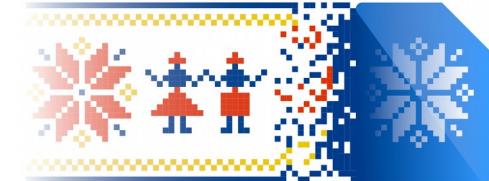




September 20 - 23, 2023



LibreOffice
The Document Foundation



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



BASIC

GitHub repository

- ▼ All examples are available at the following repository

The screenshot shows a GitHub repository page for 'LibOCon_2023_Scripting'. The repository is public and has one branch ('main') and zero tags. The last commit was made two minutes ago by 'rafaelhlima' with four commits. The commit history includes:

File / Commit Type	Description	Time Ago
rafaelhlima Basic scripts	Initial commit	348d923 2 minutes ago
Libocon2023	Basic scripts	2 minutes ago
NumberGenerator	Number generator dialog	5 minutes ago
README.md	Initial commit	9 minutes ago
conference.py	Initial Upload	6 minutes ago
numbergen.py	Initial Upload	6 minutes ago

Link: [https://github.com\(rafaelhlima/LibOCon_2023_Scripting](https://github.com(rafaelhlima/LibOCon_2023_Scripting)

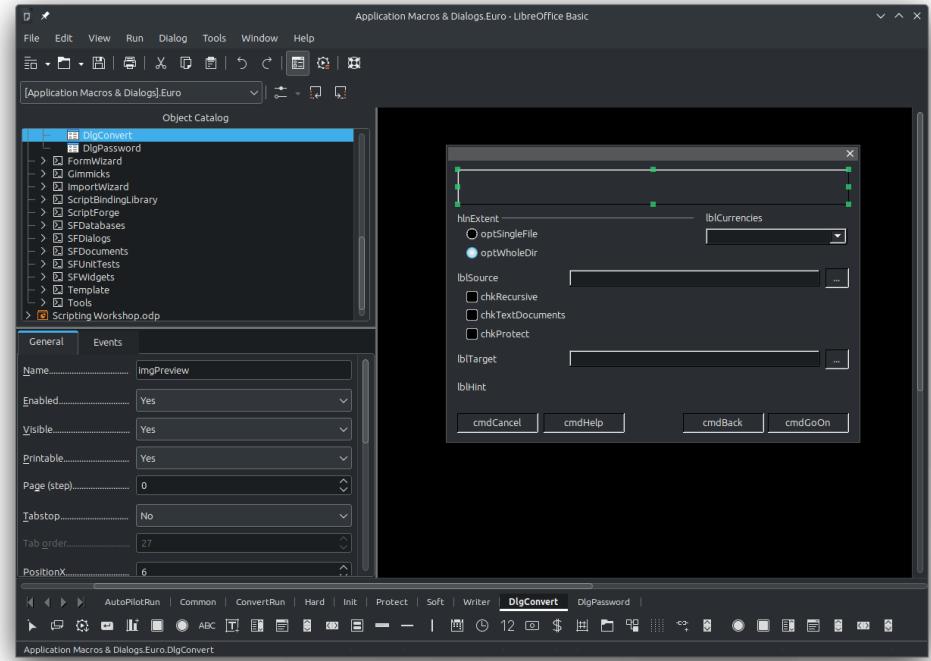
Starting with the Basics

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

The screenshot shows the LibreOffice Basic IDE interface. On the left is the Object Catalog pane, which lists various components and libraries. In the center is the code editor window displaying a script:

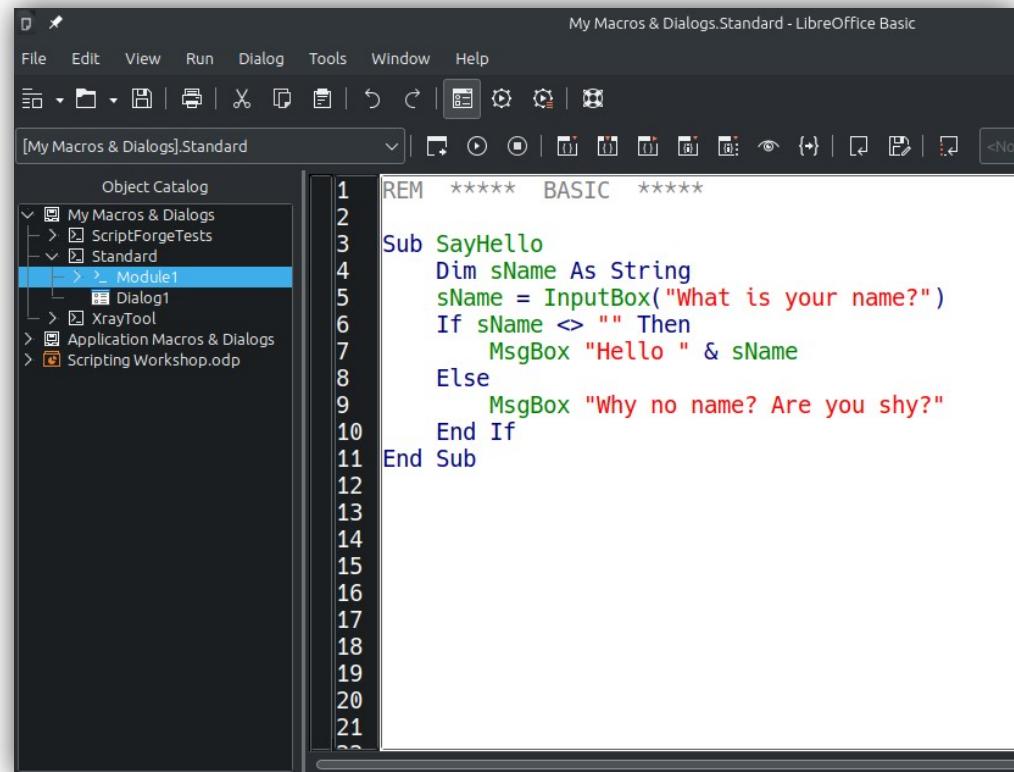
```
Option Explicit
Public oPreSelRange as Object
Sub Main()
    BasicLibraries.LoadLibrary("Tools")
    If InitResources("Euro Converter") Then
        bDoUnProtect = False
        bPreSelected = True
        oDocument = ThisComponent
        RetrieveDocumentObjects()
        InitializeConverter(oDocument.CharLocale, 1)
        GetPreSelectedRange()
        If GoOn Then
            DialogModel.lstCurrencies.TabIndex = 2
            DialogConvert.GetControl("chkComplete").SetFocus()
            DialogConvert.Execute
        End If
        DialogConvert.Dispose
    End If
End Sub
```

At the bottom, there are tabs for AutoPilotRun, Common, ConvertRun, Hard, Init, Protect, Soft, Writer, DlgConvert, and DlgPassword. The ConvertRun tab is selected.



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)

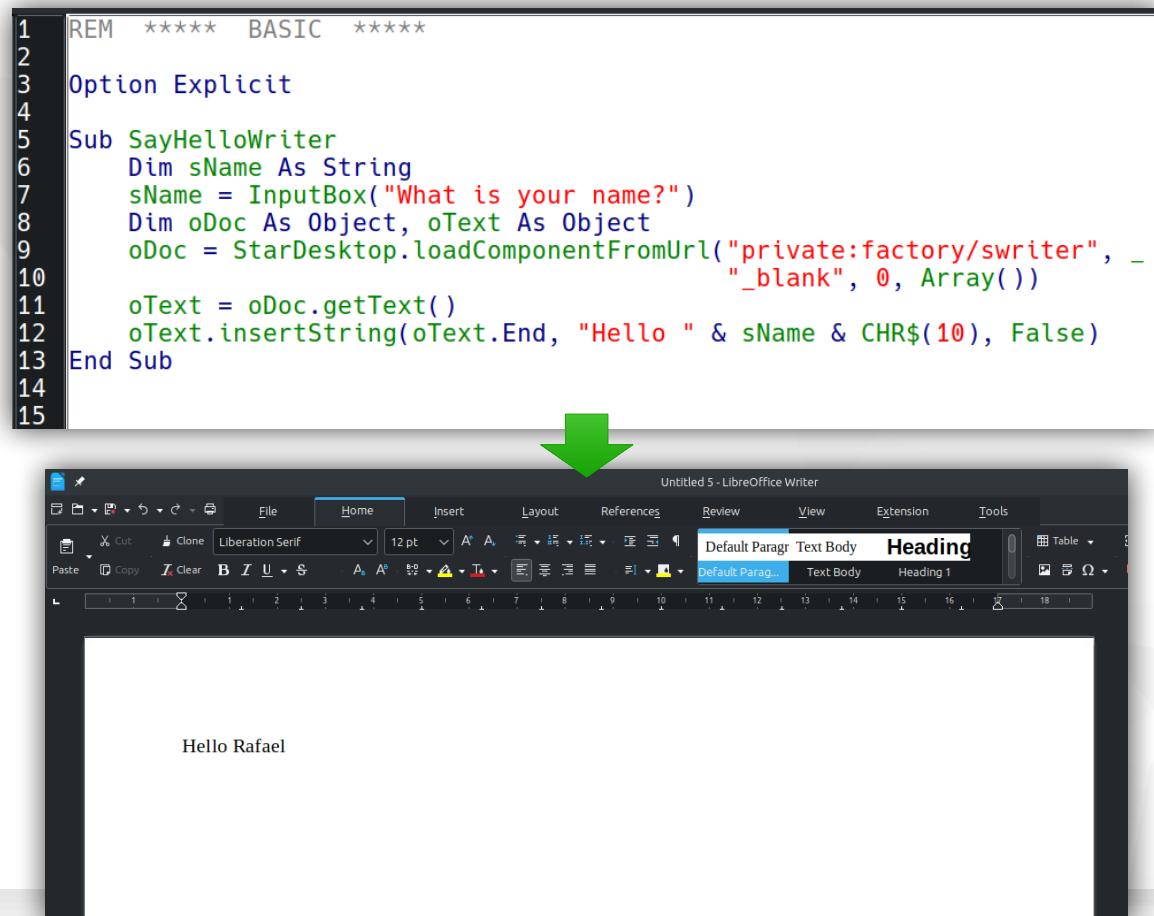


The screenshot shows the LibreOffice Basic Editor window titled "My Macros & Dialogs.Standard - LibreOffice Basic". The menu bar includes File, Edit, View, Run, Dialog, Tools, Window, and Help. The toolbar has various icons for file operations. The left pane displays the "Object Catalog" with a tree structure: My Macros & Dialogs, ScriptForgeTests, Standard, Module1 (selected), Dialog1, XrayTool, Application Macros & Dialogs, and Scripting Workshop.odp. The right pane contains the macro code:

```
REM ***** BASIC *****
Sub SayHello
    Dim sName As String
    sName = InputBox("what is your name?")
    If sName <> "" Then
        MsgBox "Hello " & sName
    Else
        MsgBox "Why no name? Are you shy?"
    End If
End Sub
```

Starting with the Basics

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



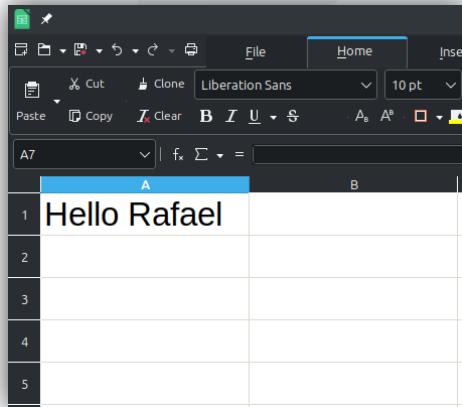
```
REM ***** BASIC *****
Option Explicit

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
```

The screenshot shows the LibreOffice Writer application interface. At the top, there's a menu bar with File, Home, Insert, Layout, References, Review, View, Extension, and Tools. Below the menu is a toolbar with various icons for text operations like Cut, Copy, Paste, and Bold. The main workspace is titled "Untitled 5 - LibreOffice Writer". In the bottom right corner of the workspace, the text "Hello Rafael" is displayed. A large green arrow points from the code editor window above to the document window below, indicating the flow from script to output.

Starting with the Basics

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



```
REM ***** BASIC *****
Option Explicit
Sub SayHelloCalc
    Dim sName As String
    sName = InputBox("What is your name?")
    Dim oCell As Object
    oCell = ThisComponent.Sheets(0).getCellRangeByName("A1")
    oCell.SetString("Hello " & sName)
End Sub
```

Starting with the Basics

▼ What are *StarDesktop* and *ThisComponent*?

StarDesktop

Special variable available in Basic to access the desktop component

More info:

<https://help.libreoffice.org/latest/en-US/text/sbasic/shared/stardesktop.html>

ThisComponent

Special variable that provides access to the current document

More info:

<https://help.libreoffice.org/latest/en-US/text/sbasic/shared/03132200.html>

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

A	B
z-Value	P(Z<z)
-3	0,001349898
-2,5	0,006209665
-2	0,022750132
-1,5	0,066807201
-1	0,158655254
-0,5	0,308537539
0	0,5
0,5	0,691462461
1	0,841344746
1,5	0,933192799
2	0,977249868
2,5	0,993790335
3	0,998650102

```
REM ***** BASIC *****
Option Explicit

Sub CreateStdDistrTable
    Dim oSheet As Object, oRange As Object, oCell As Object
    oSheet = ThisComponent.Sheets(0)
    oRange = oSheet.getCellRangeByName("A1:B1")
    oRange.setDataArray(Array(Array("z-Value", "P(Z<z)")))
    Dim zValue As Double, nRow As Integer, sAddress As String
    nRow = 1
    For zValue = -3 To 3 Step 0.5
        oCell = oSheet.getCellByPosition(0, nRow)
        sAddress = oCell.AbsoluteName
        oCell.setValue(zValue)
        oCell = oSheet.getCellByPosition(1, nRow)
        oCell.setFormula("=NORM.S.DIST(" & sAddress & ";1)")
        nRow = nRow + 1
    Next zValue
End Sub
```



Starting with the Basics

- ▼ This code formats the table with a header and borders

	A	B
1	z-Value	P(Z<z)
2	-3	0,001349898
3	-2,5	0,006209665
4	-2	0,022750132
5	-1,5	0,066807201
6	-1	0,158655254
7	-0,5	0,308537539
8	0	0,5
9	0,5	0,691462461
10	1	0,841344746
11	1,5	0,933192799
12	2	0,977249868
13	2,5	0,993790335
14	3	0,998650102

```
REM ***** BASIC *****
Option Explicit

Sub FormatRange
    Dim oRange As Object
    ' Format header
    oRange = ThisComponent.Sheets(0).getCellRangeByName( "A1:B1" )
    oRange.CellBackColor = RGB(200, 200, 200)
    oRange.CharWeight = com.sun.star.awt.FontWeight.BOLD
    'Format the entire table
    oRange = ThisComponent.Sheets(0).getCellRangeByName( "A1:B14" )
    oRange.CharFontName = "Arial"
    ' Align contents at the center
    Dim eAlign : eAlign = com.sun.star.table.CellHoriJustify
    oRange.HoriJustify = eAlign.CENTER
    ' Create a border format object
    Dim lineFormat As New com.sun.star.table.BorderLine2
    lineFormat.LineStyle = com.sun.star.table.BorderLineStyle.SOLID
    lineFormat.LineWidth = 10
    ' Set borders around all cells
    oRange.TopBorder = lineFormat
    oRange.BottomBorder = lineFormat
    oRange.LeftBorder = lineFormat
    oRange.RightBorder = lineFormat
End Sub
```



Starting with the Basics

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

```
98 Sub BtnCreateDistrTable
99     ' Check if we are in a Calc document
100    If Not ThisComponent.SupportsService( "com.sun.star.sheet.SpreadsheetDocument" ) Then
101        Exit Sub
102    End If
103    ' Call both sub routines
104    CreateStdDistrTable
105    FormatRange
106 End Sub
```



Calc	com.sun.star.sheet.SpreadsheetDocument
Draw	com.sun.star.drawing.DrawDocument
Impress	com.sun.star.presentation.PresentationDocument
Writer	com.sun.star.text.TextDocument
Base	com.sun.star.sdb.OfficeDatabaseDocument

Running Scripts

- The easiest way to run a script is to go to **Tools – Macros – Run Macro** and use the Macro Selector to choose the desired macro

A screenshot of LibreOffice showing a table of z-values and a Macro Selector dialog.

The table on the left shows the cumulative probability $P(Z < z)$ for various z-values:

	A	B
1	z-Value	$P(Z < z)$
2	-3	0,0013499
3	-2,5	0,00620967
4	-2	0,02275013
5	-1,5	0,0668072
6	-1	0,15865525
7	-0,5	0,30853754
8	0	0,5
9	0,5	0,69146246
10	1	0,84134475
11	1,5	0,9331928
12	2	0,97724987
13	2,5	0,99379033
14	3	0,9986501
15		
16		
17		
18		
19		
20		
21		
22		

The Macro Selector dialog is open on the right, showing the library structure and available macros:

- Library:** My Macros, Prova_P02A_Sub, ScriptForgeTests, Standard, Module1 (selected).
- Macro Name:** BtnCreateDistrTable, CopyPasteExample, CreateCalcDoc, CreateStdDistrTable, DispatcherExample, FormatRange, FormDocTest, ListOpenDocs, MainTest2.
- Description:** (empty)
- Buttons:** Help, Run, Cancel.

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:

The screenshot shows a LibreOffice Calc spreadsheet with a table of z-values and their corresponding P(Z<z) values. A button labeled "Generate Table" is placed on the sheet. A context menu is open on the button, displaying the "Properties" dialog (Events tab) and the "Assign Action" dialog.

Properties: Push Button (Events Tab)

- Key released.....
- Mouse inside.....
- Mouse moved while key pressed...
- Mouse moved.....
- Mouse button pressed.....

Assign Action

Event	Assigned Action
Key released	
Mouse Inside	
Mouse moved while key pressed	
Mouse moved	
Mouse button pressed	Standard.Module1.BtnCreateDistrTab
Mouse button released	
Mouse outside	
Prior to reset	

Buttons in the Assign Action dialog:
Assign (Macro...), Remove, Remove All, OK, Cancel

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/namespacem_1_1sun_1_1star_1_1table_1_1BorderLineStyle.html

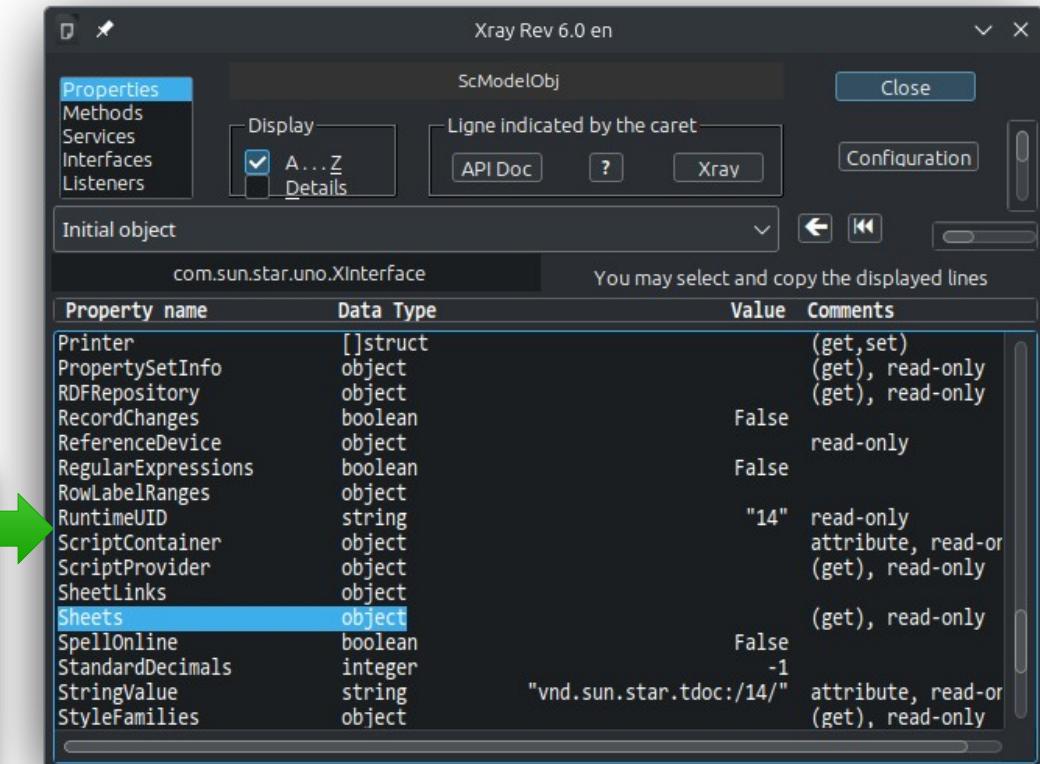
▼ (constant group) **com.sun.star.awt.FontWeight**

https://api.libreoffice.org/docs/idl/ref/namespacem_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

```
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

The screenshot shows the LibreOffice 7.6 Help interface for LibreOffice Basic. The main content area is titled "Programming with LibreOffice Basic". It contains several sections: "Basics", "Syntax", "Integrated Development Environment (IDE)", and "Document Event-Driven Macros". Below these sections, there is descriptive text and links to further information. The left sidebar is titled "Index" and lists various operators and functions such as "+", "*", "/", "\", "&", "<-", "Abs", "Access2Base", "Access2Base -- Console", etc. The right sidebar is titled "Contents" and provides a detailed table of contents for LibreOffice Basic, including sections like "Text Documents (Writer)", "HTML Documents (Writer Web)", "Spreadsheets (Calc)", "Presentations (Impress)", "Drawings (Draw)", "Database Functionality (Base)", "Formulas (Math)", "Charts and Diagrams", "Macros and Scripting", and "LibreOffice BASIC".

LibreOffice 7.6 Help

Module Language

Index

Search in chosen module

← → 1 2 3 4 5 6

Basic

- "*" operator (mathematical)
- "+" operator (mathematical)
- "-" operator (mathematical)
- "/" operator (mathematical)
- "\" operator (mathematical)
- "^" operator (mathematical)
- "& or +" concatenation (strings)
- Abs function
- Access2Base
- Access2Base -- Console
- Access2Base -- dlgTrace
- Access2Base -- _DumpToFile
- Access2Base -- Trace
- Access databases -- run In Base
- adding libraries
- Alternative Python Scripts Organizer
- ampersand symbol -- in literal notation
- ampersand symbol -- in string handling
- AND operator (logical)
- API -- ActionEvent
- API -- awt.XControl

Programming with LibreOffice Basic

This is where you find general information about working with macros and LibreOffice Basic.

Basics

This section provides the fundamentals for working with LibreOffice Basic.

Syntax

This section describes the basic syntax elements of LibreOffice Basic. For a detailed description please refer to the LibreOffice Basic Guide which is available separately.

Integrated Development Environment (IDE)

This section describes the Integrated Development Environment for LibreOffice Basic.

Document Event-Driven Macros

This section describes how to assign scripts to application, document or form events.

Please support us!

Contents

- Text Documents (Writer)
- HTML Documents (Writer Web)
- Spreadsheets (Calc)
- Presentations (Impress)
- Drawings (Draw)
- Database Functionality (Base)
- Formulas (Math)
- Charts and Diagrams
- Macros and Scripting
 - LibreOffice BASIC
 - General Information and User Interface Usage
 - Command Reference
 - Compiler options
 - Using Procedures and Functions
 - Libraries, Modules and Dialogs
 - How to Read Syntax Diagrams
 - Functions, Statements and Operators
 - Alphabetic List of Functions, Statements and Operators
 - Advanced Basic Libraries
 - ScriptForge Library
 - Overview of the ScriptForge Library
 - List of all ScriptForge methods and properties
 - Creating Python Scripts with ScriptForge
 - ScriptForge method signatures
 - Array service
 - Base service
 - Basic service
 - Calc service

Link: <https://help.libreoffice.org/latest/en-US/text/sbasic/shared/01000000.html>

How do I learn all of that?

ScriptForge Library help pages

The screenshot shows the LibreOffice 7.6 Help interface. The title bar says "LibreOffice 7.6 Help". The left sidebar has a search bar and a navigation menu with tabs for "Module" and "Language". The main content area is titled "The ScriptForge Library". It contains sections on how to access the command via Tools - Macros - LibreOffice Basic - Edit, and details about ScriptForge libraries being extensible resources. It also provides examples for loading the library in Basic and Python scripts. Below this, there's a section on invoking services and a code snippet for Basic. The right sidebar lists "Contents" including Text Documents, HTML Documents, Spreadsheets, Presentations, Drawings, Database Functionality, Formulas, Charts, Macros, and ScriptForge Library. A green arrow points from the "ScriptForge Library" link in the Contents list towards the "List of all ScriptForge methods and properties" link in the main content area.

Index

Search in chosen module

Module Language

The ScriptForge Library

To access this command... Open Tools - Macros - LibreOffice Basic - Edit and select Application Macros container.

ScriptForge libraries build up an extensible collection of macro scripting resources for LibreOffice to be invoked from Basic macros or Python scripts.

- Basic macros require to load ScriptForge library using the following statement:
`GlobalScope.BasicLibraries.loadLibrary("ScriptForge")`
- Python scripts require an import from `scriptforge` module:
`from scriptforge import CreateScriptService`

To learn more about how to create and execute Python scripts using the `ScriptForge` library, read the help page [Creating Python Scripts with ScriptForge](#).

Invoking ScriptForge services

The described modules and classes are invoked from user scripts as "Services". A generic constructor of those services has been designed for that purpose for each language.

The `Dispose` method is available in all services and should be called to free up resources after usage:

In Basic

```
1 GlobalScope.BasicLibraries.LoadLibrary("ScriptForge")
2 Set oSvc = CreateScriptService("servicename[, arg0, arg1, ...]")
3 ...
4 oSvc.Dispose()
```

Contents

- Text Documents (Writer)
- HTML Documents (Writer Web)
- Spreadsheets (Calc)
- Presentations (Impress)
- Drawings (Draw)
- Database Functionality (Base)
- Formulas (Math)
- Charts and Diagrams
- Macros and Scripting
 - LibreOffice BASIC
 - General Information and User Interface Usage
 - Command Reference
 - Compiler options
 - Using Procedures and Functions
 - Libraries, Modules and Dialogs
 - How to Read Syntax Diagrams
 - Functions, Statements and Operators
 - Alphabetic List of Functions, Statements and Operators
 - Advanced Basic Libraries
 - ScriptForge Library
 - Overview of the ScriptForge Library
 - List of all ScriptForge methods and properties
 - Creating Python Scripts with ScriptForge
 - ScriptForge method signatures
 - Array service
 - Base service
 - Basic service
 - Calc service
 - Chart service
 - Database service
 - Datasheet service
 - Dialog service
 - DialogControl service
 - Dictionary service
 - Document service
 - Exception service
 - FileSystem service
 - Form service
 - FormControl service

Link: https://help.libreoffice.org/latest/en-US/text/sbasic/shared/03/lib_ScriptForge.html

How do I learn all of that?

LibreOffice Developer's Guide

LibreOffice Developer's Guide

Page Discussion < Documentation | DevGuide

Contributing to the Developer's Guide

What This Manual Covers

This manual describes how to write programs using the component technology UNO (Universal Network Objects) with LibreOffice.

Most examples provided are written in Java. As well as Java, the language binding for C++, the UNO access for LibreOffice Basic and the OLE Automation bridge that uses LibreOffice through Microsoft's component technology COM/DCOM is described.

How This Book is Organized

Every page of this book has a Table of Contents at the right side of the page. This TOC shows the content of the current part and navigation links to browse parts and pages of this book.

First Steps

The First Steps chapter describes the setting up of a Java UNO development environment to achieve the solutions you need. At the end of this chapter, you will be equipped with the essentials required for the following chapters about the LibreOffice applications.

Professional UNO

This chapter introduces API and UNO concepts and explains the specifics of the programming languages and technologies that can be used with UNO. It will help you to write industrial strength UNO programs, use one of the languages besides Java or improve your understanding of the API reference.

Read View source View history

LibreOffice Developer's Guide

- First Steps
- Professional UNO
- Writing UNO Components
- Extensions
- Advanced UNO
- Office Development
- Text Documents
- [Spreadsheet Documents](#)
- Drawing Documents and Presentation Documents
- Charts
- LibreOffice Basic
- Database Access
- Forms
- Universal Content Broker
- Configuration Management
- JavaBean for Office Components
- Accessibility
- Scripting Framework
- Graphical User Interfaces
- Guidelines and



Link: https://wiki.documentfoundation.org/Documentation/DevGuide/LibreOffice_Developers_Guide

How do I learn all of that?

Python LibreOffice Programming

The screenshot shows a dark-themed documentation page for "Python LibreOffice Programming". At the top left is a sidebar with a "BOOK:" section containing a tree view of chapters: Preface, Part 1: Basics, Part 2: Writer, Part 3: Draw & Impress, Part 4: Calc, and Part 5: Charts. Below this are sections for "HELP:" (Help Documentation) and "GUIDES:" (Guides). The main content area has a header "Python LibreOffice Programming" and a navigation bar with a home icon, a "/ Python LibreOffice Programming" path, and a "EDIT ON GITHUB" button. The main content lists chapters under "Part 1: Basics" and "Part 2: Writer".

BOOK:

- Python LibreOffice Programming
 - Preface
 - Part 1: Basics
 - Part 2: Writer
 - Part 3: Draw & Impress
 - Part 4: Calc
 - Part 5: Charts

HELP:

- Help Documentation

GUIDES:

- Guides

/ Python LibreOffice Programming

EDIT ON GITHUB

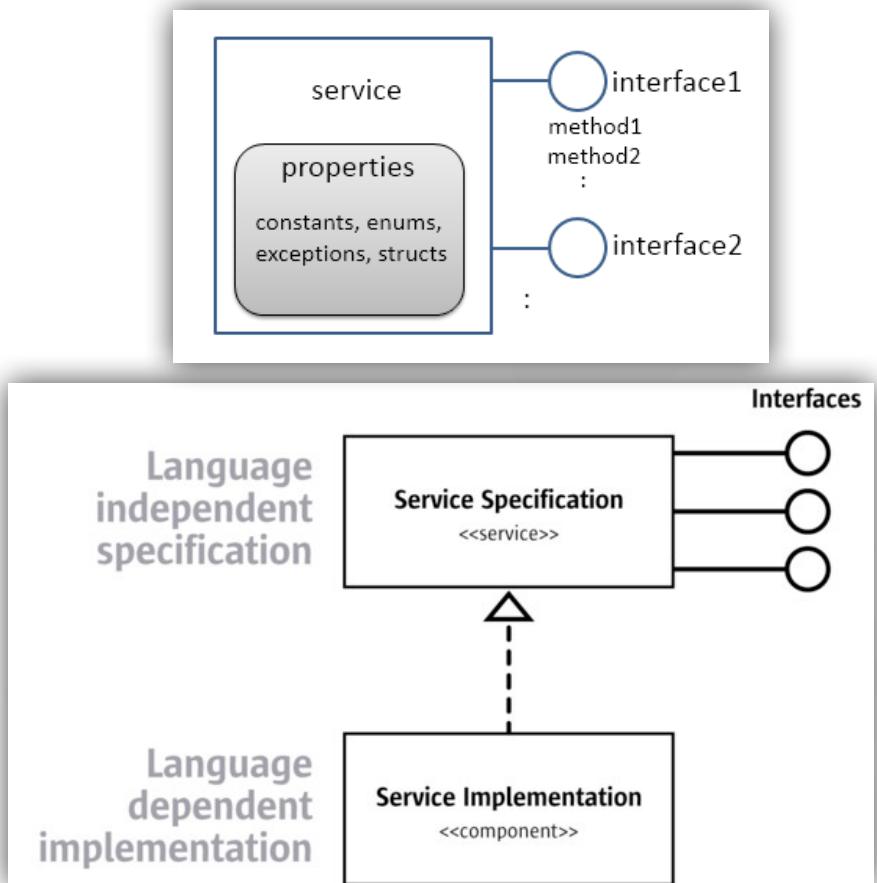
Python LibreOffice Programming

- 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

Link: <https://python-ooo-dev-tools.readthedocs.io/en/latest/odev/index.html>

The UNO 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

```
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
```



Service documentation:

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

UNO Commands

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

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

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

UNO Commands

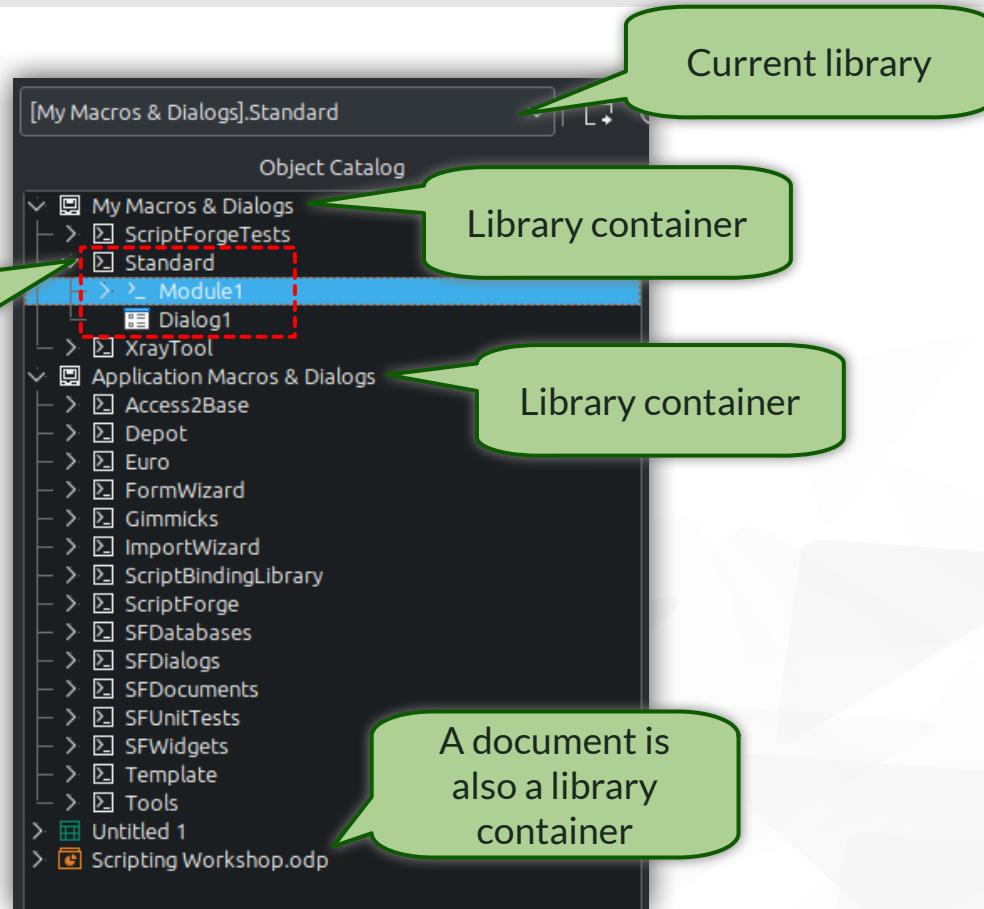
- The next example uses UNO commands to copy cell A1 and paste it into cell A2

```
REM ***** BASIC *****
1 Option Explicit
2
3 Sub CopyPasteExample
4     ' Instantiate the dispatcher
5     Dim oFrame As Object, oDispatcher As Object
6     oFrame = ThisComponent.CurrentController.Frame
7     oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")
8     ' Moves to cell A1
9     Dim args1(0) As New com.sun.star.beans.PropertyValue
10    args1(0).Name = "ToPoint"
11    args1(0).Value = "$A$1"
12    oDispatcher.executeDispatch(oFrame, ".uno:GoToCell", "", , args1())
13    ' Copy its contents (equivalent to Edit - Copy)
14    oDispatcher.executeDispatch(oFrame, ".uno:Copy", "", , Array())
15    ' Moves to cell A2
16    args1(0).Value = "$A$2"
17    oDispatcher.executeDispatch(oFrame, ".uno:GoToCell", "", , args1())
18    ' Paste contents (equivalent to Edit - Paste)
19    oDispatcher.executeDispatch(oFrame, ".uno:Paste", "", , Array())
20
21 End Sub
22
23
```

Organizing Scripts

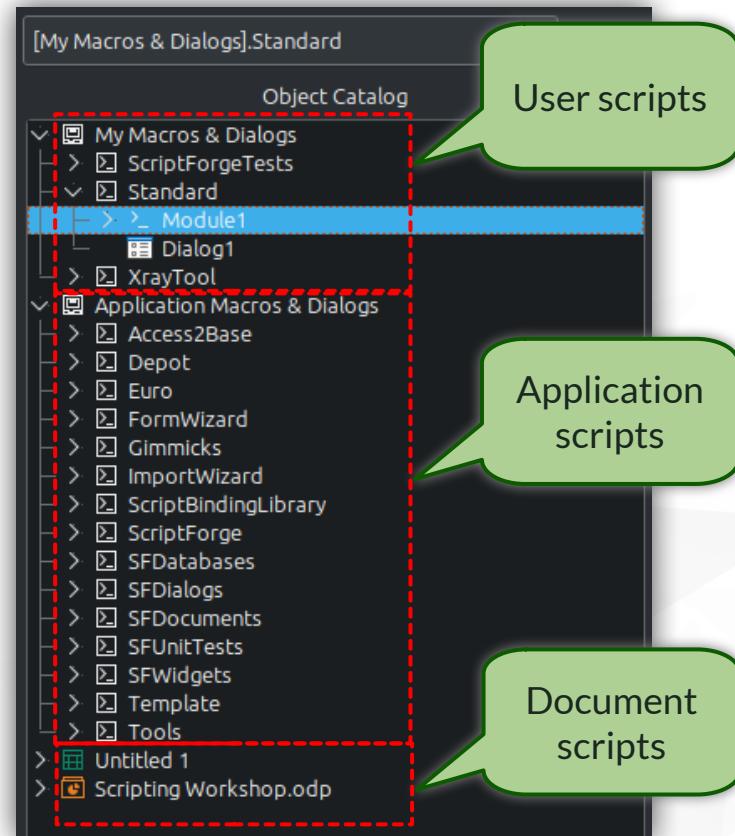
Containers, libraries, modules and dialogs

A library may
contain multiple
modules and
dialogs



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

Category	Services		
LibreOffice Basic	Array Dictionary	Exception FileSystem	String TextStream
Document Content	Base Calc Chart	Database Datasheet	Document Writer
User Interface	Dialog DialogControl Form	FormControl Menu	PopupMenu UI
Utilities	Basic L10N Platform	Region Services Session	Timer UnitTest

Access document contents

Utilities to simplify common tasks performed by macros

Extend the Basic API

Access dialogs and dialog controls, as well as UI elements

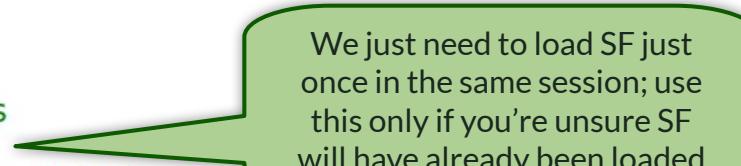
ScriptForge Library

- ▼ We need to load the ScriptForge library before using it

```
5 Sub ListOpenDocs
6     ' We need to make sure the ScriptForge library is loaded
7     GlobalScope.BasicLibraries.LoadLibrary("ScriptForge")
8     ' ...
9 End Sub
```

- ▼ We can create a library loader do make it easier

```
5 Sub RequiresSF
6     If Not GlobalScope.BasicLibraries.IsLibraryLoaded("ScriptForge") Then
7         GlobalScope.BasicLibraries.LoadLibrary("ScriptForge")
8     End If
9 End Sub
10
11 Sub ListOpenDocs
12     RequiresSF
13     Dim svcUI As Object
14     svcUI = CreateScriptService("UI")
15     ' ...
16 End Sub
```



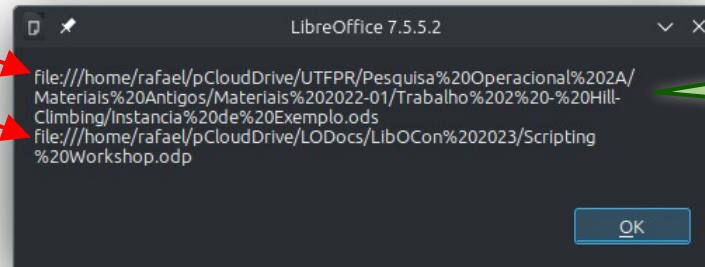
We just need to load SF just once in the same session; use this only if you're unsure SF will have already been loaded

ScriptForge Library

Example: Check all open document windows

```
11 Sub ListOpenDocs
12     RequiresSF
13     Dim svcUI As Object, arrDocs As Object
14     svcUI = CreateScriptService("UI")
15     ' Array with the list of open documents
16     arrDocs = svcUI.Documents
17     ' Concatenate the names in the array
18     Dim sOpenDocs As String
19     sOpenDocs = Join(arrDocs, CHR$(13))
20     MsgBox sOpenDocs
21 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

```
5 Sub OpenCalcDoc
6     Dim sDocPath As String, svcUI As Object, oDoc As Object
7     sDocPath = "file:///home/rafael/Documents/mydoc.ods"
8     svcUI = CreateScriptService("UI")
9     oDoc = svcUI.OpenDocument(sDocPath)
10    svcUI.Maximize(sDocPath)
11    MsgBox "This is a " & oDoc.DocumentType & " file"
12 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

```
5 Sub CreateCalcDoc
6     Dim sDocPath As String, svcUI As Object, oDoc As Object
7     sDocPath = "file:///home/rafael/Documents/newdoc.ods"
8     svcUI = CreateScriptService("UI")
9     oDoc = svcUI.CreateDocument(DocumentType := "Calc", Hidden := True)
10    oDoc.SetValue("A1", "Hello")
11    oDoc.SaveAs(sDocPath, Overwrite := True)
12    oDoc.CloseDocument()
13 End Sub
```

ScriptForge Library

- Services are composed of properties and methods (see their documentation in the corresponding help pages); Below is the “Document” service:

Properties			
Name	Readonly	Type	Description
CustomProperties (*)	No	Dictionary service	Returns a <code>ScriptForge.Dictionary</code> 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 <code>com.sun.star.util.Duration</code> items.
Description (*)	No	String	Gives access to the Description property of the document (also known as "Comments")
DocumentProperties (*)	Yes	Dictionary service	Returns a <code>ScriptForge.Dictionary</code> 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.
DocumentType	Yes	String	String value with the document type ("Base", "Calc", "Writer", etc)
ExportFilters	Yes	String array	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.

List of Methods in the Document Service		
Activate	PrintOut	SaveAs
CloseDocument	RemoveMenu	SaveCopyAs
CreateMenu	RunCommand	SetPrinter
ExportAsPDF	Save	

ScriptForge Library

- ▼ 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**, **SFDatabases**, **SFDialogs**, **SFDocuments**, **SFUnitTests** 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

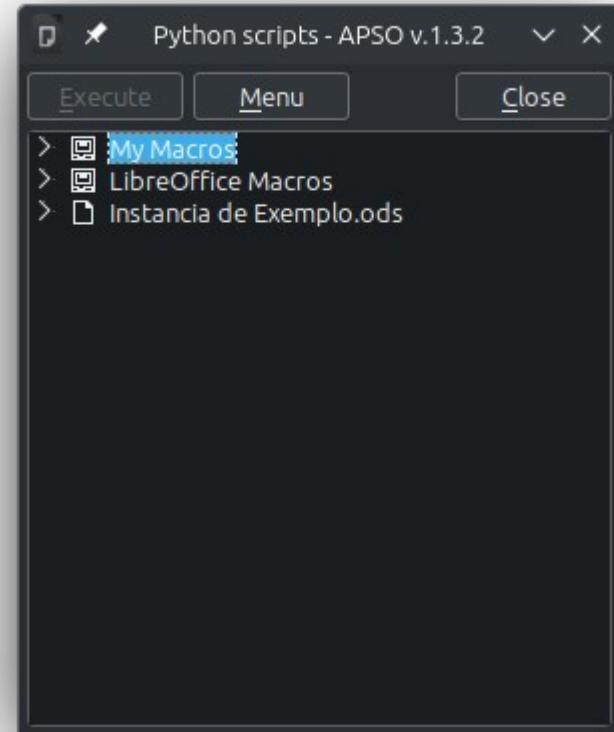
List of Methods in the Array Service

Append	Flatten	Reverse
AppendColumn	ImportFromCSVFile	Shuffle
AppendRow	IndexOf	Slice
Contains	Insert	Sort
ConvertToDictionary	InsertSorted	SortColumns
Copy	Intersection	SortRows
CountDims	Join2D	Transpose
Difference	Prepend	TrimArray
ExportToTextFile	PrependColumn	Union
ExtractColumn	PrependRow	Unique
ExtractRow	RangeInit	

Link: https://help.libreoffice.org/latest/en-US/text/sbasic/shared/03/sf_toc.html

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

The screenshot shows a LibreOffice spreadsheet application. On the left, there is a table with rows from 1 to 16. Row 1 contains "Hello" in cell A1. Rows 2 through 11 contain the numbers 0 through 9 respectively in cell A2-A11. The Python console window is open in the center, titled "APSO console". It displays the following Python code:

```
APSO python console [LibreOffice]
3.11.4 (main, Jun  9 2023, 07:59:55) [GCC 12.3.0]
Type "help", "copyright", "credits" or "license" for more information
>>> document = XSCRIPTCONTEXT.getDocument()
>>> sheet = document.Sheets[0]
>>> cell = sheet.getCellRangeByName("A1")
>>> cell.setString("Hello")
>>> data = [[i] for i in range(10)]
>>> range = sheet.getCellRangeByName("A2:A11")
>>> range.setDataArray(data)
>>>
```

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

XSCRIPTCONTEXT is documented here:

https://help.libreoffice.org/latest/en-US/text/sbasic/python/python_programming.htm

Creating Python Scripts

- ▼ In Python scripts we can use a more “Pythonic” syntax:

The image shows a screenshot of LibreOffice. On the left, there is a spreadsheet with 16 rows and 7 columns. Row 1 contains "Hello" in cell A1. Rows 2 through 11 contain the numbers 0 through 9 respectively in cell A2-A11. The rest of the cells are empty. On the right, there is an "APSO console" window titled "APSO python console [LibreOffice]". It displays Python code used to set the values in the spreadsheet. The code is as follows:

```
APSO python console [LibreOffice]
3.11.4 (main, Jun  9 2023, 07:59:55) [GCC 12.3.0]
Type "help", "copyright", "credits" or "license" for more information.
>>> document = XSCRIPTCONTEXT.getDocument()
>>> data = [[i] for i in range(10)]
>>> sheet = document.Sheets[0]
>>> sheet["A1"].setString("Hello")
>>> sheet["A2:A11"].setDataArray(data)
>>> |
```

Creating Python Scripts

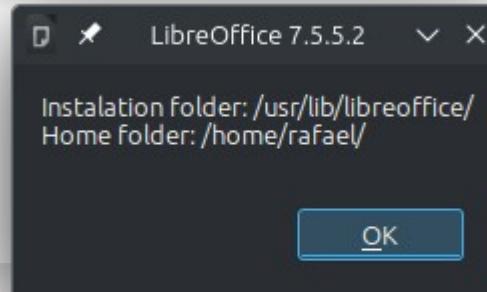
Where can we create user scripts that can be used across documents?

- ▼ 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):

```
5 Sub GetFolders
6     Dim fs as Object, sMessage As String
7     fs = CreateScriptService("FileSystem")
8     ' Use native OS file naming notation
9     fs.FileNaming = "SYS"
10    sMessage = "Instalation folder: " & fs.InstallFolder & CHR$(13) &
11        "Home folder: " & fs.HomeFolder
12    MsgBox sMessage
13 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)

```
1 from scriptforge import CreateScriptService  
2  
3 bas = CreateScriptService("Basic")  
4 ui = CreateScriptService("UI")  
5  
6 def create_writer_doc(args=None):  
7     name = bas.InputBox("What is your name?")  
8     doc = ui.createDocument("Writer")  
9     doc_component = doc.XComponent  
10    text = doc_component.getText()  
11    text.insertString(text.End, f"Hello {name}\n", False)  
12  
13 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

```
1 from scriptforge import CreateScriptService
2
3 bas = CreateScriptService("Basic")
4
5 def create_normdist_table(args=None):
6     # Get the Document service instance for the current component
7     doc = CreateScriptService("Document", bas.ThisComponent)
8     # If it is not a Calc document, do nothing
9     if not doc.isCalc:
10         return
11     # Insert the data in the sheet
12     doc.setValue("A1:B1", ["z-Value", "P(Z<z)"])
13     z_values = [-3 + z * 0.5 for z in range(13)]
14     z_range = doc.Offset("A2", height=len(z_values))
15     doc.setValue(z_range, z_values)
16     # Insert the formulas
17     formula_range = doc.Offset("B2", height=len(z_values))
18     base_formula = "=NORM.S.DIST(A2;1)"
19     doc.setFormula(formula_range, base_formula)
20
21 g_exportedScripts = (create_normdist_table, )
22
```

Creating Python Scripts

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

Continuation of
create_normdist_table

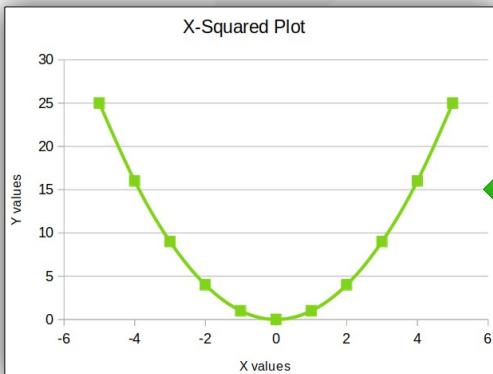
```
1  from scriptforge import CreateScriptService
2  from com.sun.star.awt import FontWeight
3  from com.sun.star.table import BorderLineStyle
4  from uno import createUnoStruct, Enum

31 # Format the header of the table
32 range_obj = doc.XCellRange("A1:B1")
33 range_obj.CellBackColor = bas.RGB(200, 200, 200)
34 range_obj.CharWeight = FontWeight.BOLD
35 # Format the remainder of the table
36 table_range = doc.Offset("A1", width=2, height=len(z_values)+1)
37 range_obj = doc.XCellRange(table_range)
38 range_obj.CharFontName = "Arial"
39 justify_center = Enum("com.sun.star.table.CellHoriJustify", "CENTER")
40 range_obj.HoriJustify = justify_center
41 # Struct that defines the line format
42 line_format = createUnoStruct("com.sun.star.table.BorderLine2")
43 line_format.LineStyle = BorderLineStyle.SOLID
44 line_format.LineWidth = 10
45 range_obj.TopBorder = line_format
46 range_obj.BottomBorder = line_format
47 range_obj.LeftBorder = line_format
48 range_obj.RightBorder = line_format
49
```

Creating Python Scripts

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

Instance of the "Chart" service

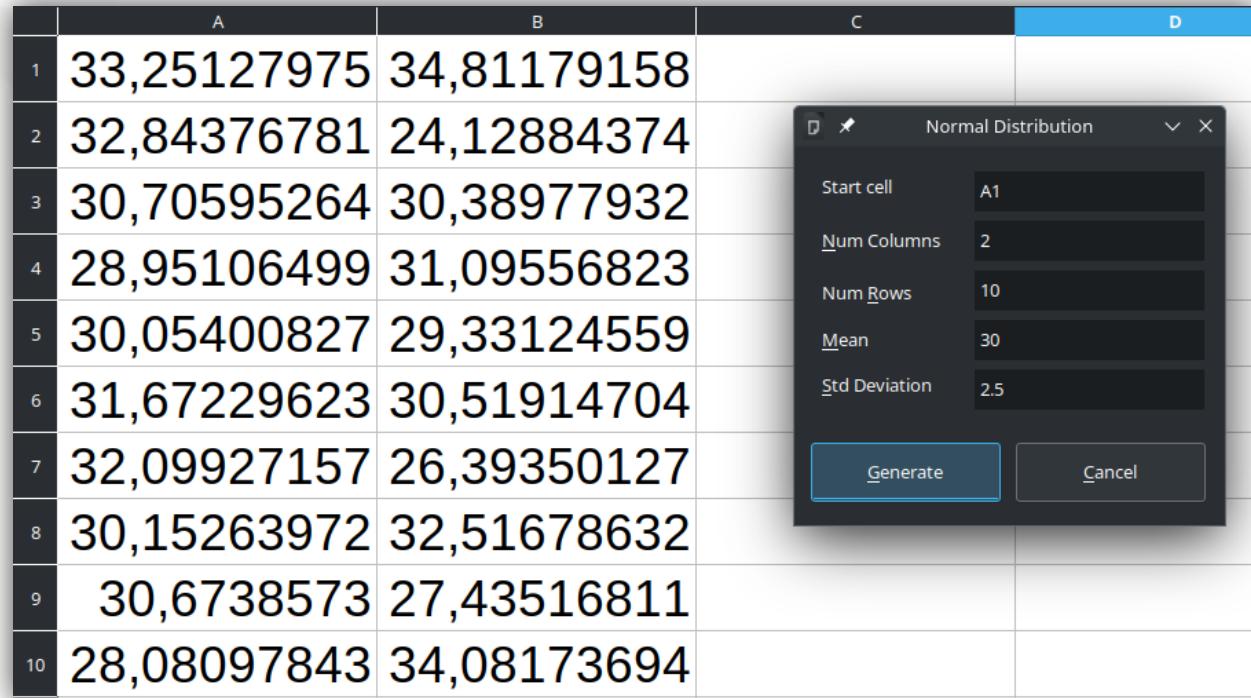


```
51 def plot_function(args=None):
52     # Plot X and Y values for X2 function
53     data = [[x, math.pow(x, 2)] for x in range(-5, 6)]
54     doc = CreateScriptService("Calc", bas.ThisComponent)
55     doc.setValue("A1:B1", [["X", "Y"]])
56     data_range = doc.Offset("A2", width=2, height=len(data))
57     doc.setValue(data_range, data)
58     # Select the entire table
59     table_range = doc.Region("A1")
59     # Insert the chart
60     cur_sheet = doc.SheetName(doc.CurrentSelection)
61     chart = doc.CreateChart("X-Squared", cur_sheet, table_range)
62     chart.ChartType = "XY"
63     chart.Legend = False
64     chart.Title = "X-Squared Plot"
65     chart.XTitle = "X values"
66     chart.YTitle = "Y values"
67     # Set the line type to "smooth" (using cubic splines)
68     diagram = chart.XDiagram
69     diagram.SplineType = Enum("com.sun.star.chart2.CurveStyle", "CUBIC_SPLINES")
70     # Place the Y-axis at -6 for better visualization
71     y_axis = diagram.getYAxis()
72     y_axis.CrossoverPosition = Enum("com.sun.star.chart.ChartAxisPosition", "VALUE")
73     y_axis.CrossoverValue = -6
74
```

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

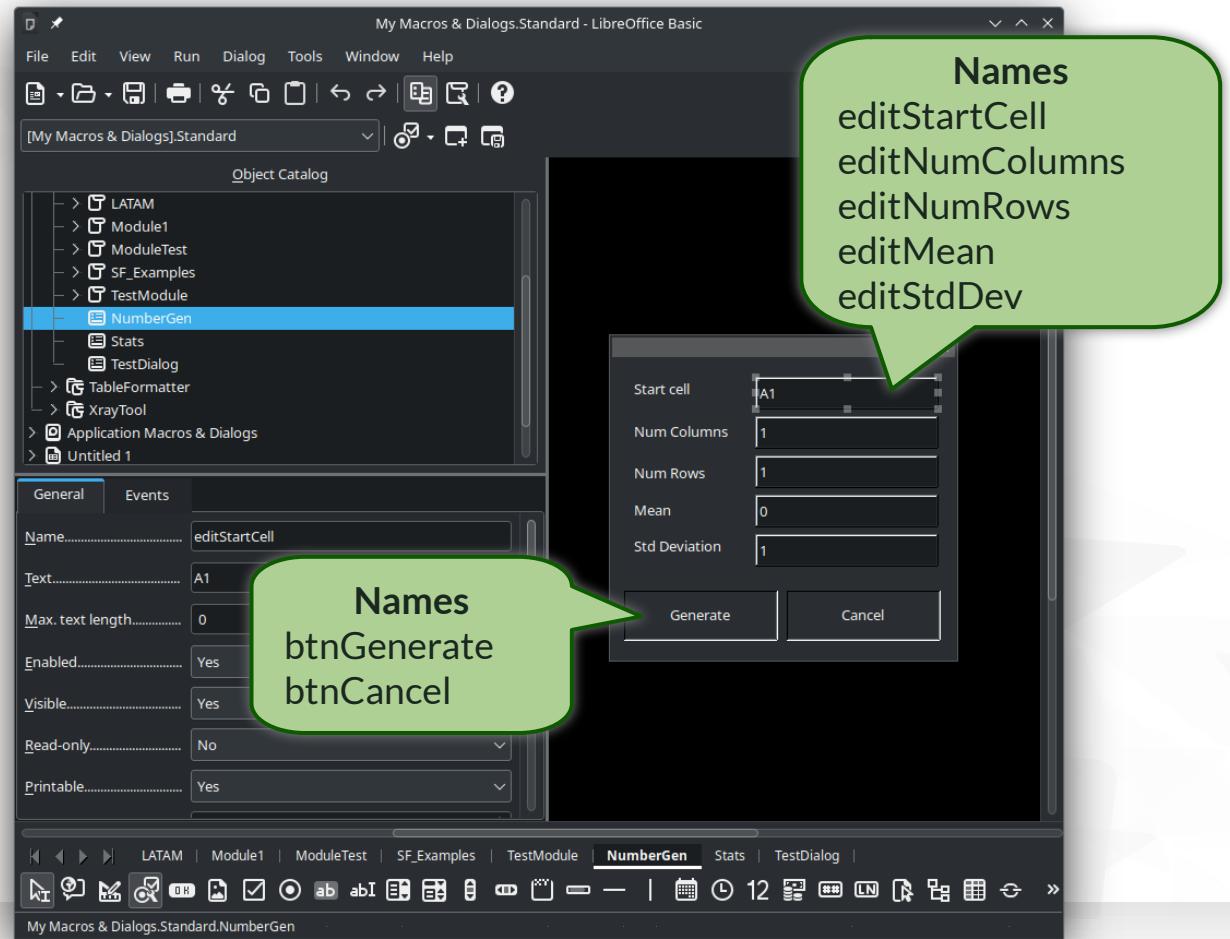
	A	B	C	D
1	33,25127975	34,81179158		
2	32,84376781	24,12884374		
3	30,70595264	30,38977932		
4	28,95106499	31,09556823		
5	30,05400827	29,33124559		
6	31,67229623	30,51914704		
7	32,09927157	26,39350127		
8	30,15263972	32,51678632		
9	30,6738573	27,43516811		
10	28,08097843	34,08173694		



The image shows a screenshot of LibreOffice Calc. On the left, there is a table with 10 rows and 2 columns, labeled A and B. The first column contains values like 33,25127975 and 32,84376781. The second column contains values like 34,81179158 and 24,12884374. To the right of the table is a configuration dialog box titled "Normal Distribution". The dialog has fields for "Start cell" (A1), "Num Columns" (2), "Num Rows" (10), "Mean" (30), and "Std Deviation" (2.5). It also has "Generate" and "Cancel" buttons.

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



Creating Python Scripts

- Now let's create a Python file `numbergen.py` with the macros that open the dialog and close it when "Cancel" is pressed

```
1 from scriptforge import CreateScriptService
2 import random as rnd
3
4 bas = CreateScriptService("Basic")
5
6 def open_dialog(args=None):
7     dlg = CreateScriptService("Dialog", "GlobalScope", "NumberGenerator", "NumberGenDlg")
8     dlg.Execute()
9
10 def btn_cancel_click(event=None):
11     # Get the control that was clicked
12     control = CreateScriptService("DialogEvent", event)
13     # Get the parent dialog and terminate it
14     dlg = control.Parent
15     dlg.EndExecute(bas.IDCANCEL)
16
```

Creating Python Scripts

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

```
17 def btn_generate_click(event=None):
18     # Get the control that was clicked
19     control = CreateScriptService("DialogEvent", event)
20     dlg = control.Parent
21     # Get the parameters from the dialog
22     start_cell = dlg.Controls("editStartCell").Value
23     num_cols = int(dlg.Controls("editNumColumns").Value)
24     num_rows = int(dlg.Controls("editNumRows").Value)
25     mean = float(dlg.Controls("editMean").Value)
26     std_dev = float(dlg.Controls("editStdDev").Value)
27     # Generate the numbers
28     numbers = [[rnd.gauss(mean, std_dev) for _ in range(num_cols)] for _ in range(num_rows)]
29     # Get the document and add the data
30     doc = CreateScriptService("Document", bas.ThisComponent)
31     num_range = doc.Offset(start_cell, width=num_cols, height=num_rows)
32     doc.setValue(num_range, numbers)
```

Thank you ...



LibreOffice
The Document Foundation

- ▶ The ScriptForge team
 - Jean-Pierre Ledure, Rafael Lima, Alain Romedenne
- ▶ Documentation
 - https://help.libreoffice.org/latest/en-US/text/sbasic/shared/03/lib_ScriptForge.html
- ▶ ScriptForge Sources
 - <https://gitlab.com/LibreOfficant/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.