LibreOffice
The Document Foundation

LibreOffice
Conference 2021

Python scripts in LibreOffice Calc using the ScriptForge library

Rafael Lima

Outline

- What is ScriptForge?
- Overview of the Calc service
- Reading and Writing data to ranges
- Clearing Cell Contents
- Copying and Moving Ranges
- Managing Sheets
- Basic Calc functions
- Loading CSV files

All examples use Python

Download examples at:
https://github.com/rafaelhlima/LibOCon_2021_SFCalc

+

+

What is ScriptForge?

- The ScriptForge library provides a set of services that makes scripting easier and more accessible for LibreOffice users

Array	DialogControl
Dictionary	UI
Exception	Form
FileSystem	FormControl
String	Basic
TextStream	L10N
Base	Platform
Calc	Services
Database	Session
Document	Timer
Dialog	

Currently there are 21 services related to various aspects of macro programming

ScriptForge is available since LibreOffice 7.1 for BASIC

Python support is available since LibreOffice 7.2

Why use ScriptForge?

- The ScriptForge library hides API and UNO complexity, which is a challenge for those starting to write macros in LibreOffice
- **Example:** Checking if the current document is a Calc document

```
# Gets the current document
doc = XSCRIPTCONTEXT.getDocument()
# Checks if it is a Calc document
isCalc = oDoc.supportsService(
    "com.sun.star.sheet.SpreadsheetDocument")
if isCalc:
   # do something
else:
   return
```

Usual approach

```
# Gets the current document
doc = CreateScriptService("Calc")
# Checks if it is a Calc document
if doc.IsCalc:
   # do something
else:
   return
```

Using ScriptForge

ScriptForge Calc Service

- The Calc service currently has 30 methods and 11 properties in addition to the methods and properties provided by the Document service

Activate	DMin	Offset
ClearAll	DSum	RemoveSheet
ClearFormats	Forms	RenameSheet
ClearValues	GetColumnName	SetArray
CopySheet	GetFormula	SetValue
CopySheetFromFile	GetValue	SetCellStyle
CopyToCell	ImportFromCSVFile	SetFormula
CopyToRange	ImportFromDatabase	SortRange
DAvg	InsertSheet	
DCount	MoveRange	
DMax	MoveSheet	

ScriptForge Calc Service

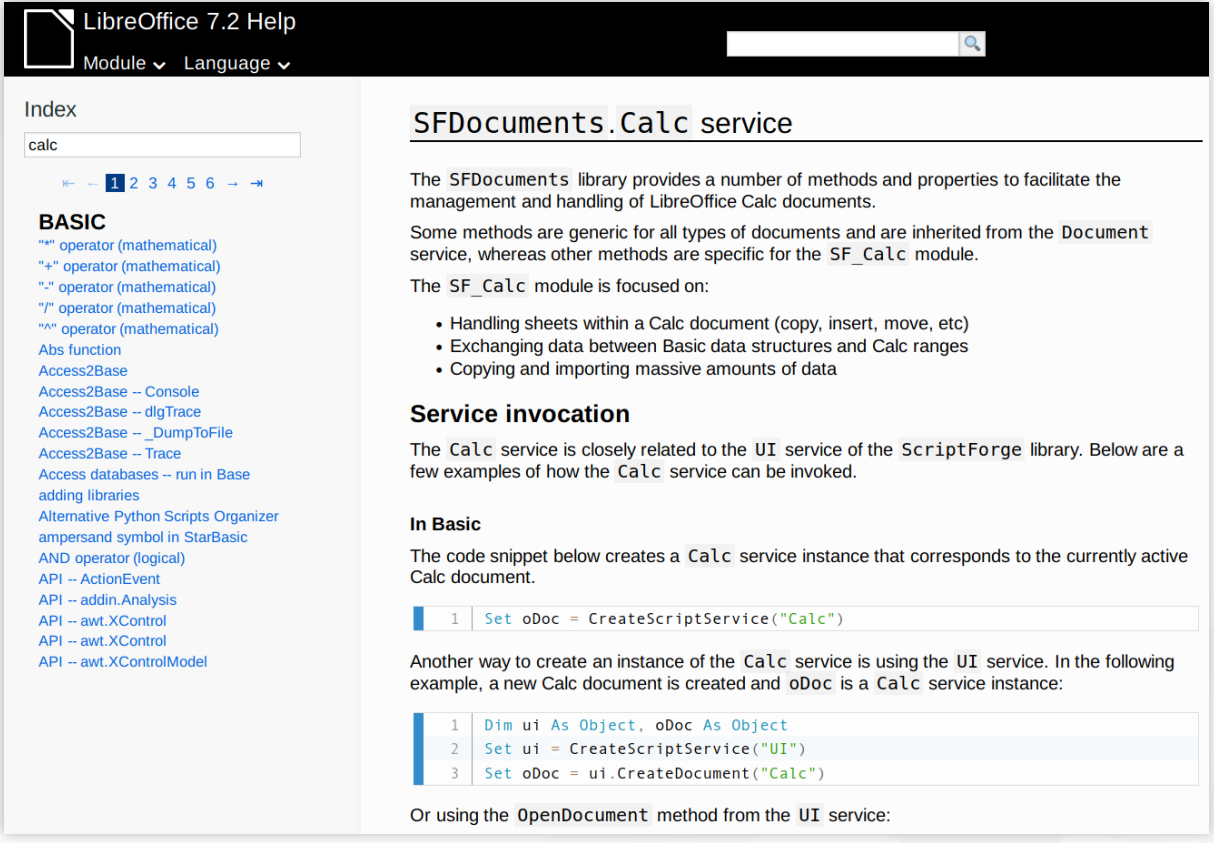
- Documentation is provided in the Online Help

ScriptForge Library

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

Calc Service

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



LibreOffice 7.2 Help

Module Language

Index

calc

1 2 3 4 5 6

BASIC

"*" operator (mathematical)
"+" operator (mathematical)
"-" operator (mathematical)
"/" operator (mathematical)
"^" operator (mathematical)
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 StarBasic
AND operator (logical)
API -- ActionEvent
API -- addin.Analysis
API -- awt.XControl
API -- awt.XControl
API -- awt.XControlModel

SFDocuments.Calc service

The SFDocuments library provides a number of methods and properties to facilitate the management and handling of LibreOffice Calc documents.

Some methods are generic for all types of documents and are inherited from the Document service, whereas other methods are specific for the SF_Calc module.

The SF_Calc module is focused on:

- Handling sheets within a Calc document (copy, insert, move, etc)
- Exchanging data between Basic data structures and Calc ranges
- Copying and importing massive amounts of data

Service invocation

The Calc service is closely related to the UI service of the ScriptForge library. Below are a few examples of how the Calc service can be invoked.

In Basic

The code snippet below creates a Calc service instance that corresponds to the currently active Calc document.

```
Set oDoc = CreateScriptService("Calc")
```

Another way to create an instance of the Calc service is using the UI service. In the following example, a new Calc document is created and oDoc is a Calc service instance:

```
Dim ui As Object, oDoc As Object
Set ui = CreateScriptService("UI")
Set oDoc = ui.CreateDocument("Calc")
```

Or using the OpenDocument method from the UI service:

Reading/Writing Values

- The code below creates a 4x4 matrix starting at cell “A1” with random “ODD” and “EVEN” strings based on randomly generated data

```
import random as rnd

def create_random_matrix_v1(args=None):
    doc = CreateScriptService("Calc")
    for i in range(4):
        for j in range(4):
            target_cell = doc.Offset("A1", i, j)
            r = rnd.random()
            if r < 0.5:
                doc.setValue(target_cell, "EVEN")
            else:
                doc.setValue(target_cell, "ODD")
```

Cell “A1” offset by *i* rows and *j* columns

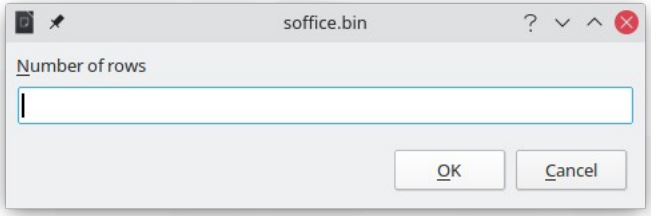
Sets the value in the cell referred in target_range

Reading/Writing Values

- The code below creates a *n x m* matrix starting at cell “A1”. It uses the “Basic” service to ask the user about the desired size:

```
def create_random_matrix_v2(args=None):
    doc = CreateScriptService("Calc")
    bas = CreateScriptService("Basic")
    n_rows = bas.InputBox("Number of rows")
    n_cols = bas.InputBox("Number of columns")
    for i in range(int(n_rows)):
        for j in range(int(n_cols)):
            target_cell = doc.Offset("A1", i, j)
            r = rnd.random()
            if r < 0.5:
                doc.setValue(target_cell, "EVEN")
            else:
                doc.setValue(target_cell, "ODD")
```

The InputBox method belongs to the “Basic” service



Reading/Writing Values

- It is always faster to use `setArray`!

```
def create_random_matrix_v3(args=None):
    doc = CreateScriptService("Calc")
    bas = CreateScriptService("Basic")
    n_rows = bas.InputBox("Number of rows")
    n_cols = bas.InputBox("Number of columns")
    rnd_word = lambda : "EVEN" if rnd.random() < 0.5 else "ODD"
    values = [[rnd_word() for _ in range(int(n_cols))]
              for _ in range(int(n_rows))]
    doc.setArray("A1", values)
```

The method setArray expands according to the size of the array

Reading/Writing Values

- This code iterates over the current selection, replaces negative values with the string “INVALID” and sets a different cell style

```
def mark_invalid(args=None):
    doc = CreateScriptService("Calc")
    # Gets address of current selection
    cur_selection = doc.CurrentSelection
    # Gets address of first cell in the selection
    first_cell = doc.Offset(cur_selection, 0, 0, 1, 1)
    for i in range(doc.Height(cur_selection)):
        for j in range(doc.Width(cur_selection)):
            cell = doc.Offset(first_cell, i, j)
            value = doc.getValue(cell)
            if value < 0:
                doc.setValue(cell, "INVALID")
                doc.setCellStyle(cell, "Bad")
```

Gets the address of the current selection

Applies the “Bad” cell style

Clearing Cell Contents

- There are three methods to clear cell values:
  - `ClearAll(range: str)`
    - All cell contents and formats
  - `ClearFormats(range: str)`
    - Cell formats (values are kept)
  - `ClearValues(range: str)`
    - Cell contents (formats are kept)

```
def clear_contents_v1(args=None):
    doc = CreateScriptService("Calc")
    doc.clearAll("H1:H5")
```

```
def clear_contents_v2(args=None):
    doc = CreateScriptService("Calc")
    doc.clearFormats("H1:H5")
```

```
def clear_contents_v3(args=None):
    doc = CreateScriptService("Calc")
    doc.clearValues("H1:H5")
```

Copying and Moving Ranges

- There are three methods to copy or move ranges:
  - `CopyToCell(source, destination)`
    - All contents in **source** are copied into the **destination**. The size of the affected destination is equal to the size of **source**
  - `CopyToRange(source, destination)`
    - Equivalent to pasting contents when a range of cells is selected as **destination** and it is larger than **source**
  - `MoveRange(source, destination)`
    - All contents in **source** are moved into the **destination**. The size of the affected **destination** is equal to the size of the **source**

Copying and Moving Ranges

- Examples using `CopyToCell` and `CopyToRange`

```
def copy_cells_v1(args=None):
    doc = CreateScriptService("Calc")
    doc.copyToCell("A1:A4", "C1")
```

If the destination were a range (f.i. “B1:D10”, only the top-left cell would be considered”)

```
def copy_cells_v2(args=None):
    doc = CreateScriptService("Calc")
    doc.copyToRange("A1:A4", "E1:F6")
```

Note that the destination range is larger than the source

Copying and Moving Ranges

- A more complex example with `CopyToCell`:
  - This example copies from a different file that is currently open

```
def copy_range_example_v2(args=None):
    # Reference to current document (destination)
    doc = CreateScriptService("Calc")
    # Reference to the source document
    svc = CreateScriptService("UI")
    source_doc = svc.getDocument("DataSource.ods")
    source_range = source_doc.Range("Sheet1.A1:A5")
    # Pastes the contents into the destination
    doc.copyToCell(source_range, "A1")
```

The “UI” service provides access to other documents that are currently open

Pastes into cell “A1” of the current sheet in the current document

Managing Sheets

- There following methods are used to handle sheets
  - `Activate(sheetname)`
  - `CopySheet(sheetname, newname, [beforesheet])`
  - `InsertSheet(sheetname, [beforesheet])`
  - `MoveSheet(sheetname, [beforesheet])`
  - `RemoveSheet(sheetname)`
  - `RenameSheet(sheetname, newname)`

Managing Sheets

- Examples using `InsertSheet`, `Activate` and `CopySheet`

```
def create_sheet_example(args=None):
    doc = CreateScriptService("Calc")
    doc.insertSheet("TestSheet", 2)
    doc.activate("TestSheet")
```

It can also be a string with the name of an existing sheet

Activating the sheet is equivalent to selecting its tab

```
def copy_sheet_example(args=None):
    doc = CreateScriptService("Calc")
    doc.copySheet("TestSheet", "Copy_TestSheet")
```

```
def remove_sheet_example(args=None):
    doc = CreateScriptService("Calc")
    doc.removeSheet("Copy_TestSheet")
```

Managing Sheets

- `CopySheetFromFile`: copies a sheet from a Calc file. The file can be currently **open** or **closed**.

```
def copy_from_file_example(args=None):
    doc = CreateScriptService("Calc")
    doc.copySheetFromFile("/home/rafael/Documents/DataSource.ods",
                          "Sheet2", "Copy_Sheet2")
```

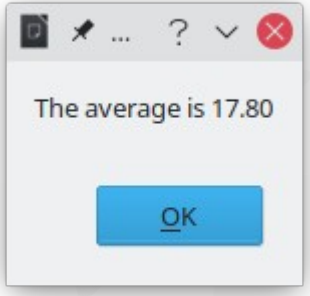
- If the file does not exist, an error is raised.
- If the file is not a valid Calc file, a blank sheet is inserted.
- If the source sheet does not exist in the input file, an error message is inserted at the top of the newly pasted sheet.

Basic Calc Functions

- `DAvg, DCount, DMax, DMin, DSum`: Calls the following Calc functions:

Method	Calc function
DAvg	AVERAGE
DCount	COUNT
DMax	MAX
DMin	MIN
DSum	SUM

```
def calculate_average(args=None):
    doc = CreateScriptService("Calc")
    bas = CreateScriptService("Basic")
    result = doc.DAvg("A1:E1")
    bas.MsgBox("The average is {:.02f}".format(result))
```



Loading CSV Files

- `ImportFromCSVFile`: Loads a CSV text file and places the contents into the destination cell

```
def open_csv_file_v1(args=None):
    doc = CreateScriptService("Calc")
    doc.ImportFromCSVFile("/home/rafael/Documents/JobData_v1.csv", "A1")
```

- The input file encoding is UTF8.
- The field separator is a comma, a semi-colon or a Tab character.
- The string delimiter is the double quote (").
- All lines are included.
- Quoted strings are formatted as text.

Loading CSV Files

```
def open_csv_file_v3(args=None):
    doc = CreateScriptService("Calc")
    filter_option = "59,34,UTF-8,1"
    doc.ImportFromCSVFile("/home/rafael/Documents/JobData_v2.csv",
                          "A1", filter_option)
```

Text delimiter = double quotes (34 in ASCII)

Number of first line

```
filter_option = "59,34,UTF-8,1"
```

Field separator = ';' (59 in ASCII)

Character set

LibreOffice
The Document Foundation

Thank you …

- The team
  - Jean-Pierre Ledure, Rafael Lima, Alain Romedenne
- Documentation
  - https://help.libreoffice.org/7.2/en-US/text/sbasic/shared/03/lib_ScriptForge.html?DbPAR=BASIC
- Sources
  - https://gitlab.com/LibreOfficiant/scriptforge
  - Gerrit
- Telegram groups
  - ScriptForge
  - LibreOffice Macros & Scripting

Finish