EMF+ - Rework
How to abstract VectorData Imports

CIB SOFTWARE GMBH

LIBREOFFICE CONFERENCE ROME

OCTOBER 11TH, 2017

Armin.Le.Grand@cib.de

ROME - IT - MMXVII
CONFERENCE

Agenda

- Motivation
- How is EMF+ used at all currently?
- Current Problems
- Basic Goals
- Where to place an EMF+ Interpreter
- How to implement that
- What to do with existing importers
- Preservation of original Data
- Sticking it all together
- State of this Change
- Missing Steps
- Future of this Abstraction

Motivation

CIB

- Long existing problems with the WMF/EMF/EMF+ formats from MS
  - We import to our own Metafile format
  - It is similar to MS stuff, both are recording of Paint-Actions
  - Similar because VCL is an (old) abstraction of WinGDI with the intention to run system-independent
  - WMF: Initial Format, encapsulates all commands of old WinGDI
  - EMF: ‘E’nhanced MetaFile, similar, adopted to newer APIs
  - EMF+: Based on EMF, binary data chunks added as ‘Comment’ Actions (exclusive or replacing parts of EMF), backward compatible

Motivation

- All those formats get interpreted at load time and converted to our own Metafile format. The set of commands is not and never was complete ;-(
- There is already code to read and hold that EMF+ binary data, a flag is set to mark the Metafile to contain EMF+
- No interpretation of EMF+ binary data in this place
- Normal Metafile usage will not use those parts (e.g. ::Play())
- Not included in any transformation (::Move(), ::Scale(), …)
- Not used in ‘Break’ action
- Save/Load works due to binary data in comments being added

Motivation

- University of Dresden had (as many others) issues with EMF+ e.g. containing visualizations of Molecule Data
- They looked for help and contacted CIB, so a cooperation could be established
- There exist numerous bugs in that area anyways, so this was a good opportunity to reorganize things

How is EMF+ used at all currently?

- There is an UNO API based renderer that uses Canvas functionality
- Directly interprets our own Metafile and included EMF+
- Directly renders these to Canvas
- Used in Slideshow and EditViews
- Triggered when painting a Graphic that has the ‘EMF+’ Flag set

Current Problems

- No separation of interpretation and rendering (re-usability, mixed code)
- A Subset of GDI+ is implemented
- Problems with OutDev target type Metafile (PDF, Printing, …)
- To extend, broad knowledge of two complex areas is needed
- Not all Canvas implementations are complete
- Quality of Canvas implementation vary widely on systems
- Setup of Canvas until rendering starts can take quite some time
- Resource usage is not controllable, a source of many crashes (try to zoom in EditView)

Basic Goals

- The idea is to
  - Separate interpretation and rendering
  - Make contained graphic definitions reusable
    - Further processing (break, change attributes, edit, …)
    - Use in EditViews for visualization in a standard way
    - Use it for processing (PDF, Print, …)

Where to place an EMF+ Interpreter

CIB

- Can best be done using Primitives to fit seamlessly in the existing tool chain (also UNO API capable)
- Simplest starting point is the existing MetafilePrimitive
  - Encapsulates a Metafile
  - Decomposes on demand and buffers
  - Creates sequence of Primitives
  - In use for quite some time (stable, proved)

So the obvious place for adding an EMF+ interpreter is inside the decomposition of the MetafilePrimitive. This will be the central and only place where Metafiles and EMF+ sub-contents will be interpreted

How to implement that

- Can be speed up by reusing as much of the existing interpreter as possible
- Interpretation binary EMF+ parts is the same, data extraction is ‘prepared’
  - Created data has to be adapted to primitive definitions
  - Contained transformations and their relations have to be reorganized (metafile is linear, primitives have tree structure)
  - Embedding of ClipRegions needs to adapted in a similar way

What to do with existing Importers

- Nice-to-have, but not urgently needed
- Isolate existing WMF/EMF importers
  - Get them out of intensively used modules
  - Convert them to a BlackBox (no one needs their internals)
  - Move return value to Primitives
    - Currently a single MetafilePrimitive in a sequence<primitive>
- Took that opportunity, done that ;-)

Preservation of original Data

CIB

- Loading WMF/EMF/EMF+ imported to own Metafile
- On Save time, our own Metafile format is written
- Original Data is lost forever
  - No way to profit later from enhancements of the importer(s)
  - Is the User aware that the original data is not part of the created ODF…?
- But wait – I did something similar for SVG import!
  - SVG gets loaded, original data is preserved, decomposed on-demand, original data saved in ODF, even offers BitmapEx fallback for part of office that can not yet use primitives
- So – why not use that and abstract it? So I did exactly that ;-)

SVGGraphicData → VectorGraphicData

- Abstracted the proved SVG mechanism to all VectorGraphicFormats
- All uniformly handled as described
  - Preserve original Data: Kept and stored as Graphic in ODF
  - Interpreted on-demand (use migrated importers), buffered
  - Optional BitmapEx available
  - Original accessible and exportable anytime (context menu)

Sticking it all together

- Isolation of existing WMF/EMF/EMF+ Importers, adapted return data type to Primitives (MetafilePrimitive)
- Extension of the MetafilePrimitive Decomposer to EMF+ reusing and adapting the existing interpreter
- Unified handling of all VectorGraphicFormats with preservation of the original Data

State of this Change

- All of the above implemented
- Some parts by me, some by Patrick Jaap from University of Dresden
- I did initial transfer of modules, make it principally work
- Patrick adopted EMF+ interpretation
- He is fixing bugs and improving EMF+ interpretation
- He is taking over care for that part
- Good example of a well-working cooperation ;-)

Missing Steps

- Slideshow needs to be adapted to use Primitives
  - Currently based on Metafiles
  - Planned for some time, but was never done
  - Needed to remove the direct renderer completely

Future of this Abstraction

CIB

- Imagine to extend this to all GraphicDataTypes
  - Include Pixel formats
  - All would preserve original data in ODF
- Imagine to add a central/automatic/hidden organizing Instance that
  - Manages that data based on size/usage/timestamps
  - Throws away unused sequences<primitive>
  - Throws away/streams to TempFiles unused BinaryData (PGPed)

This would lead to a central, automated mechanism that would completely replace all the existing SwapIn/SwapOut stuff that is old and unstable anyways :-)

Just dreaming :-)