UNO uses, mis-uses, etc

What it is

- Allegedly “Universal Network Objects”
  - https://www.openoffice.org/udk/common/man/uno.html
- Influenced by COM, CORBA, Java RMI
- Supports exceptions

What it probably isn’t

- We have “command urls” which can be “dispatched” like “uno:InsertGraphic to do stuff
- Got uno in their name, so “uno commands”
  - Probably makes things somewhat murky wrt. uno
  - There was “slot:number” where number was hardcoded in .hrc files, hopefully all gone in favour of the symbolic names

Motivations (group a)

- “[enable] third parties to write components which could be integrated into the office suite.
- “StarOffice components could not be used outside of StarOffice.”
- “There was a requirement to integrate components from other object models like CORBA, COM/DCOM, or Java into StarOffice.”

https://www.openoffice.org/udk/common/man/uno_the_idea.html

What you can do with it #1

- Supports Remote calls
- Run soffice –accept=socket,…
- Write another program that calls methods on its local uno proxy objects that are transported to the soffice server and executed there and results transported back

What you can do with it #2

- Use it to bridge between languages
- Extensions in multiple languages, java, python, starbasic, etc
- Most of the core is in c++, but there are some python bits by default in the desktop case f.e. emailmerge etc
- Within the same environment, e.g. the typical LibreOffice in-process case then the calls are typical direct virtual calls on the destination object

Practical bits

- IDL files
  - udkapi dir
    - uno development kit, core bits needed to do anything. stdlib.h of uno
  - offapi dir
    - Office Suite apis, everything else basically
  - oovbaapi dir
    - Vba compatibility stuff separate for some reason or other

UNO APIs

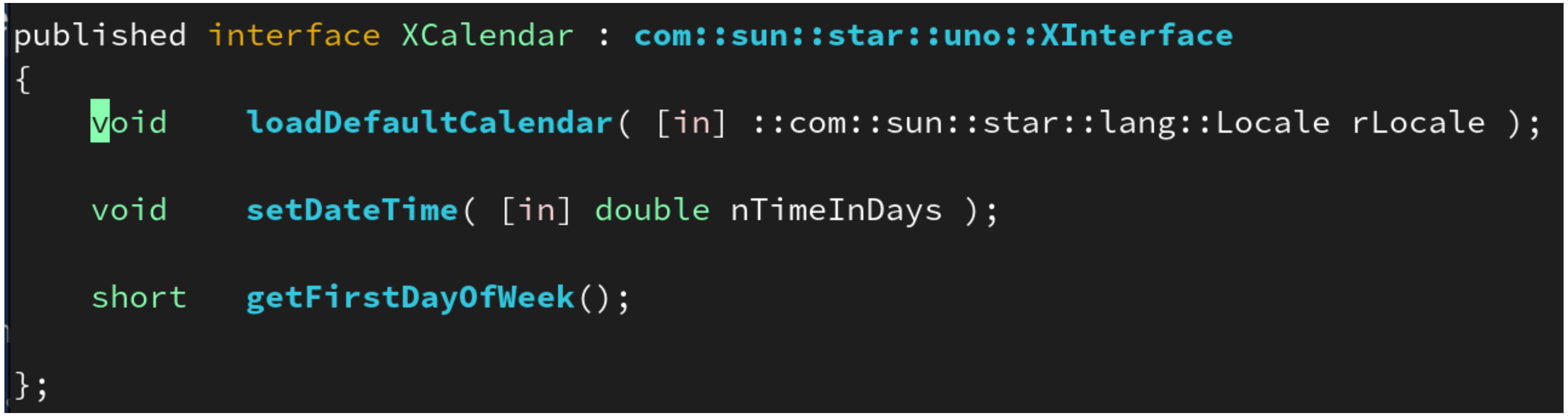
- “Published” ones are inflexible
  - enums cannot be extended
    - published enum TextureKind
  - “constants” can
    - published constants NumberingType
- Unpublished ones are nominally free for all
  - But arbitrary change is rare

UNO Types

- Various obvious types, byte, boolean, double, float, short, void
- Semi-obvious, “string” maps to c++ rtl::OUString, “any” to com:sun:star:uno::Any
- “long” maps to c++ sal_Int32
- “hyper” maps to c++ sal_Int64

Sample

- IDL gets compiled so at runtime the typeinfo is available as to how to call, pass params, to method

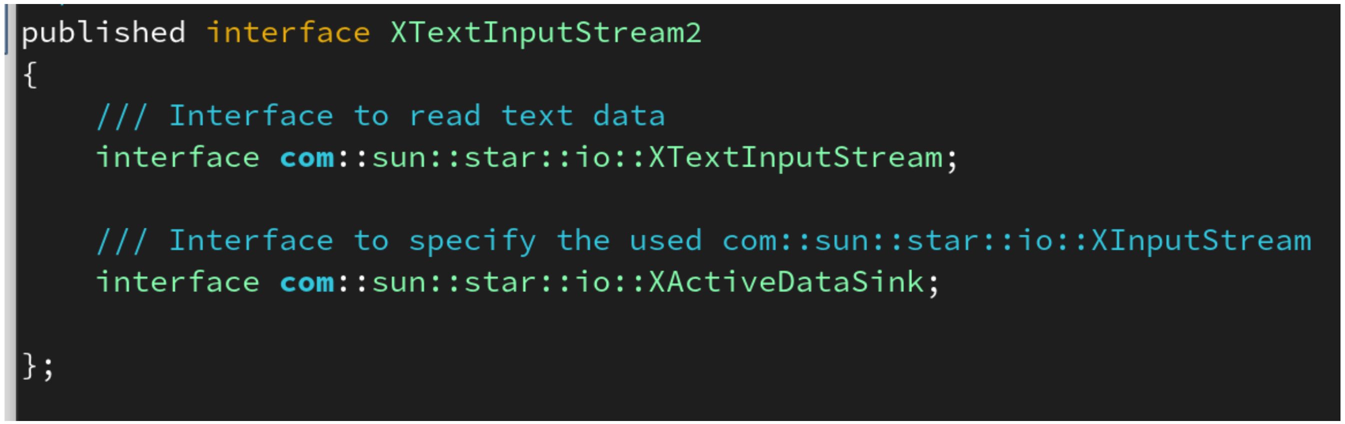


```
published interface XCalendar : com::sun::star::uno::XInterface
{
    void    loadDefaultCalendar( [in] ::com::sun::star::lang::Locale rLocale );

    void    setDateTime( [in] double nTimeInDays );

    short   getFirstDayOfWeek();

};
```

- So, C++ to Python/Java/Remote calls local stub object virtual method, which forwards to a generic arch-specific function providing typeinfo, index of method etc

UNO API Extensibility Sample

- Can create a new better interface that provides the old interface
- Plus the stuff the old interface should have had



```
published interface XTextInputStream2
{
    /// Interface to read text data
    interface com::sun::star::io::XTextInputStream;

    /// Interface to specify the used com::sun::star::io::XInputStream
    interface com::sun::star::io::XActiveDataSink;

};
```

Build misery circa 2000

- Two day+ builds from scratch
- Releng provides pre-built libs for everyone else to link against
- A dev checks out code from cvs, builds e.g. just sw, but links against those pre-built libs for everything else. Maybe needs new vcl api, so wants to build vcl + sw.
- So concept of a “compatible” build where additional non-virtual methods can be added to vcl headers, etc so rebuild against those headers still links and works with this weeks pre-built libs without the changes
- “Incompatible” rebuild at w/e or so

Motivations (group b)

- “There are a number of base projects (Tools, Streams, Visual Class Library, Framework, etc.) The higher projects, such as the word processor, calc, etc., use the classes of these base projects. *After a change of some of these classes, for example, a new member or virtual method is added, the entire office suite needs to be rebuilt*. This takes two days, if no problems occur.”
- “The API of the base project, with a few exceptions, is not well documented. The base projects grow with the requirements of the higher projects.”
- “The projects dependencies are complicated and difficult to understand. Before making changes to an API we need to know, exactly, which projects are affected.”

https://www.openoffice.org/udk/common/man/uno_the_idea.html

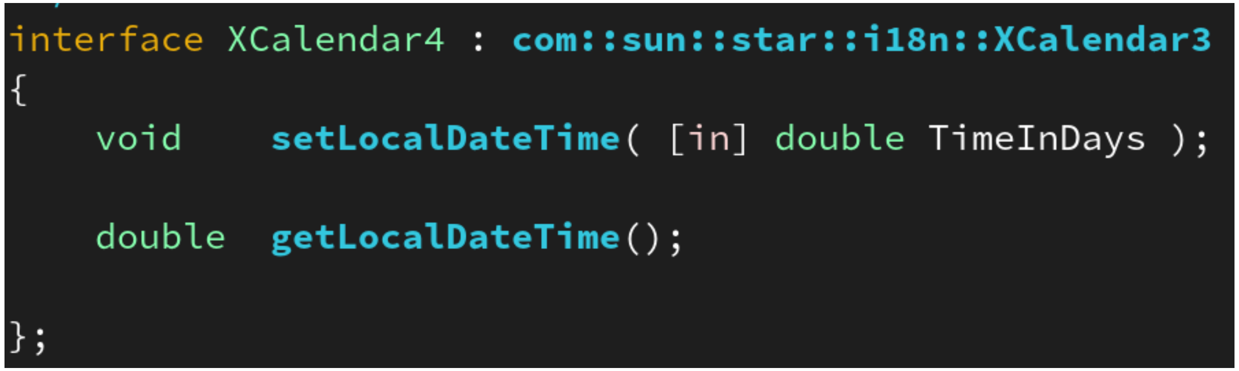
Motivations (group b)

- “There is a mechanism which enables a new method to be added to an existing class: this is done with interface technology. Only interfaces are exposed to other projects. To add a new method, you only have to add a new interface. So, new methods can be added to an existing old class, and then the other projects can use these new features. There is a migration path to the new API.”
- “Use of an IDL-language to describe our interfaces and the functionality of components. To do this on an abstract level, normally, the documentation is better and the API is not implementation dependent.”
- “To reduce the build dependencies of a specific component only interfaces are used to communicate with other components and the base libraries. In this case, the dependencies are flat.”

https://www.openoffice.org/udk/common/man/uno_the_idea.html

Example

- So if you used f.e. XCalendar via uno instead of its concrete impl exposed via c++ headers, and then need to add something new to it
- Can rebuild offapi and module implementing new version without breaking everything using old version
- Could replace impl with python :-)



```
interface XCalendar4 : com::sun::star::i18n::XCalendar3
{
    void    setLocalDateTime( [in] double TimeInDays );

    double  getLocalDateTime();

};
```

If we added uno today?

- Periphery entry point for extensions, scripting API, remote call support etc, surely yes.
- But for widespread internal use?
  - Rebuild times orders of magnitude less
  - Incompatible builds not a thing
  - A handy go-to for virtual base class implementation
  - A handy go-to for circular dependencies