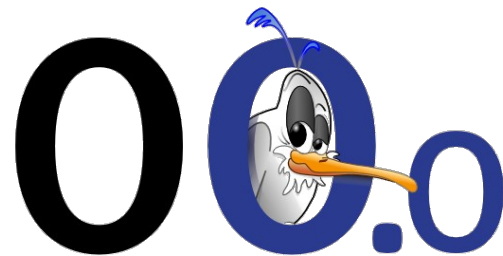




Introduction to OpenOffice.org scripting features



Laurent Godard

Agenda

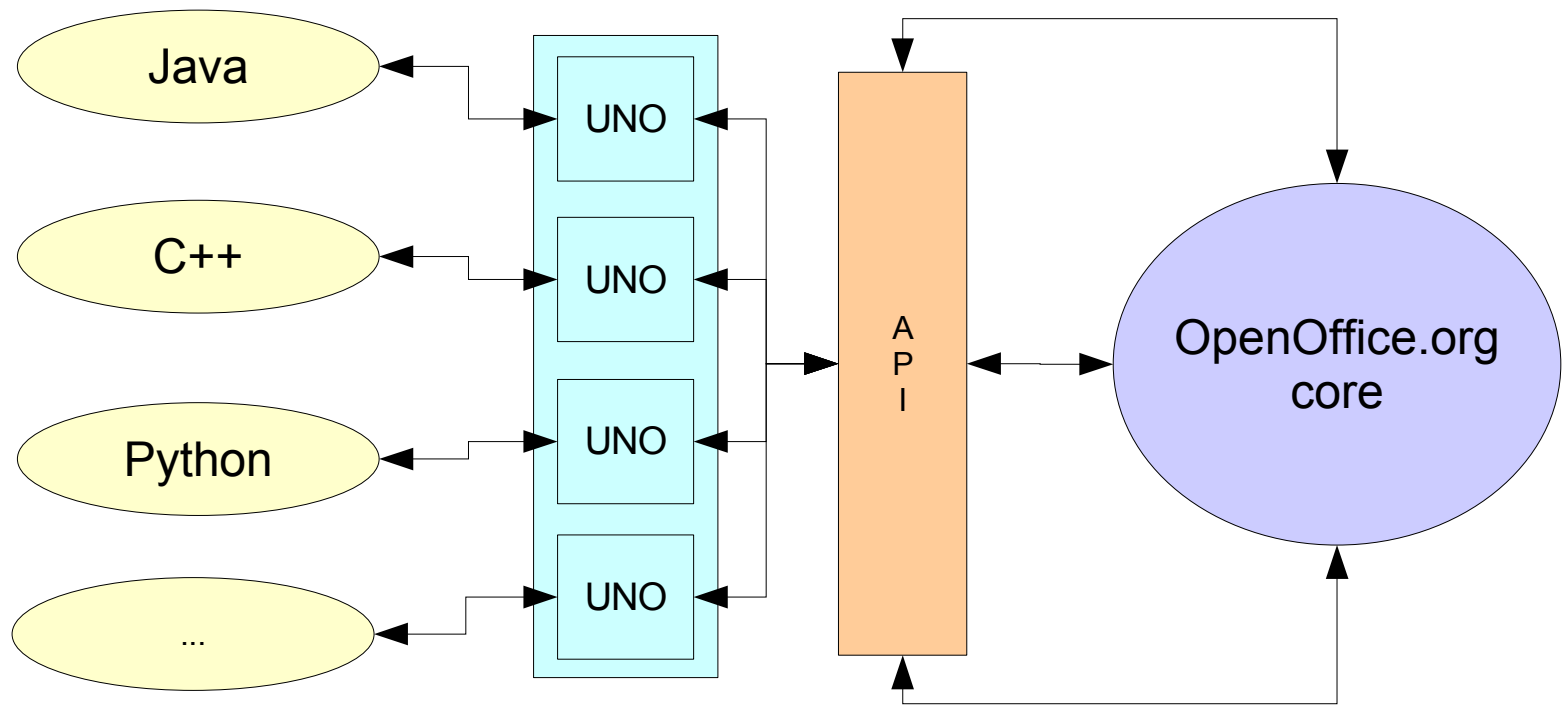
- Context
 - What is scripting ?
- Tools
 - What does OpenOffice.org provide ?
- Scripting OpenOffice.org
 - Presentation
 - Basic
 - Python

Scripting

- Possibility to pilot OpenOffice.org elements for our own needs
- Access its API/features by code
- This can be done
 - From a code inside OpenOffice.org : macros
 - From outside OpenOffice.org : connection line
 - `ooffice "-accept=socket,host=localhost,port=2002;urp;"`
 - `setup.xcu <prop oor:name="ooSetupConnectionURL">`
 - Network connection on localhost and even network
 - From components deployed at OOo level : Add-ons
- Once a connection is set up, scripting is almost the same

> Context and definitions

- Many languages
 - Java, C++, COM (VB, delphi), .Net, ...
- Language uses OOo API via UNO bridges



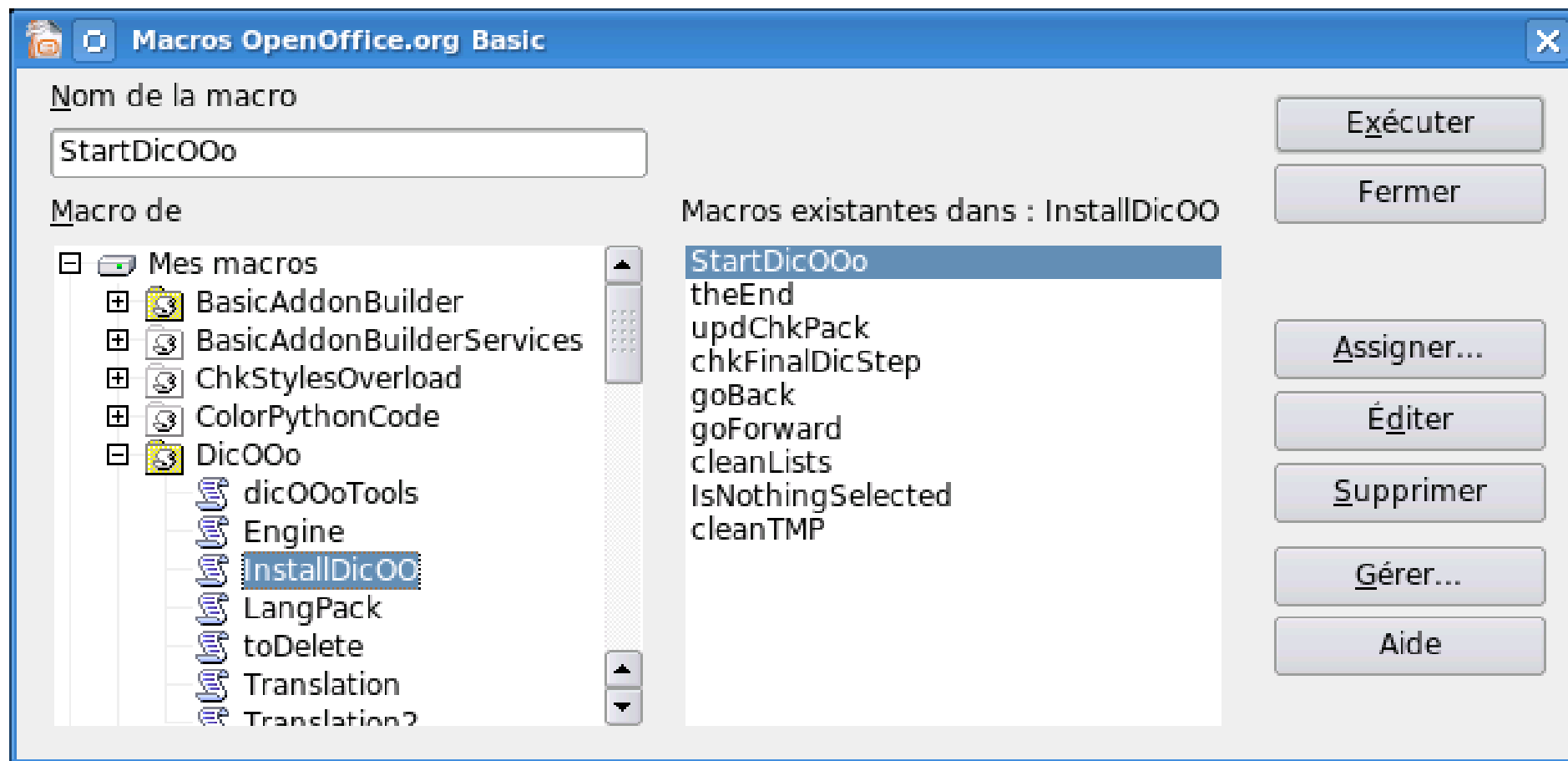
Definitions

- **API : Application Programming Interface**
 - Access to objects, properties and methods of OOo core
 - No need to know OOo source code but only the API it exposes
- **SDK : Software Development Kit**
 - Toolbox dedicated to developers to use OpenOffice.org API and build their programs
- **UNO : Universal Network Object**
 - Description of objects that allow to communicate with API no matter the localization of objects
 - The bridges for different languages allow its use in heterogeneous environments

- IDE
 - Integrated development environment
 - OOO dialog builder graphical tool
- SDK
 - What does it contains as documentation ?
 - How to browse it ?
- Helpers
 - What are the coding facilities introduced in scripting languages ?
 - Useful developments by community

Macro management

- Hierarchical organization



The basic IDE – code

- Syntax coloration
- Step by step debugging tool
- Call stack
- Variable values and object explorer

The screenshot shows the OpenOffice.org Basic IDE window titled "Mes macros et boîtes de dialogue.Standard - OpenOffice.org Basic". The menu bar includes "Fichier", "Édition", "Affichage", "Outils", "Fenêtre", and "Aide". The toolbar contains various icons for file operations and development. The main editor displays the following BASIC code with syntax highlighting:

```
REM ***** BASIC *****  
  
Sub Main  
    print "My first macro"  
    print addNumbers(1,2)  
End Sub  
  
function addNumbers(a,b)  
    addNumbers = a + b  
end function
```

Below the editor, there is a "Témoin" (Watch) window and an "Appels" (Calls) window. The "Témoin" window shows a table with the following data:

variable	Valeur	Type
a	1	Variant/Integer

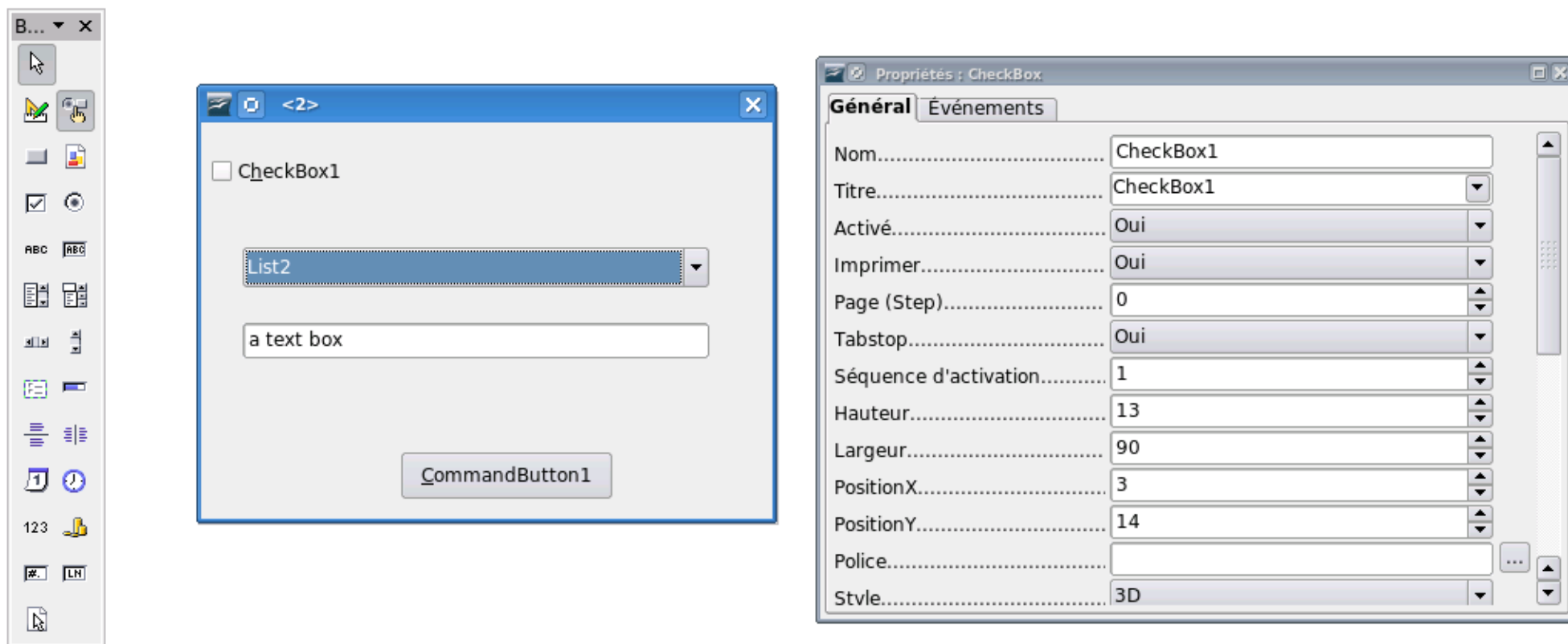
The "Appels" window shows the call stack with the following entries:

- 0: addNumbers(a=1, b=2)
- 1: Main

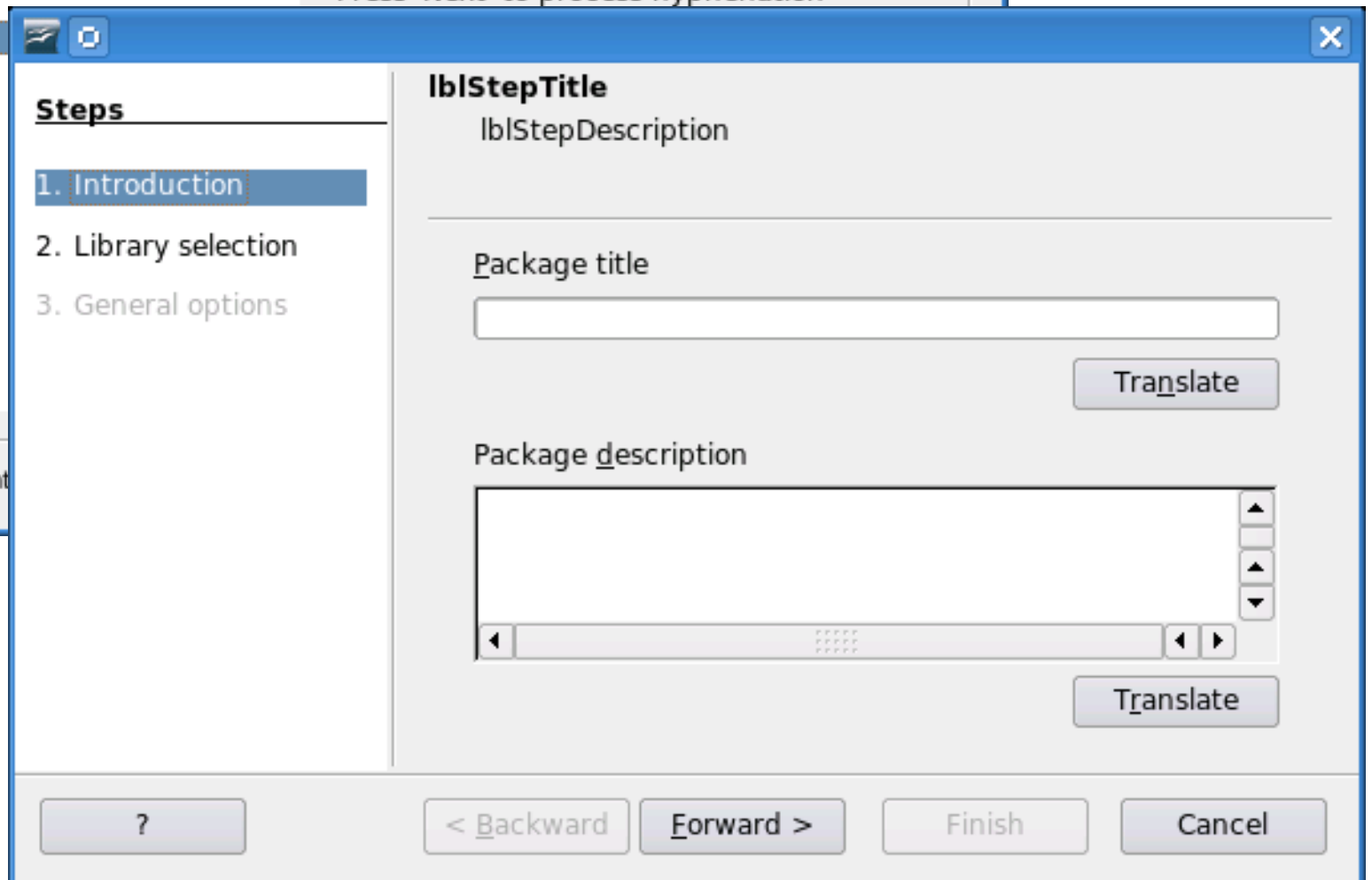
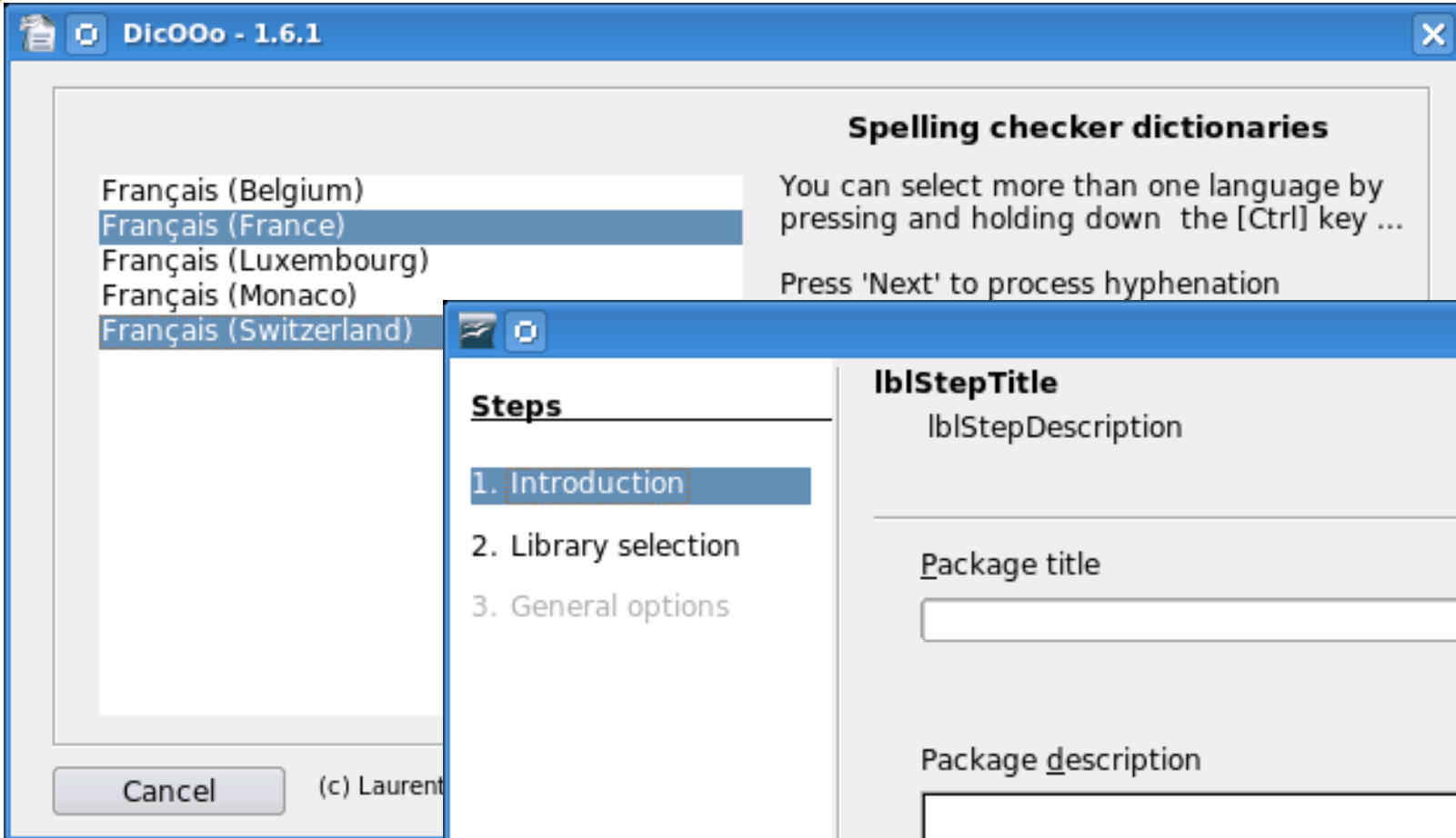
The status bar at the bottom indicates the current file is "Mes macros et boîtes de dialogue.Standard.Module9" at "Li 7, Col 1" with the "INS" key highlighted.

The basic IDE – dialogs

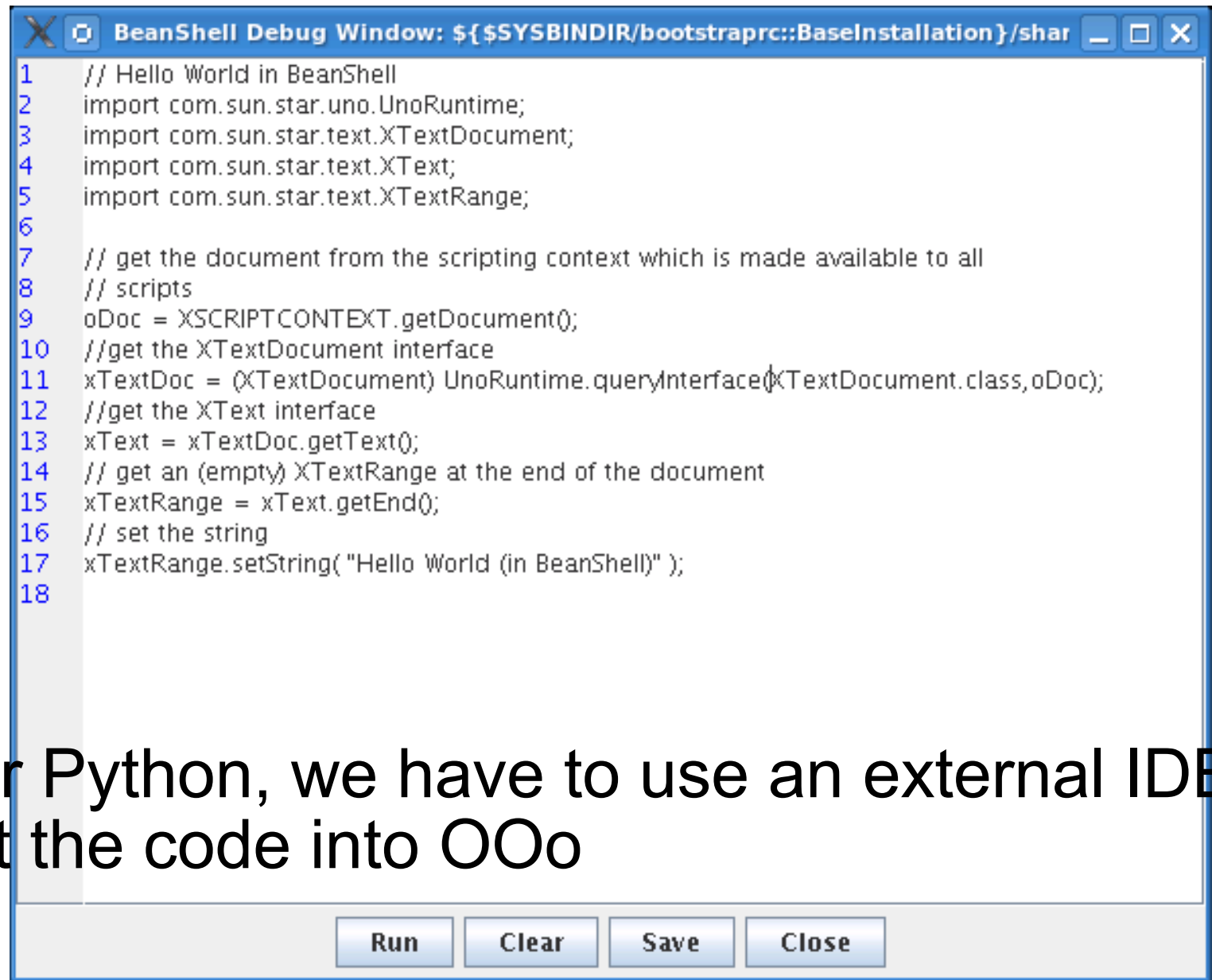
- Build your own dialogs with integrated tools



Complex dialogs



IDE for other languages



```
BeanShell Debug Window: ${$SYSBINDIR/bootstraprc::BaseInstallation}/shar
1 // Hello World in BeanShell
2 import com.sun.star.uno.UnoRuntime;
3 import com.sun.star.text.XTextDocument;
4 import com.sun.star.text.XText;
5 import com.sun.star.text.XTextRange;
6
7 // get the document from the scripting context which is made available to all
8 // scripts
9 oDoc = XSCRIPTCONTEXT.getDocument();
10 //get the XTextDocument interface
11 xTextDoc = (XTextDocument) UnoRuntime.queryInterface(XTextDocument.class,oDoc);
12 //get the XText interface
13 xText = xTextDoc.getText();
14 // get an (empty) XTextRange at the end of the document
15 xTextRange = xText.getEnd();
16 // set the string
17 xTextRange.setString("Hello World (in BeanShell)");
18
```

Run Clear Save Close

- Javascript
- Beanshell
- Nothing for Python, we have to use an external IDE and import the code into OOo

> First Macro

- Hello Message

```
sub HelloMessage
    aName = inputBox("What is your name ?")
    msgbox "Hello " + aName
end sub
```

- No OOo interaction
- Basic language compliant

Going deeper

- Need to access OOo objects to create business helpers inside the office suite
- Where to find them ?
 - The SDK provides many documentation on the API
 - Developers guide
- Related OpenOffice.org projects
 - <http://extensions.openoffice.org>
 - <http://api.openoffice.org>
 - <http://udk.openoffice.org>
 - <http://framework.openoffice.org>

The SDK (OOoBasic Use)

- A deployable archive
 - <http://api.openoffice.org>
 - around 30 Mb (100 Mb installed)
 - In english
 - IDL reference
 - Developers guide
 - Examples
 - OpenDocument and OpenOffice.org 1.x file specification
 - Building tools and Java/C++ reference (not needed for basic)
 - ...

IDL Reference

- Synthetic information
 - For a given service and interface, enumerates all properties and methods it offers
 - Some comment lines on each
 - Hyperlink navigation allowing exploration of returned types
- Index and navigation pages
 - Alphabetical
 - Hyperlink navigation allowing deep exploration
 - Hyperlinks between IDL reference and developers guide
- Ideal for finding its way in the API

IDL Reference (II)

Overview [Module](#) [Use](#) [Devguide](#) [Index](#)

NESTED
MODULES

SERVICES

SINGLETONS

INTERFACES

STRUCTS

EXCEPTIONS

ENUMS

TYPEDFS

CONSTANT
GROUPS

:: com :: sun :: star :: text :: textfield ::

module docinfo

Services

ChangeAuthor

specifies service of a text field that provides information about the author of the last change.

ChangeDateTime

specifies service of a text field that provides information about the date and time the document was last changed.

CreateAuthor

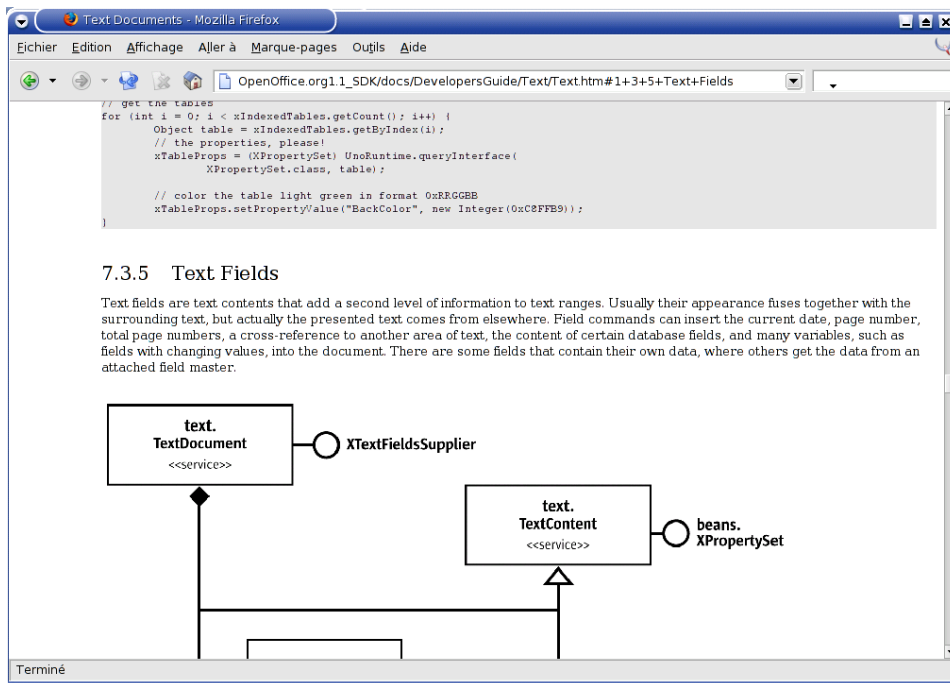
specifies service of a text field that provides information about the author who created the document.

CreateDateTime

specifies service of a text field that provides information about the date and time of the document creation.

Developers Guide

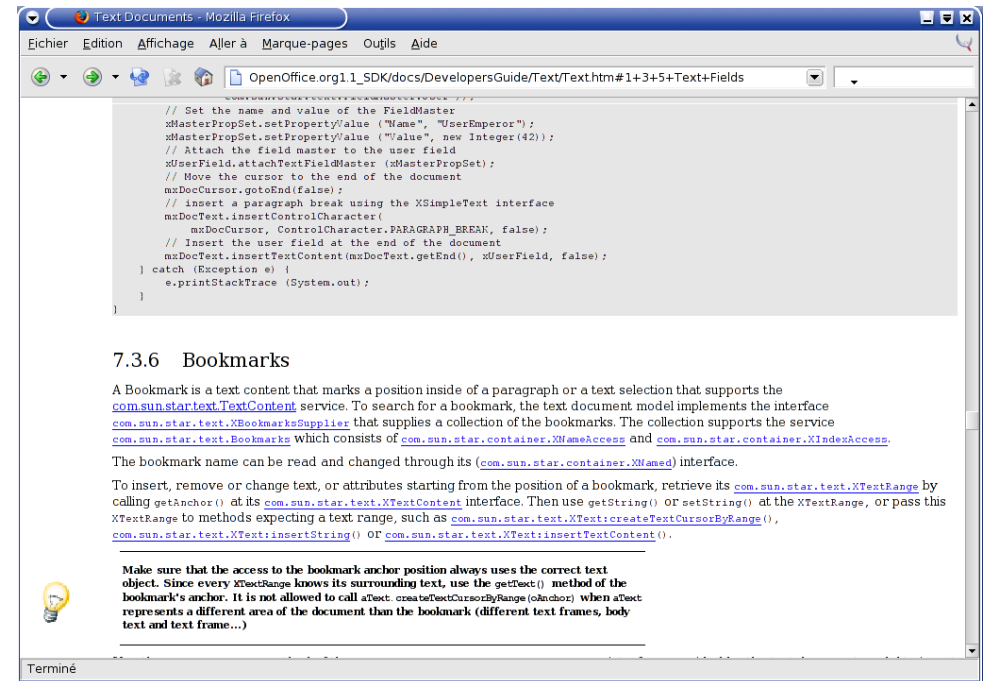
- Full documentation on UNO and API
 - more than 1000 pages
 - HTML or PDF
- Cross hyperlink reference to IDL
- Numerous code examples (Java) and UML diagrams



The screenshot shows a Mozilla Firefox browser window with the address bar pointing to `OpenOffice.org1_1_SDK/docs/DevelopersGuide/Text/Text.htm#1+3+5+Text+Fields`. The main content area displays a code editor with the following Java code:

```
// get the tables
for (int i = 0; i < xIndexedTables.getCount(); i++) {
    Object table = xIndexedTables.getByIndex(i);
    // the properties, please!
    xTableProps = (XPropertySet) UnoRuntime.queryInterface(
        XPropertySet.class, table);
    // color the table light green in format 0xREGGBB
    xTableProps.setPropertyValue("BackColor", new Integer(0xC8FFB9));
}
```

Below the code, the section **7.3.5 Text Fields** is visible. It contains a paragraph explaining text fields and a UML class diagram. The diagram shows two classes: `text.TextDocument` (labeled as a service) and `text.TextContent` (also a service). `text.TextDocument` has a dependency on `beans.XTextFieldsSupplier`. `text.TextContent` has a dependency on `beans.XPropertySet`. Both classes are subclasses of a common base class, indicated by solid lines with hollow triangle heads pointing to the base class.



The screenshot shows a Mozilla Firefox browser window with the same address bar as the previous image. The main content area displays a code editor with the following Java code:

```
// Set the name and value of the FieldMaster
xMasterPropSet.setPropertyValue("Name", "UserEmperor");
xMasterPropSet.setPropertyValue("Value", new Integer(42));
// Attach the field master to the user field
xUserField.attachTextFieldMaster(xMasterPropSet);
// Move the cursor to the end of the document
mxDocCursor.gotoEnd(false);
// insert a paragraph break using the XSimpleText interface
mxDocText.insertControlCharacter(
    mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);
// Insert the user field at the end of the document
mxDocText.insertTextContent(mxDocText.getEnd(), xUserField, false);
} catch (Exception e) {
    e.printStackTrace(System.out);
}
}
```

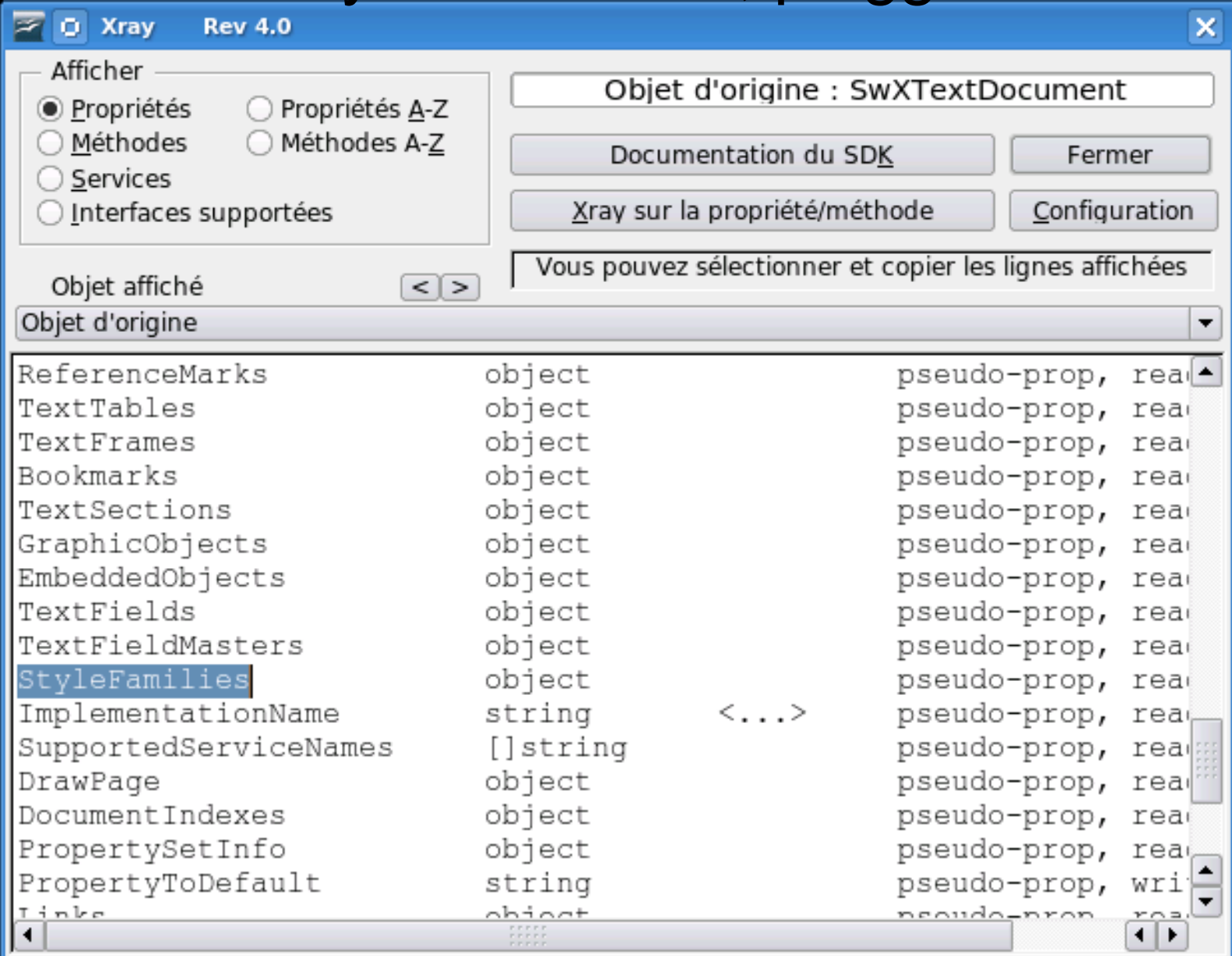
Below the code, the section **7.3.6 Bookmarks** is visible. It contains a paragraph explaining bookmarks and their usage. The text includes several links to service interfaces and methods. At the bottom of the section, there is a lightbulb icon and a note: "Make sure that the access to the bookmark anchor position always uses the correct text object. Since every `XTextRange` knows its surrounding text, use the `getText()` method of the bookmark's anchor. It is not allowed to call `createTextCursorByRange(anchor)` when `atext` represents a different area of the document than the bookmark (different text frames, body text and text frame...)"

OOoBasic coding simplifications

- ThisComponent, StarDesktop
- CreateUNOService, CreateUNOStruct
- ConvertToURL, convertFromURL
- Get and set methods are binded to properties in read and write mode
 - setString, getString methods to String property
- Collections are translated to arrays
 - getByIndex method calls not needed anymore
- Structures and named constant recognized
 - com.sun.star.beans.PropertyValue
- Introspection : dbg_methods, dbg_properties

Xray - OOoBasic

- Navigate recursively into the API, plugged to the IDL



The screenshot shows the Xray Rev 4.0 application window. The title bar reads "Xray Rev 4.0". The window contains a control panel on the left with radio buttons for "Afficher" (Display) options: "Propriétés" (selected), "Propriétés A-Z", "Méthodes", "Méthodes A-Z", "Services", and "Interfaces supportées". On the right, there is a text field "Objet d'origine : SwXTextDocument" and buttons for "Documentation du SDK", "Fermer", "Xray sur la propriété/méthode", and "Configuration". Below the control panel is a section "Objet affiché" with navigation arrows and a text box containing "Vous pouvez sélectionner et copier les lignes affichées". A dropdown menu shows "Objet d'origine". The main area is a list of properties:

Property Name	Type	Value	Access
ReferenceMarks	object		pseudo-prop, read
TextTables	object		pseudo-prop, read
TextFrames	object		pseudo-prop, read
Bookmarks	object		pseudo-prop, read
TextSections	object		pseudo-prop, read
GraphicObjects	object		pseudo-prop, read
EmbeddedObjects	object		pseudo-prop, read
TextFields	object		pseudo-prop, read
TextFieldMasters	object		pseudo-prop, read
StyleFamilies	object		pseudo-prop, read
ImplementationName	string	<...>	pseudo-prop, read
SupportedServiceNames	[]string		pseudo-prop, read
DrawPage	object		pseudo-prop, read
DocumentIndexes	object		pseudo-prop, read
PropertySetInfo	object		pseudo-prop, read
PropertyToDefault	string		pseudo-prop, write
Links	object		pseudo-prop, read

XRay – SDK binding

Interface XStyleFamiliesSupplier - Mozilla Firefox

Fichier Edition Affichage Aller à Marque-pages Outils Aide

file:///opt/openoffice.org2.0_sdk/docs/common/ref/com/sun/star/style/XSt

OK

:: com :: sun :: star :: style ::

interface XStyleFamiliesSupplier

Description

This interface provides access to the style families within the container document.

Developers Guide

[8.4.1 Spreadsheet Documents - Overall Document Features - Styles](#)

Methods' Summary

getStyleFamilies	This method returns the collection of style families available in the container document.
----------------------------------	---

Methods' Details

getStyleFamilies

```
::com::sun::star::container::XNameAccess  
getStyleFamilies();
```

Description

This method returns the collection of style families available in the container document.

See also

[StyleFamilies](#)

Terminé

> First OOo API use

- Convert a document to PDF

```
sub launchMacro()  
    call DocumentToPDF("test.odt","test.pdf")  
end sub  
  
sub DocumentToPDF(source, destination)  
  
    sourceURL = convertToURL(source)  
    sourceDoc = StarDesktop.loadComponentFromURL(sourceURL, "_blank", _  
                                                0, Array())  
  
    destinationURL = convertToURL(destination)  
    dim args(0) as new com.sun.star.beans.PropertyValue  
    args(0).Name = "FilterName"  
    args(0).Value = "writer_pdf_export"  
  
    sourceDoc.storeToURL(destinationURL,args())  
  
    sourceDoc.close(False)  
end sub
```

Packaging as addon

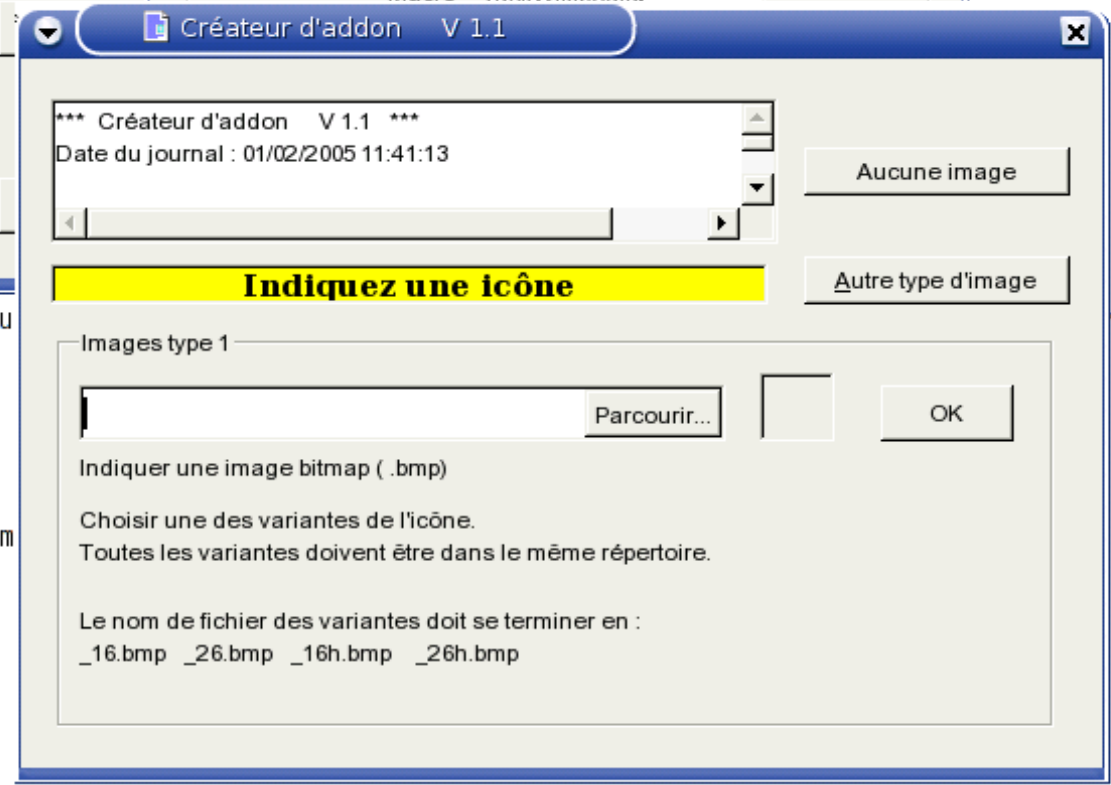
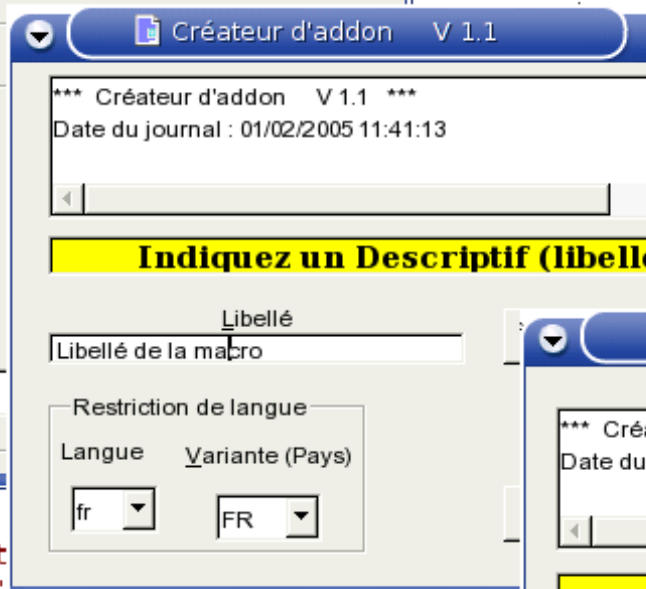
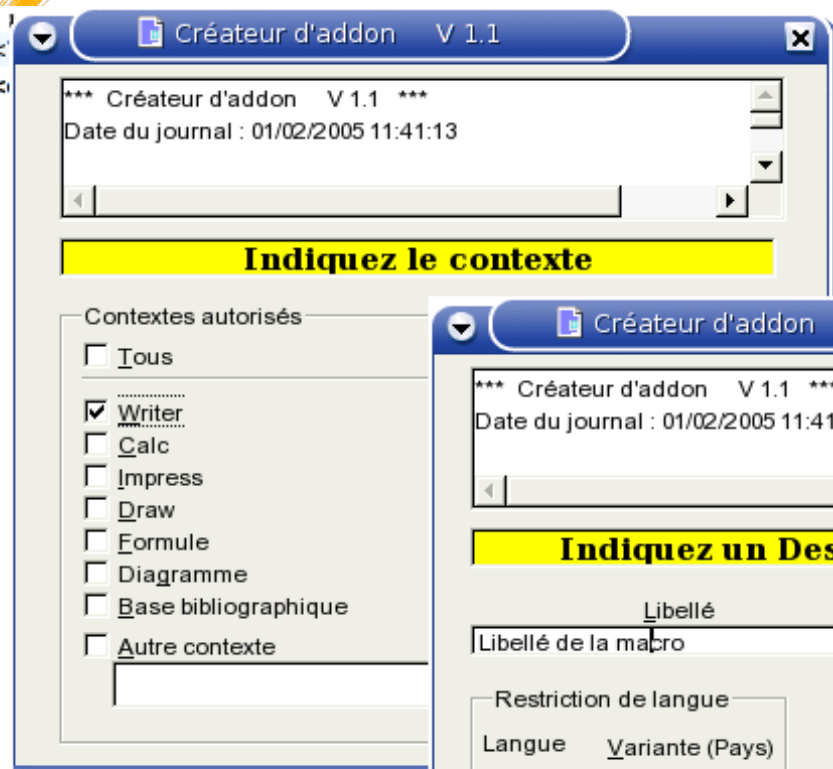
- Can be distributed inside a document but then not integrated to OOo
- Definition of an addon
 - A deployable set of code containing information on its availability and access in OpenOffice.org in a standalone file
 - Compressed file containing code, resources and configuration – a new filename extension defined : oxt (OOo 2.0.4)
 - Developers guide chapter 4
- Paolo's previous year presentation
 - http://marketing.openoffice.org/ooocon2005/presentations/thursday_d4.pdf

addons.xcu

- OOo GUI integration
- XML file containing the toolbar and menu layout of the packaged code
- Defines icons resources
- Titles and Translations
- Associates code to be launched to each interface elements
 - Toolbar
 - Main menu and submenus
 - Tools > Addons submenu
 - Help menu
- **Addon tool** <http://www.oocomacros.org/dev.php#101618>

Addon tool

B. Marcellly

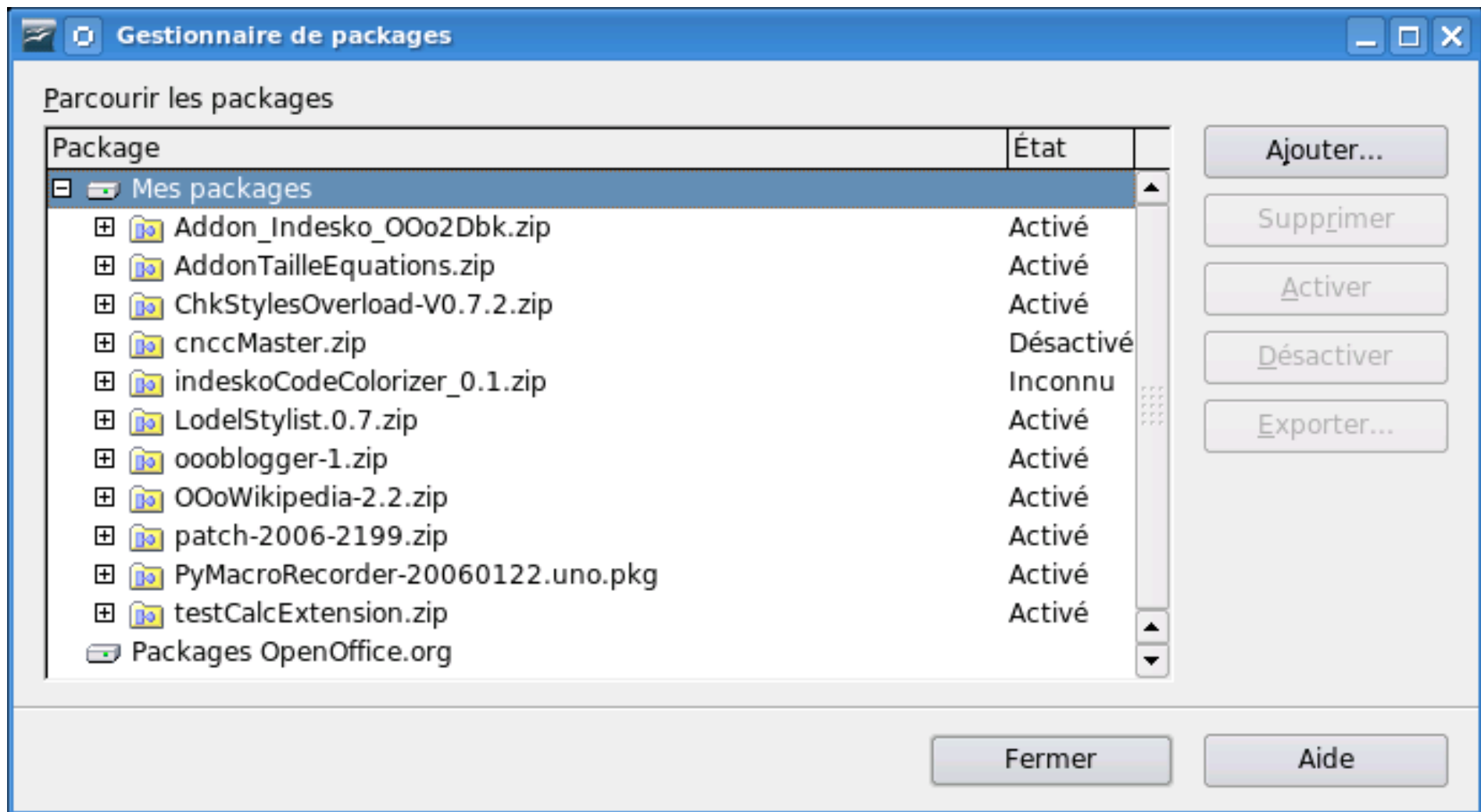


```
</node>  
<node oor:name="bt  
<prop oor:name="<br/><value>com.sun.star.text.TextDocument,com.sun<br/></prop>  
<prop oor:name="Title" oor:type="xs:string"><br/><value>Incrémenter la taille</value><br/></prop>  
<prop oor:name="URL" oor:type="xs:string"><br/><value>macro:///TailleEquations.Module1.Augm<br/></prop>  
<prop oor:name="Target" oor:type="xs:string"><br/><value>_self</value><br/></prop>
```


Deploy to user

- Using command line tool unopkg (OOo closed)
 - `<ooo>/program/unopkg`
- Package manager
 - Tools > Package manager
- To all users in *share* directory or only at *user* level
 - OOo installation directory `share/uno_packages`
 - OpenOffice.org Packages
 - OOo user directory `user/uno_packages`
 - My Packages

Deploy to user



> Access with command lines

- Macros can be accessed by launching a command line
- Example : automatically export a file to PDF

```
soffice 'macro:///myLibrary.module.DocumentToPDF  
("/home/lgodard/source.odt" ,  
"/home/lgodard/result.pdf")'
```

- Python bridge to OOO API
 - Use for macros
 - Use for external/remote scripting
 - Deployable as addons
- Code simplifications similar to OOoBasic ones
- Object oriented so that we can create our own services
 - By overloading existing ones
 - Creating a totally new (defining a new IDL)
- Simple as OOoBasic, powerful as Java ;)
- See Paolo's Mantovani examples
 - <http://www.paolo-mantovani.org/>

> Python example

- HelloWorld python script as a macro

```
def HelloWorldPython( ):
    """Prints the string 'Hello World(in Python)' into the current document"""
    #get the doc from the scripting context which is made available to all scripts
    model = XSCRIPTCONTEXT.getDocument()
    #get the XText interface
    text = model.Text
    #create an XTextRange at the end of the document
    tRange = text.End
    #and set the string
    tRange.String = "Hello World (in Python)"
    return None
```

- All written in pyUNo using OOo graphical toolkit
- Points to the SDK offline and online
- http://www.indesko.com/sites/en/downloads/pyxray___a_tool_for/view
- Under development, need feed back

```
from pyXray import XrayBox
```

```
...  
desktop = smgr.createInstanceWithContext( "com.sun.star.frame.Desktop",ctx)  
XrayBox(ctx,desktop)
```

```
# access the current writer document  
model = desktop.getCurrentComponent()  
XrayBox(ctx,model)
```

The screenshot shows the pyXray - 0.6 beta application window. The title bar includes the application name and standard window controls. The 'Afficher' menu is open, showing options: Propriétés, Méthodes, Services, Interfaces supportées, and Listeners supportés. To the right of the menu are buttons for 'SwXStyleFamilies', 'Documentation du SDK', and 'Xray sur la sélection'. The 'inDesko' logo and contact information for Laurent Godard (lgodard@indesko.com) are also visible. Below the menu is a 'Fermer' button and a note: 'Vous pouvez sélectionner et copier le détail affiché'. The 'Objet affiché' dropdown is set to 'Origine.StyleFamilies'. The main area is split into two panes: 'Count' on the left and 'Détail de la sélection' on the right. The 'Count' pane lists properties: ElementNames (highlighted), ElementType, ImplementationId, ImplementationName, StyleLoaderOptions, SupportedServiceNames, and Types. The 'Détail de la sélection' pane shows the details for 'ElementNames', including its type '[]string', a 'readonly' attribute, and a list of values: 0 (CharacterStyles), 1 (ParagraphStyles), 2 (FrameStyles), 3 (PageStyles), and 4 (NumberingStyles).

Python for remotely driving OOo

- Open OOo in listen mode
 - Command line
 - `<OOo>/program/soffice "-accept=socket,host=localhost,port=2002;urp;"`
 - Every time
 - Configuration file `<Ooo>/share/registry/data/org/openoffice/Setup.xcu`
`<prop oor:name="ooSetupConnectionURL" oor:type="xs:string">`
`<value>socket,host=localhost,port=2002;urp;</value>`
`</prop>`
 - Host & port allow remote scripting
- eg : oooconv
 - A converter farm on an intranet (XML-RPC and asynchronous using twisted framework)
 - <http://svn.nuxeo.org/trac/pub/browser/OOo/oooconv>

Example : doctests

- http://blogs.nuxeo.com/sections/blogs/laurent_godard/2006_04_13_testing-pyuno-programs-with-doctests

```
import doctest  
import sys
```

```
def oooTesting():  
    r""" Let's define the listening host we have to reach and the port ...
```

```
>>> HOST = 'localhost'  
>>> PORT = 11111
```

We now call out helper connecting class:

```
>>> ooo = OOoTools(HOST, PORT)  
>>> ctx = ooo.ctx  
>>> desktop = ooo.desktop
```

So, we are now connected to the listen OpenOffice.org instance

We now start with Calc manipulations by creating a blank spreadsheet file

```
>>> doc = desktop.loadComponentFromURL("private:factory/scalc", '_blank', 0, ())
```

We can verify that this new document is really a spreadsheet by checking the supported OOo service:

```
>>> doc.supportsService("com.sun.star.sheet.SpreadsheetDocument")  
True
```

The new Calc documents opens on a new blank activesheet we retrieve
We also verify that this objetc is really a spreadsheet by checking the relevant supported services.

```
>>> sheet = doc.CurrentController.ActiveSheet  
>>> sheet.supportsService("com.sun.star.sheet.Spreadsheet")  
True
```

Using UNO services

- Overload existing services
- Creating your own IDL
 - Define your own service and callable methods
 - More advanced use but powerful

```
import uno
import unohelper

class EtatSyntheseJob( unohelper.Base, XJobExecutor ):

    def __init__(self, ctx):

    def trigger(self, args):

# pythonloader looks for a static g_ImplementationHelper variable
g_ImplementationHelper = unohelper.ImplementationHelper()
g_ImplementationHelper.addImplementation(EtatSyntheseJob, # UNO object class
    "myownname.EtatSynthese", # implemenation name
    ("org.openoffice.pyuno.myownname.EtatSynthese",),) # list of implemented services
```

PyUNO needs you

- Version 2.3.5
 - Following python versions would be great
- Need an editor or at least a binding
- Enhance addon management allowing several .py files in the extension file
- Use pyUNO to create more and more Extensions
- Extension project
 - http://wiki.services.openoffice.org/wiki/Extensions_development_python
 - More helpers
 - More documentation & feedback

Conclusion

- Create your daily business programs or helpers by implementing scripting Extensions
- A lot of tools and documentation available
- Extensions project and scripting framework
 - Helps you starting
<http://wiki.services.openoffice.org/wiki/Extensions>
 - Distribute your useful tools, feel free to contribute
 - dev@extensions.openoffice.org
 - A download site is being setup (any help ?)
- Tracks to follow at OOoCon 2006
 - Eg: Development track on wednesday afternoon (Juergen Schmidt about extensions infrastructure and Cedric Bosdonnat about URE for going further)

> Thanks



Illustrations from Ben Bois