LibreOffice Reference Card

LibreOffice Basic
Calc

v. 1.11 – January 13, 2018

Beginner

Written with LibreOffice v. 5.3.3 – Platform : All

LibreOffice Documents

Current document

```
Dim Doc As Object
Doc = ThisComponent
```

Open another existing document

Visible mode

```
Dim Doc As Object
Dim PathDoc As String
Dim Props()   'here, this table is not initialized
PathDoc = ConvertToURL("C:\Path\To\CalcFile.ods")
Doc = StarDesktop.loadComponentFromURL(PathDoc, "_blank", 0, _
                                       Props())
```

Invisible mode

```
Dim Doc As Object
Dim PathDoc As String
Dim Props(0) As New com.sun.star.beans.PropertyValue
PathDoc = ConvertToURL("C:\Path\To\CalcFile.ods")
Props(0).Name  = "Hidden"  'the document will open hidden"
Props(0).Value = True
Doc = StarDesktop.loadComponentFromURL(PathDoc, "_blank", 0, _
                                       Props())
```

Turn visible *a posteriori*

```
Doc.CurrentController.Frame.ContainerWindow.Visible = True
Doc.CurrentController.Frame.ContainerWindow.toFront()
```

Create a new Calc document

From (1) the default template or (2) a specific template.

```
Dim Doc As Object
Dim Props()   'here, this table is not initialized
Model = "private:factory/scalc"                    '(1)
'or
Model = "C:\Path\To\ACalcTemplateFile.ots"  '(2)
Doc = StarDesktop.loadComponentFromURL(Model, "_blank", 0, Props())
```

Save a document

The document already exist

(equivalent to **File > Save**)

Use the method `store` from the document object. Ex : `ThisComponent.store`

The document was not yet saved

(equivalent to **File > Save as**)

```
Dim Doc As Object       'the object document to store
Dim PathDoc As String 'the path for saving
Dim Props()             'the saving properties. (empty)
PathDoc = ConvertToURL("C:\Path\To\CalcFile.ods")
Doc.storeAsURL(PathDoc, Props())
```

☞ If a copy, it **turns** to active document

Save a copy

Like above but with `Doc.storeToURL(PathDoc, Props())`

☞ The copy does **not** become the active document.

Close a document

Use the method `close` from the document object: `ThisComponent.close(True)`

Document information

The document object expose properties

`Location`	The folder of the document. ☞ Empty string is not yet saved
`DocumentProperties (Object)`	Additional properties (below).

DocumentProperties

`Author`	Author's name	`ModifyDate`	Last modification date
`CreationDate`	Creation date.	`Subject`	Document subject (string).
`Description`	Document description	`Title`	Document title
`ModifiedBy`	User name who modified the document.	`UserDefined-Properties`	Custom properties (Object).

Is it a Calc document?

The `Doc` object points to the document (ex: `Doc=ThisComponent`).

```
CalcOK = Doc.SupportsService("com.sun.star.sheet.SpreadsheetDocument")
```

Calc – General functionalities

The `Doc` object points to the document (ex: `Doc=ThisComponent`).

Automatic calculation

Active? (Boolean)	`Auto = Doc.isAutomaticCalculationEnabled`
Disable	`Doc.enableAutomaticCalculation(False)`
Enable	`Doc.enableAutomaticCalculation(True)`
Force recalculation	`Doc.calculate` (only for formulas not updated) `Doc.calculateAll` (all formulas)

Protect document

Is document protected?	`Test = Doc.isProtected`
Protect document	`Doc.protect(password)` [password can be empty]
Unprotect document	`Doc.unprotect(password)`

Sheets (*sheets*)

The `Doc` object points to the document (ex: `Doc=ThisComponent`).

Access to sheets

Work with Sheets objects:	
Active sheet	`MySheet = Doc.CurrentController.ActiveSheet`
Sheet list	`AllSheets = Doc.Sheets`
Number of sheets	`NumberSheets = Doc.Sheets.Count`
Sheet object (by index [base 0])	`MySheet = Doc.Sheets(index)`
Sheet object (by name)	`MySheet = Doc.Sheets.getByName("SheetName")`
Check existence (name)	`Exist = Doc.Sheets.hasByName("SheetName")`
Sheet index	`Index = MySheet.RangeAddress.Sheet`

Modify sheets

Add a sheet named `Name` at position `p`(base 0)	`Doc.Sheets.insertNewByName(Name, p)`
Delete a sheet	`Doc.Sheets.removeByName("SheetName")`
Duplicate a sheet to the position `p` (base 0)	`Doc.Sheets.copyByName("SourceName","Target-Name", p)`
Move sheet to the position `p` (base 0)	`Doc.Sheets.moveByName(SheetName, p)`

Manage sheets

`MySheet` is a sheet object.

Activate sheet	`Doc.CurrentController.ActiveSheet = MySheet`
Protect sheet (`password` can be empty)	`MySheet.protect(password)`
Unprotect sheet	`MySheet.unprotect(password)`
Tab color	`MySheet.tabColor = RGB(255, 255, 0)`

Link a sheet

Link to a file (ex: CSV)	`MySheet.link(URL, "", "Text - txt - csv (StarCalc)", _` `Filter, com.sun.star.sheet.SheetLinkMode.VALUE)`
Break a link	`MySheet.setLinkMode(com.sun.star.sheet.SheetLinkMode.NONE)`

Find last row/column used

`MySheet` is the sheet object to explore. `Row` and `Col` are information to fetch.

```
Dim Cur As Object           ' cursor on the cell
Dim Range As Object         ' the used range
Dim Row As Long
Dim Col As Long
Cur = MySheet.createCursorByRange(MySheet.getCellRangeByName("A1"))
Cur.gotoEndOfUsedArea(True)
Range = MySheet.getCellRangeByName(Cur.AbsoluteName)
Row = Range.RangeAddress.EndRow
Col   = Range.RangeAddress.EndColumn
```

Cells (*cells*)

Below `Cel` is an cell object.

Access to cells

`MySheet` is a sheet object. Access to cell object:

By cell default notation	`Cel = MySheet.getCellRangeByName("A4")`
By name	`Cel = MySheet.getCellRangeByName("TVA")`
By coordinates X and Y	`Cel = MySheet.getCellByPosition(0,3)` `Wih X=0 (col.A) ; Y=3 (row.4)`

Access to active cell

`Doc` is an document object and `ActiveCel` the active cell object.

```
If Doc.currentSelection.supportsService _
                        ("com.sun.star.sheet.SheetCell") Then
  'It's a cell
  ActiveCel = Doc.currentSelection
End If
```

Select a cell

```
ThisComponent.CurrentController.select(Cel)
```

Cell coordinates

Coordinates (Object)	`Coord = Cel.CellAddress`
Sheet index (Integer)	`NumS  = Cel.CellAddress.Sheet`
Columns index (Long)	`NumC  = Cel.CellAddress.Column`
Row index (Long)	`NumL  = Cel.CellAddress.Row`
Sheet container object	`MySheet = Cel.Spreadsheet`
Absolute coordinates (String)	`Coord = Cel.AbsoluteName`

(un)protect cells

`Cel.CellProtection` can take boolean values:

Prevent modification	`CellProtection.IsLocked = True`
Hide cell formula	`CellProtection.IsFormulaHidden = True`
Hide cell	`CellProtection.IsHidden = True`
Don't print cell	`CellProtection.IsPrintHidden = True`

Access cell contents

Properties

Read text contents	`MyText = Cel.String`
Read numeric contents	`aNumeber = Cel.Value`
Read cell formula (en-US names)	`TheFormula = Cel.Formula`
Read cell formula (localized names)	`LaFormule = Cel.FormulaLocal`
Cell type	`TheType = Cel.Type`
Empty a cell	`Cel.String = ""`

Contents type (`Type` property)

The constants `com.sun.star.table.CellContentType.XXX` represent the cell information type (`Cel.Type`, above) :

`EMPTY`	Empty cell	`VALUE`	Numerical value
`TEXT`	Text contents	`FORMULA`	Formula contents

Write in a cell

Replace existing text	`Cel.String = "Hello !"`
Replace an existing value	`Cel.Value = 1.234`
Replace an existing formula (localized)	`Cel.Formula = "=AND(A1="YES";A2="OK")"`
Replace an existing formula (localized)	`Cel.FormulaLocal = "=ET(A1="OUI";A2="OK")"`

Ranges (*ranges*)

Range = set of cells, (including a single one): `Dim MyRange As Object`

Access to ranges

`MySheet` is a sheet object. Get a range object `Ran`>:

By cell default notation	`Ran = MySheet.getCellRangeByName("C2:G14")`
By name	`Ran = MySheet.getCellRangeByName("RangeName")`
By coordinates (X1, Y1, X2, Y2)	`Ran = MySheet.getCellRangeByPosition(2, 1, 6, 13)`
Randomly (ex third sheet)	`Ran = ThisComponent.Sheets.getCellRangeByPosition(2, 2, 1, 6, 13)`

Active range

Like active cell, but check `"com.sun.star.sheet.SheetCellRange"` or `"[...].SheetCellRanges"`.

Range selection

`ThisComponent.CurrentController.select(MyRange)` where `MyRange` is an objetc.

Range coordinates

Coordinates (Object)	`Coord = MyRange.RangeAddress`
Sheet index (Integer)	`Ran  = MyRange.RangeAddress.Sheet`
Column rank (Long) top/left corner	`NumCHG = MyRange.RangeAddress.StartColumn`
Row rank (Long) top/left corner	`NumLHG = MyRange.RangeAddress.StartRow`
Column rank (Long) bottom/right corner	`NumCBD = MyRange.RangeAddress.EndColumn`
Row rank (Long) bottom/right corner	`NumLBD = MyRow.RangeAddress.EndRow`
Sheet container object	`MySheet = MyRange.Spreadsheet`
Absolute coordinates (String)	`Coord = MyRange.AbsoluteName`

Named ranges

The `Doc` object points to the document. With `Dim TheRanges As Object`

Named ranges	`TheRanges = Doc.NamedRanges`
Number (Long)	`Nb = TheRanges.Count`
Get a range (by index)	`MyRange = TheRanges(index)`
Check existence (name)	`Exist = TheRanges.hasByName("RangeName")`
Get range (by name)	`MyRange = TheRanges.getByName("RangeName")`
Add `Coord` : range coordinates `CellRef` : reference cell object	`TheRanges.addNewByName("Rangename", Coord, _` `CellRef.CellAddress, 0)`
Delete (by name)	`TheRanges.removeByName("RangeName")`

Erase a range

Erase `MyRange` contents `MyRange.clearContents(EraseMode)`

`EraseMode` is a value that define the type of cleaning. Use `com.sun.star.sheet.CellFlags.XXX` and combine them with `+`) :

`ANNOTATION`	Comments	`STRING`	Text
`DATETIME`	Date/time formatted numbers	`VALUE`	Numbers (except date/time)
`FORMULA`	Formulae		

Get cell contents in a range

`MyRange.DataArray` is a table of cell values for `MyRange`.

Copy range contents into another range

Have 2 ranges `Source` and `Target`, **with same dimensions**.

Copy contents (**values**) from `Source` into `Target.DataArray = Source.DataArray`

`Target`.

Write values in a range

`MyRange` is a range object and `MyTable` a table, **with same dmensions**, where values must be transfered to the range.

```
Dim MyTable As Variant
MyTable = MyRange.DataArray 'MyTable takes the range dimensions
'(give values to the tabel elements)
MyRange.DataArray = MyTable
```

☞ `.DataArray` is an embedded table: use `.DataArray(i)(j)`

Traverse cells in a range

From a collection (`MyRanges.Cells`) create an `enumeration`. Traverse the range calling its properties `hasMoreElements` and `NextElement`:

```
Dim Plages As Object
MyRanges =
    ThisComponent.createInstance("com.sun.star.sheet.SheetCellRanges")
MyRanges.insertByName("", MyRange)
LEnum = MyRanges.Cells.CreateEnumeration
Do While LEnum.hasMoreElements
  MyCell = LEnum.NextElement
  ' apply instructions to object cell
Loop
```

Empty cells **are not** traversed !

Ranges: Miscellaneous

Merge cells of `MyRange` `MyRange.Merge`

Range types

Depending of the access mode, ranges implement one of these services:

① `com.sun.star.sheet.SheetCell`
② `com.sun.star.table.CellRange`
③ `com.sun.star.sheet.NamedRange`
④ `com.sun.star.sheet.SheetCellRange`
⑤ `com.sun.star.sheet.SheetCellRanges`

Depending on the service implemented, ranges must be employed **differently**.
Test through method `supportsService()` (ex. below)

Cell or range?

To know the object type, test `supportsService()` with `service_name` below (boolean) on the object (range or cell):

`If MyObj.supportsService(service_name) Then …`

Replace `service_name` by :

Cell ?	`"com.sun.star.sheet.SheetCell"` ①
Single range ?	`"com.sun.star.sheet.SheetCellRange"` ④
Multiple range ?	`"com.sun.star.sheet.SheetCellRanges"` ⑤

Always test a cell **before** a single range because a cell is **also** a single range !

Rows/Columns (*rows/columns*)

Rows and columns are Sheet and Range objects properties.

General

Rows (`TheRows` object)	`TheRows = MyRange.Rows`
Columns (`TheCols` object)	`TheCols = MyRange.Columns`
Counting	`NbL = MyRange.Rows.Count` `NbC = MyRange.Columns.Count`
A row (`TheRow` object (base 0)	`TheRow = MyRange.Rows(index)`
A column (`TheCol` object (base 0)	`TheCol = MyRange.Columns(index)`

Row/Columns properties

Applies to Row or Rows (resp. Column or Columns).

Visible / Hidden (Boolean)	`IsVisible = True`
Optimal width (Boolean)	`OptimalWidth = True`

Insert/delete rows/columns

Define object `RorC`, and `FirstPos` and `LastPos` the positions of the begining and end of the row set (resp. columns) to add/delete (Long).

Insert	`RorC.insertByIndex(FirstPos, LastPos)`
Delete	`RorC.removeByIndex(FirstPos, LastPos)`

Freeze row/columns

Use the Controller object: `MyController = ThisComponent.CurrentColtroller`

Is there one?	`Freeze = MyController.hasFrozenPanes`
Freeze	`MyController.freezeAtPosition(1, 2)`
Delete	`MyController.freezeAtPosition(0, 0)`

Call a Calc function

Use service `"com.sun.star.sheet.FunctionAccess"`

Usage

```
Dim FCalc As Object
Dim Result As (context dependent)
Dim Params As (context dependent)
Dim FunctionName As String
FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
Results = FCalc.callFunction(FunctionName, Params)
```

☞ Function name, parameters and type of results depends on the selected function

The function name must be its **English** name.
To get the function English name, switch temporarily to English Calc function names display, at **Tools > Options > LibreOffice Calc > Formula**, **Use English function names**.

Example 1 (function `SUM()`)

```
Dim FCalc As Object
Dim Results As Long
FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
Results = FCalc.callFunction("SUM", Array(1, 55, 321, 8))
```

Example 2 (function `ROUND()`)

```
Dim FCalc As Object
Dim Results As Double
Dim Params(1) As Variant
Params(0) = 1,2345 'number to round
Params(1) = 3      '3 places
FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
Results = FCalc.callFunction("ROUND", Params())
```

Create a Calc function

Create

Example : calculate the area of a trapeze (`S = ((B + b) / 2)  × H`)

```
Function AreaTrapeze(GB As Double, PB As Double, H As Double) As
    Double
  AreaTrapeze = ((GB + PB) / 2) * H
End Function
```

Usage in Calc

If `A2` is the large base, `A3` the small base and `A4` the height, the area of the trapeze is obtained inserting the following formula in a cell: `=AREATRAPEZE(A2;A3;A4)`

☞ The macro receives the **values** of the arguments and not the cell object.

☞ The macro **returns a value**. It **does not** applies to a cell.

☞ The function must be located in a library accessible at runtime (e.g.`Standard` library of the document or user) (otherwise, it produces the `#VALUE!` error).

Credits

Author : Jean-François Nifenecker – jean-francois.nifenecker@laposte.net

We are like dwarves perched on the shoulders of giants. If we are able to see more and farther than the latter, and this is not at all because of the acuteness of our sight or the stature of our body, but because we are carried aloft and elevated by the magnitude of the giants (Bernard de Chartres [attr.])

Hisory

Version	Date	Comments
1.01	01/10/2017	First version.
1.11	13/01/2018	Add range types.