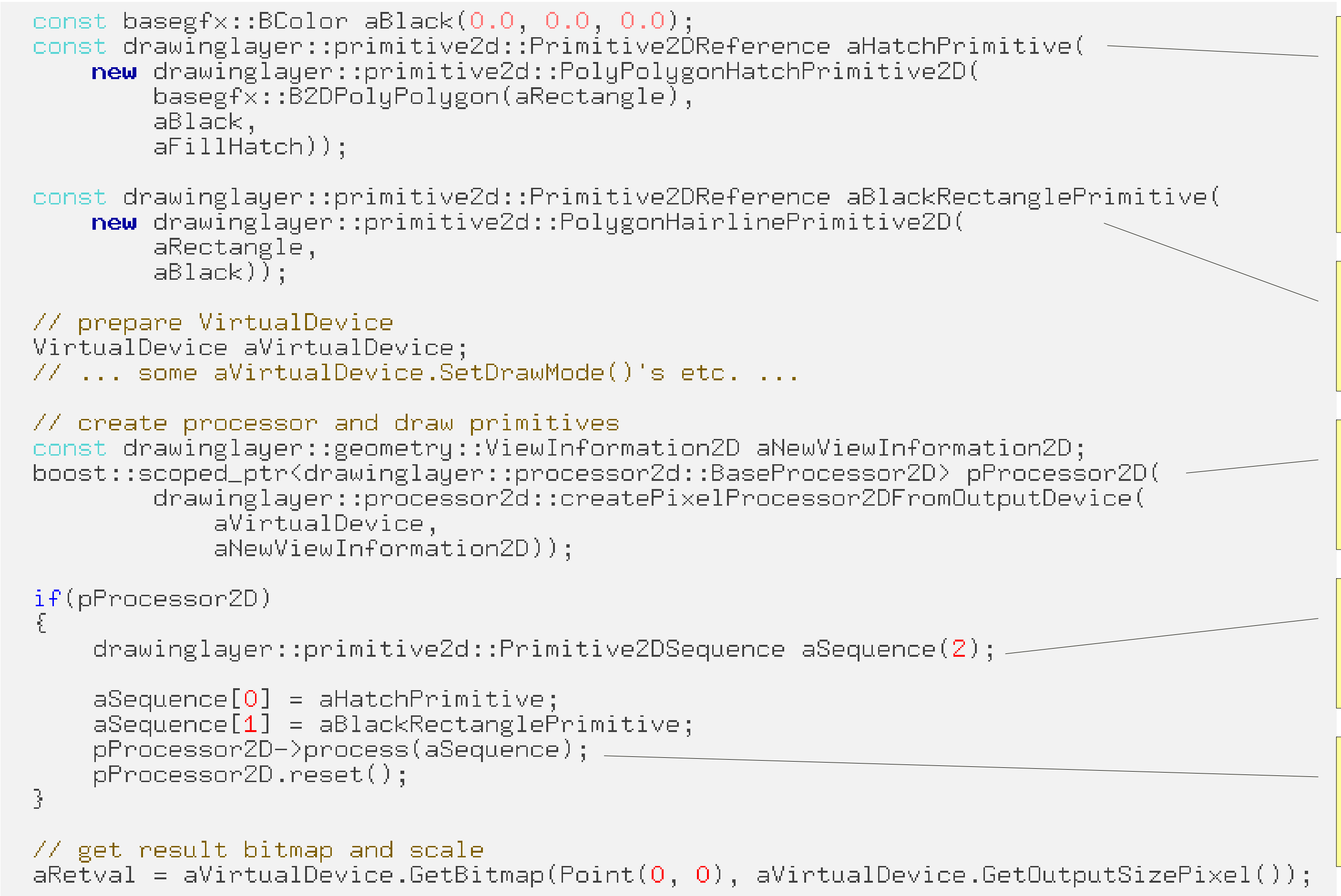
Black 0 d

- Black 0 degrees
- Black 45 degrees
- Black -45 degrees
- Black 90 degrees
- Red crossed 45 degrees
- Red crossed 0 degrees
- Blue crossed 45 degrees
- Blue crossed 0 degrees
- Blue triple 90 degrees
- Black 45 degrees wide

One of these...

The Code that Does That

svx/source/xoutdev/xtabhtch.cxx:121



```
const basegfx::BColor aBlack(0.0, 0.0, 0.0);
const drawinglayer::primitive2d::Primitive2DReference aHatchPrimitive(
    new drawinglayer::primitive2d::PolyPolygonHatchPrimitive2D(
        basegfx::B2DPolyPolygon(aRectangle),
        aBlack,
        aFillHatch));

const drawinglayer::primitive2d::Primitive2DReference aBlackRectanglePrimitive(
    new drawinglayer::primitive2d::PolygonHairlinePrimitive2D(
        aRectangle,
        aBlack));

// prepare VirtualDevice
VirtualDevice aVirtualDevice;
// ... some aVirtualDevice.SetDrawMode()'s etc. ...

// create processor and draw primitives
const drawinglayer::geometry::ViewInformation2D aNewViewInformation2D;
boost::scoped_ptr<drawinglayer::processor2d::BaseProcessor2D> pProcessor2D(
        drawinglayer::processor2d::createPixelProcessor2DFromOutputDevice(
            aVirtualDevice,
            aNewViewInformation2D));

if(pProcessor2D)
{
    drawinglayer::primitive2d::Primitive2DSequence aSequence(2);

    aSequence[0] = aHatchPrimitive;
    aSequence[1] = aBlackRectanglePrimitive;
    pProcessor2D->process(aSequence);
    pProcessor2D.reset();
}

// get result bitmap and scale
aRetval = aVirtualDevice.GetBitmap(Point(0, 0), aVirtualDevice.GetOutputSizePixel());
```

Creation of the Hatch Primitive (to add to a kind of display list, to render later).

Creation of the Hairline Primitive (rectangle)

Processor to render the “display list” later.

The “display list”.

The rendering itself.

Primitives

- Classes that represent what should be drawn
- 'Basic 2D primitives':
  - BitmapPrimitive2D (bitmap data)
  - PointArrayPrimitive2D (single points)
  - PolygonHairlinePrimitive2D (hairline curves/polygons)
  - PolyPolygonColorPrimitive2D (colored polygons)

Decomposition

- Many more Primitives are created by combining the Basic primitives
- These have a defined 'decomposition', a way to break down to a combination of the Basic primitives.
- Example: PolygonMarkerPrimitive2D
  - Two colored marker hairline (to be drawn around objects etc.)
  - Decomposed to two PolyPolygonHairlinePrimitive2D's (one that has 1st and the other 2nd color)

Processors

- Processor is a class that goes through the hierarchy of the Primitives, and renders it some way
  - By hierarchy I mean the hierarchy that is implied by the various decompositions
- Various processors
  - VclPixelProcessor2D – renders to the screen
  - VclMetafileProcessor2D – renders to the VCL metafile
  - And other, with special purpose(s)

How to Implement a new Primitive

- [Uh, what does that mean? ;-)]
  - How to implement a new type that 'draws something' – like eg. soft shadows, etc.
- Create an ancestor of BasePrimitive2D (or of its ancestor if it fits the purpose better)
  - And assign it an ID [in drawinglayer_primitivetypes2d.hxx]
  - Implement its decomposition [virtual Primitive2DSequence create2DDecomposition(...)]
- Extend the (various) processor(s)
  - If you need more than relying on the decomposition

Main DrawingLayer Uses

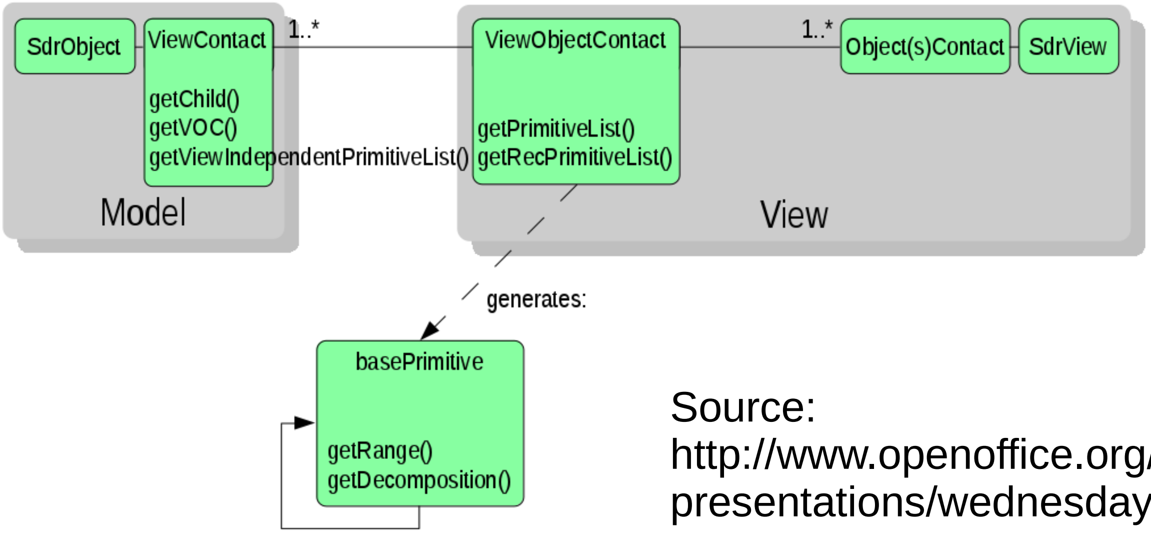
- SdrObject(s) (rectangles, Circles, predefined shapes etc.)
- Selections
- Various smaller cases to 'just draw something'
  - Draw to a virtual device, and use the resulting bitmap
- Custom widgets (like the Header / Footer indicator button)

SdrObject(s)

- Represent the shapes you can draw in LibreOffice
  - Various classes derived from SdrObject (like SdrTextObj [via SdrAttrObj], SdrRectObj, SdrCircObj)
  - Builds quite some additional complexity on top of the DrawingLayer itself

SdrObject(s) + DrawingLayer

- SdrObjects have been reworked to use DrawingLayer
  - Attempted to move to Model/View/Controller, but at the moment only Model and View is there:



Source: http://www.openoffice.org/marketing/ooocon2006/presentations/wednesday_g11.odp

SdrObject(s) + DrawingLayer II

- Own Primitives for SdrObjects in svx/source/sdr/primitive2d
- The ViewContact / ViewObject / ViewObjectContact in svx/source/sdr/contact
  - Decomposes the SdrObjects, and does all sort of operations on them

DrawingLayer Problems

- In way too many cases, the Processor(s) end up drawing to an intermediate bitmap
  - which may be inevitable, but...
    - the primitives have no knowledge of the underlying device,
      - which means there is no way to cache the intermediate results, as the Processors live shortly
- Complexity of SdrObjects
  - Apparently this is half-way of a rework, but who knows if that'll ever finish – so don't be afraid to fix or rework there stuff yourself
- ...
- But the problems are all fixable, of course ;-)
  - Please help extending the {drawinglayer,svx}/README

Thank You for Your Attention!

LibreOffice from COLLABORA