Introduction to LibreOffice Development

LibreOffice automation via Scripting

Rafael Henrique Palma Lima
Why Scripting?

- Automate common and repetitive tasks
- Extend application functionalities
- Interact directly with the API
- Create custom applications tailored for your needs (integrate databases, create dialogs, forms, etc)

Supported Scripting Languages in LibreOffice

- Basic
- Python
- JavaScript
- BeanShell
Outline

- Creating scripts in Basic
- The ScriptForge library
- Types of scripts in LibreOffice
- Creating Python scripts
GitHub repository

All examples are available at the following repository

Link: https://github.com/rafaelhlima/LibOCon_2023_Scripting
Starting with the Basics

- LibreOffice has a built-in IDE for creating scripts with the Basic language.
- It can also be used to create and edit dialogs.

![Image of LibreOffice IDE and dialog creation]

LibreOffice
Starting with the Basics

- Open any LibreOffice application (f.i. open Calc)
- Go to Tools – Macros – Edit Macros
- A default module named “Module 1” from the “Standard” library will be selected
- Create the sub `SayHello` by writing the next code
- Place the cursor anywhere inside the sub
- Click the Run button (or press F5)
Starting with the Basics

- Open any LibreOffice application
- Add the following subroutine
- This script will create a Writer document with some text in it

```vba
Option Explicit

Sub SayHelloWriter
    Dim sName As String
    sName = InputBox("What is your name?")
    Dim oDoc As Object, oText As Object
    oDoc = StarDesktop.loadComponentFromUrl("private:factory/swriter", "/_blank", 0, Array())
    oText = oDoc.getText()
    oText.insertString(oText.End, "Hello " & sName & CHR$(10), False)
End Sub
```

Hello Rafael
Starting with the Basics

- Open LibreOffice Calc
- Create the next subroutine and run it
- It will write the provided name in cell “A1”
Starting with the Basics

What are *StarDesktop* and *ThisComponent*?

- **StarDesktop**
  Special variable available in Basic to access the desktop component
  

- **ThisComponent**
  Special variable that provides access to the current document
  
Starting with the Basics

- Open LibreOffice Calc
- Create the next subroutine and run it
- It will simply create a table with Z values (standard normal distribution) and cumulative probabilities
Starting with the Basics

This code formats the table with a header and borders.
Starting with the Basics

The code below calls both Subs and makes sure they are only executed on Calc documents

```
Sub BtnCreateDistrTable
    ' Check if we are in a Calc document
    If Not ThisComponent.SupportsService("com.sun.star.sheet.SpreadsheetDocument") Then
        Exit Sub
    End If
    ' Call both sub routines
    CreateStdDistrTable
    FormatRange
End Sub
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Calc</td>
<td>com.sun.star.sheet.SpreadsheetDocument</td>
</tr>
<tr>
<td>Draw</td>
<td>com.sun.star.drawing.DrawDocument</td>
</tr>
<tr>
<td>Impress</td>
<td>com.sun.star.presentation.PresentationDocument</td>
</tr>
<tr>
<td>Writer</td>
<td>com.sun.star.text.TextDocument</td>
</tr>
<tr>
<td>Base</td>
<td>com.sun.star.sdb.OfficeDatabaseDocument</td>
</tr>
</tbody>
</table>
The easiest way to run a script is to go to **Tools – Macros – Run Macro** and the use the Macro Selector to choose the desired macro.
Running Scripts

You can also add controls to the document and associate macros with control events. For instance, create a button and associate it with our macro:
How do I learn all of that?

API references for the previous example

- (service) com.sun.star.table.CellRange
  https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1table_1_1CellRange.html

- (service) com.sun.star.text.CellRange
  https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1text_1_1CellRange.html

- (service) com.sun.star.style.CharacterProperties
  https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1style_1_1CharacterProperties.html

- (struct) com.sun.star.table.BorderLine2
  https://api.libreoffice.org/docs/idl/ref/structcom_1_1sun_1_1star_1_1table_1_1BorderLine2.html

- (constant group) com.sun.star.table.BorderLineStyle
  https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1table_1_1BorderLineStyle.html

- (constant group) com.sun.star.awt.FontWeight
  https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1awt_1_1FontWeight.html
How do I learn all of that?

- The learning curve may seem steep at first, but there are tools and learning resources to help smooth it.
- **XRay tool**: used to inspect objects and helps to discover their properties and methods.

```vbnet
REM ***** BASIC *****

Option Explicit

Sub XrayExample
    ' Inspect the object ThisComponent
    XRay ThisComponent
End Sub
```

**Download link:**
https://berma.pagesperso-orange.fr/Files_en/XrayTool60_en.odt
How do I learn all of that?

LibreOffice Basic official help pages

How do I learn all of that?

How do I learn all of that?

LibreOffice Developer's Guide

How do I learn all of that?

Python LibreOffice Programming

- Preface
- Part 1: Basics
  - Chapter 1. LibreOffice API Concepts
  - Chapter 2. Starting and Stopping
  - Chapter 3. Examining
  - Chapter 4. Listening, and Other Techniques
- Part 2: Writer
  - Chapter 5. Text API Overview
  - Chapter 6. Text Styles
  - Chapter 7. Text Content Other than Strings
  - Chapter 8. Graphic Content
  - Chapter 9. Text Search and Replace
  - Chapter 10. The Linguistics API

UNO stands for Universal Network Objects and is the base component technology for LibreOffice.

It allows to write components using multiple languages (C++, Java, Basic and Python).

The Scripting framework allows to create scripts using Basic, Python, JavaScript and BeanShell.

Terminology in the API: interfaces, services, properties, constants and components.
The UNO API

Service Manager

- Allows to instantiate UNO services and use them in scripts

Service documentation:
https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1linguistic2_1_1SpellChecker.html

REM ***** BASIC *****
Option Explicit
Sub SpellCheckerExample
    Dim oSpellChecker As Object, bReturn As Boolean
    Dim aLocale As New com.sun.star.lang.Locale
    aLocale.Language = "en"
    aLocale.Country = "US"
    ' Create an instance of the SpellChecker service
    oSpellChecker = CreateUnoService("com.sun.star.linguistic2.SpellChecker")
    ' Use the service to test if a word is valid
    Dim sWord As String
    sWord = InputBox("Type a word")
    bReturn = oSpellChecker.isValid(sWord, aLocale, Array())
    If bReturn Then
        MsgBox "The word " & sWord & " is spelled correctly"
    Else
        MsgBox "The word " & sWord & " is not spelled correctly"
    End If
End Sub
UNO Commands

**Command Dispatcher:** you can also dispatch UNO commands in your scripts using the DispatchHelper service

```vba
Sub DispatcherExample
    Dim oFrame As Object, oDispatcher As Object
    oFrame = ThisComponent.CurrentController.Frame
    oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")
    ' Dispatch the .uno:Save command (equivalent to File - Save)
    oDispatcher.executeDispatch(oFrame, ".uno:Save", ":", , Array())
End Sub
```

There are various commands that can be dispatched, for a full list, see
https://wiki.documentfoundation.org/Development/DispatchCommands
UNO Commands

The next example uses UNO commands to copy cell A1 and paste it into cell A2.

```basic
Option Explicit

Sub CopyPasteExample
    ' Instantiate the dispatcher
    Dim oFrame As Object, oDispatcher As Object
    oFrame = ThisComponent.CurrentController.Frame
    oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")
    ' Moves to cell A1
    Dim argS as Object
    argS = New com.sun.star.beans.PropertyValue
    argS.Name = "ToPoint"
    argS.Value = "$A$1"
    oDispatcher.executeDispatch(oFrame, ".uno:GoToCell", ",", Array(argS))
    ' Copy its contents (equivalent to Edit - Copy)
    oDispatcher.executeDispatch(oFrame, ".uno:Copy", ",", Array())
    ' Moves to cell A2
    argS = New com.sun.star.beans.PropertyValue
    argS.Name = "ToPoint"
    argS.Value = "$A$2"
    oDispatcher.executeDispatch(oFrame, ".uno:GoToCell", ",", Array(argS))
    ' Paste content (equivalent to Edit - Paste)
    oDispatcher.executeDispatch(oFrame, ".uno:Paste", ",", Array())
End Sub
```
Organizing Scripts

- Containers, libraries, modules, and dialogs

A library may contain multiple modules and dialogs

Current library

Library container

Library container

A document is also a library container
Organizing Scripts

- **My Macros & Dialogs**: available to all documents for the current user
- **Application Macros & Dialogs**: available to all users and documents (read-only)
- **Document Macros & Dialogs**: used to embed macros in a document
ScriptForge Library

- Offers a variety of services and methods to simplify the creation of Basic and Python scripts by hiding the complexity of the UNO API

<table>
<thead>
<tr>
<th>Category</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibreOffice Basic</td>
<td>Array Dictionary, Exception File System, String TextStream</td>
</tr>
<tr>
<td>Document Content</td>
<td>Base Calc Chart, Database Datasheet, Document Writer</td>
</tr>
<tr>
<td>User Interface</td>
<td>Dialog DialogControl Form, FormControl Menu, PopupMenu UI</td>
</tr>
<tr>
<td>Utilities</td>
<td>Basic L10N Platform, Region Services Session, Timer UnitTest</td>
</tr>
</tbody>
</table>

- Extend the Basic API
- Access dialogs and dialog controls, as well as UI elements
- Access document contents
- Utilities to simplify common tasks performed by macros
ScriptForge Library

- We need to load the ScriptForge library before using it

```vba
Sub ListOpenDocs
    ' We need to make sure the ScriptForge library is loaded
    GlobalScope.BasicLibraries.LoadLibrary("ScriptForge")
    '...
End Sub
```

- We can create a library loader do make it easier

```vba
Sub RequiresSF
    If Not GlobalScope.BasicLibraries.IsLibraryLoaded("ScriptForge") Then
        GlobalScope.BasicLibraries.LoadLibrary("ScriptForge")
    End If
End Sub

Sub ListOpenDocs
    RequiresSF
    Dim svcUI As Object
    svcUI = CreateScriptService("UI")
    '...
End Sub
```

We just need to load SF just once in the same session; use this only if you're unsure SF will have already been loaded.
ScriptForge Library

**Example:** Check all open document windows

```vbnet
Sub ListOpenDocs
    RequiresSF
    Dim svcUI As Object, arrDocs As Object
    svcUI = CreateScriptService("UI")
    ' Array with the list of open documents
    arrDocs = svcUI.Documents
    ' Concatenate the names in the array
    Dim sOpenDocs As String
    sOpenDocs = Join(arrDocs, CHR$(13))
    MsgBox sOpenDocs
End Sub
```

The UI service provides access to all open windows.

The documents here are shown using URL notation.
ScriptForge Library

**Example:** Open an existing Calc document and maximize its window

```vbscript
Sub OpenCalcDoc
    Dim sDocPath As String, svcUI As Object, oDoc As Object
    sDocPath = "file:///home/rafael/Documents/mydoc.ods"
    svcUI = CreateScriptService("UI")
    oDoc = svcUI.OpenDocument(sDocPath)
    svcUI.Maximize(sDocPath)
    MsgBox "This is a " & oDoc.DocumentType & " file"
End Sub
```

In this macro “oDoc” is an instance of the “Calc” service.
ScriptForge Library

Example: Creates an empty Calc document, adds some content and saves it without showing the window to the user

```vba
Sub CreateCalcDoc
    Dim sDocPath As String, svcUI As Object, oDoc As Object
    sDocPath = "file:///home/rafael/Documents/newdoc.ods"
    svcUI = CreateScriptService("UI")
    oDoc = svcUI.CreateDocument(DocumentType := "Calc", Hidden := True)
    oDoc.SetValue("A1", "Hello")
    oDoc.SaveAs(sDocPath, Overwrite := True)
    oDoc.CloseDocument()
End Sub
```
Services are composed of properties and methods (see their documentation in the corresponding help pages); Below is the “Document” service:

<table>
<thead>
<tr>
<th>Properties</th>
<th>Readonly</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties (*)</td>
<td>No</td>
<td>Dictionary service</td>
<td>Returns a ScriptForge.Dictionary object instance. After update, can be passed again to the property for updating the document. Individual items of the dictionary may be either strings, numbers, (Basic) dates or com.sun.star.util.Duration items.</td>
</tr>
<tr>
<td>Description (*)</td>
<td>No</td>
<td>String</td>
<td>Gives access to the Description property of the document (also known as “Comments”)</td>
</tr>
<tr>
<td>DocumentProperties (*)</td>
<td>Yes</td>
<td>Dictionary service</td>
<td>Returns a ScriptForge.Dictionary object containing all the entries. Document statistics are included. Note that they are specific to the type of document. As an example, a Calc document includes a “CellCount” entry. Other documents do not.</td>
</tr>
<tr>
<td>DocumentType</td>
<td>Yes</td>
<td>String</td>
<td>String value with the document type (“Base”, “Calc”, “Writer”, etc)</td>
</tr>
<tr>
<td>ExportFilters</td>
<td>Yes</td>
<td>String array</td>
<td>Returns a list with the export filter names applicable to the current document as a zero-based array of strings. Filters used for both import/export are also returned.</td>
</tr>
</tbody>
</table>

**List of Methods in the Document Service**

- Activate
- CloseDocument
- CreateMenu
- ExportAsPDF
- RunCommand
- Save
- SaveCopyAs
- SetPrinter
- PrintOut
- RemoveMenu
This help page has a list of all properties and methods provided by all ScriptForge services.

List of all **ScriptForge** methods and properties

This help page shows all methods and properties available in the ScriptForge library by service with links to the corresponding documentation.

The **Basic** source code for all ScriptForge services is available via **Application Macros and Dialogs** and is distributed in multiple libraries: ScriptForge, SFDatabase, SFDialogs, SFDocuments, SUUnitTests and SFWidgets. The **Python** portion of the source code is available in the `program\scriptforge.py` file under the LibreOffice installation directory.

**ScriptForge.Array service**

<table>
<thead>
<tr>
<th>List of Methods in the Array Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append</td>
</tr>
<tr>
<td>AppendColumn</td>
</tr>
<tr>
<td>AppendRow</td>
</tr>
<tr>
<td>Contains</td>
</tr>
<tr>
<td>ConvertToDictionary</td>
</tr>
<tr>
<td>Copy</td>
</tr>
<tr>
<td>CountDims</td>
</tr>
<tr>
<td>Difference</td>
</tr>
<tr>
<td>ExportToTextFile</td>
</tr>
<tr>
<td>ExtractColumn</td>
</tr>
<tr>
<td>ExtractRow</td>
</tr>
</tbody>
</table>

Creating Python Scripts

- LibreOffice has supported Python scripts since its first release in 2010
- The default LibreOffice installation comes with a bundled Python interpreter
- However, LibreOffice does not have an integrated Python IDE (yet 😊)
- To make it easier to create and run Python scripts, it is recommended to use the APSO (Alternative Script Organizer for Python) extension

Download link: https://extensions.libreoffice.org/en/extensions/show/apso-alternative-script-organizer-for-python
Creating Python Scripts

You can use APSO’s Python console to execute Python commands

---

XSCRIPTCONTEXT is a global variable that gives access to the Desktop (via `getDesktop`) and the Document (via `getDocument`)

XSCRIPTCONTEXT is documented here:
Creating Python Scripts

In Python scripts we can use a more “Pythonic” syntax:
Creating Python Scripts

Where can we create user scripts that can be used across documents?

- **Application Macros (all users)**
  - **For Windows**: `{Installation}\share\Scripts\python`
  - **For Linux and macOS**: `{Installation}/share/Scripts/python`

- **User Macros (current user only)**
  - **For Windows**: `%APPDATA%\LibreOffice\4\user\Scripts\python`
  - **For Linux and macOS**: `$HOME/.config/libreoffice/4/user/Scripts/python`
Creating Python Scripts

Run the following Basic script to locate where your installation and home folder are (note that the script uses the FileSystem service from the ScriptForge library):

```vbnet
Sub GetFolders
    Dim fs as Object, sMessage As String
    fs = CreateScriptService("FileSystem")
    ' Use native OS file naming notation
    fs.FileNaming = "SYS"
    sMessage = "Installation folder: " & fs.InstallFolder & CHR$(13) & _
              "Home folder: " & fs.HomeFolder
    MsgBox sMessage
End Sub
```
Creating Python Scripts

Let’s convert to Python the script that creates a Writer document with a hello message in it (using ScriptForge)

```python
from scriptforge import CreateScriptService

bas = CreateScriptService("Basic")
ui = CreateScriptService("UI")

def create_writer_doc(args=None):
    name = bas.InputBox("What is your name?")
    doc = ui.createDocument("Writer")
    doc_component = doc.XComponent
    text = doc_component.getText()
    text.insertString(text.End, f"Hello {name}\n", False)

g_exportedScripts = (create_writer_doc, )
```

Tells which scripts will be available via Tools – Macros – Run Macro dialog
Creating Python Scripts

Now let’s convert the “normal distribution” example using Python and ScriptForge

```python
from scriptforge import CreateScriptService

bas = CreateScriptService("Basic")

def create_normdist_table(args=None):
    # Get the Document service instance for the current component
    doc = CreateScriptService("Document", bas.ThisComponent)
    # If it is not a Calc document, do nothing
    if not doc.isCalc:
        return
    # Insert the data in the sheet
    doc.setValue("A1:B1", ["z-Value", "P(Z<z)"])
    z_values = [-3 + z * 0.5 for z in range(13)]
    z_range = doc.Offset("A2", height=len(z_values))
    doc.setValue(z_range, z_values)
    # Insert the formulas
    formula_range = doc.Offset("B2", height=len(z_values))
    base_formula = "=NORM.S.DIST(A2;1)"
    doc.setFormula(formula_range, base_formula)

g_exportedScripts = (create_normdist_table, )
```
Creating Python Scripts

- Now let's add the table formatting code
- Note the new imports of the UNO API

```python
from scriptforge import CreateScriptService
from com.sun.star.awt import FontWeight
from com.sun.star.table import BorderLineStyle
from uno import import createUnoStruct, Enum

# Format the header of the table
range_obj = doc.XCellRange("A1:B1")
range_obj.CellBackColor = bas.RGB(200, 200, 200)
range_obj.CharWeight = FontWeight.BOLD

# Format the remainder of the table
range_obj.CharFontName = "Arial"
justification_center = "com.sun.star.table.CellHoriJustify", "CENTER"
range_obj.HoriJustify = justification_center

# Struct that defines the line format
line_format = createUnoStruct("com.sun.star.table.BorderLineStyle")
line_format.LineStyle = BorderLineStyle.SOLID
line_format.LineWidth = 10
range_obj.TopBorder = line_format
range_obj.BottomBorder = line_format
range_obj.LeftBorder = line_format
range_obj.RightBorder = line_format
```
Creating Python Scripts

This example plots a chart of the X-squared function to illustrate charting capabilities.

```python
def plot_function(args=None):
    # Plot X and Y values for X² function
    data = [(x, math.pow(x, 2)) for x in range(-5, 6)]
    doc = CreateScriptService("Calc", bas.ThisComponent)
    doc.setValue("A1:B1", [["X", "Y"]])
    data_range = doc.Offset("A2", width=2, height=len(data))
    doc.setValue(data_range, data)
    # Select the entire table
    table_range = doc.Region("A1")
    # Insert the chart
    cur_sheet = doc.SheetName(doc.CurrentSelection)
    chart = doc.CreateChart("X-Squared", cur_sheet, table_range)
    chart.ChartType = "XY"
    chart.Legend = False
    chart.Title = "X-Squared Plot"
    chart.XTitle = "X values"
    chart.YTitle = "Y values"
    # Set the line type to "smooth" (using cubic splines)
    diagram = chart.XDiagram
    diagram.SplineType = Enum("com.sun.star.chart2.CurveStyle", "CUBIC_SPLINES")
    # Place the Y-axis at -6 for better visualization
    y_axis = diagram.getYAxis()
    y_axis.CrossoverValue = -6
```
Creating Python Scripts

Now we will create a script that uses a dialog.

It will be a simple random number generator that uses the Gaussian implementation in the “random” Python module.
Creating Python Scripts

- First we need to create the dialog using the Basic IDE
- The dialog name is NumberGenDlg
- Let’s create it in a new library called NumberGenerator
Now let’s create a Python file `numbertest.py` with the macros that open the dialog and close it when “Cancel” is pressed.

```python
from scriptforge import CreateScriptService
import random as rnd

bas = CreateScriptService("Basic")

def open_dialog(args=None):
    dlg = CreateScriptService("Dialog", "GlobalScope", "NumberGenerator", "NumberGenDlg")
    dlg.Execute()

def btn_cancel_click(event=None):
    # Get the control that was clicked
    control = CreateScriptService("DialogEvent", event)
    # Get the parent dialog and terminate it
    dlg = control.Parent
    dlg.EndExecute(bas.IDCANCEL)
```
Creating Python Scripts

This macro needs to be associated with the “Generate” button

```python
def btn_generate_click(event=None):
    # Get the control that was clicked
    control = CreateScriptService("DialogEvent", event)
    dlg = control.Parent
    # Get the parameters from the dialog
    start_cell = dlg.Controls("editStartCell").Value
    num_cols = int(dlg.Controls("editNumColumns").Value)
    num_rows = int(dlg.Controls("editNumRows").Value)
    mean = float(dlg.Controls("editMean").Value)
    std_dev = float(dlg.Controls("editStdDev").Value)
    # Generate the numbers
    numbers = [[rnd.gauss(mean, std_dev) for _ in range(num_cols)] for _ in range(num_rows)]
    # Get the document and add the data
    doc = CreateScriptService("Document", bas.ThisComponent)
    num_range = doc.Offset(start_cell, width=num_cols, height=num_rows)
    doc.setValue(num_range, numbers)
```
Thank you ...

- The ScriptForge team
  Jean-Pierre Ledure, Rafael Lima, Alain Romedenne
- Documentation
- ScriptForge Sources
  https://gitlab.com/LibreOfficiant/scriptforge
- Type Support (typings) for ScriptForge, by PauL Moss.
  https://github.com/Amourspirit/python-types-scriptforge
- Telegram groups
  ScriptForge
  LibreOffice Macros & Scripting

All text and image content in this document is licensed under the Creative Commons Attribution-Share Alike 4.0 License (unless otherwise specified). “LibreOffice” and “The Document Foundation” are registered trademarks. Their respective logos and icons are subject to international copyright laws. The use of these thereof is subject to trademark policy.