LibreOffice RefCard

LibreOffice Basic
Calc

Beginner

v. 1.15 – 12/14/2019

Written using LibreOffice v. 5.3.3 – Platform: All

LibreOffice Documents

Current Document

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

Opening an existing document

In Visible Mode

```
Dim Doc As Object, DocPath As String
Dim Props()   'here, an unitialized table
DocPath = ConvertToURL("C:\Path\To\SomeSpreadsheet.ods")
Doc = StarDesktop.loadComponentFromURL(DocPath, "_blank", 0, _
                                       Props())
```

In hidden Mode

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

Make visible an hidden document

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

Creating a new calc document

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

```
Dim Doc As Object
Dim Props()   'here, an unitialized table
Template = "private:factory/scalc"        '(1)
'or
Template = "C:\Path\To\SomeTemplate.ots"  '(2)
Doc = StarDesktop.loadComponentFromURL(Template, "_blank", 0, Props())
```

Saving a document

The document already exists

(same as **File > Save**)

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

The document wasn't saved yet

(same as à **File > Save As**)

```
Dim Doc As Object, DocPath As String
Dim Props()            'save properties (empty)
DocPath = convertToURL("C:\Path\To\SomeSpreadsheet.ods")
Doc.storeAsURL(DocPath, Props())
```

☞ In case of a duplicate, it **becomes** the active document.

Saving a copy

As above, using `Doc.storeToURL(DocPath, Props())`

☞ The copy **won't** become the active document.

Closing a document

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

Getting document information

The document object exposes properties

`Location`	The document storage directory ☞ 0-length string if not saved yet.
`DocumentProperties (Object)`	Some more information (see below).

DocumentProperties (File > Properties)

`Author`	Author's name.	`ModifyDate`	Last modification date.
`CreationDate`	Creation date.	`Subject`	Subject (String).
`Description`	Comments.	`Title`	Title (String).
`ModifiedBy`	Name of the user who modified the document.	`UserDefinedProperties`	Custom properties (Object).

Is this a Calc document?

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

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

Calc – General

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

Automatic calculation

Active? (Boolean)	`Auto = Doc.isAutomaticCalculationEnabled`
Deactivate	`Doc.enableAutomaticCalculation(False)`
Activate	`Doc.enableAutomaticCalculation(True)`
Force calculation	`Doc.calculate` (only non updated formulas) `Doc.calculateAll` (everything)

Protecting the spreadsheet

Is the document protected?	`Test = Doc.isProtected`
Protect document	`Doc.protect(Password) '[may be empty]`
Unprotect document	`Doc.unprotect(Password)`

Sheets

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

Getting sheets

Working with sheet objects (index is 0-based):

The active sheet	`ASheet = Doc.CurrentController.ActiveSheet`
The sheets list	`TheSheets = Doc.Sheets`
Sheet count	`SheetNum = Doc.Sheets.Count`
Sheet object (by index)	`ASheet = Doc.Sheets(index)`
Sheet object (by its name)	`ASheet = Doc.Sheets.getByName(SheetName)`
Check existence (name)	`Exists = Doc.Sheets.hasByName(SheetName)`
Sheet index	`SheetIndex = ASheet.RangeAddress.Sheet`

Modifiying sheets

Below `p` is the position in the spreadsheet (base 0).

Adding a named sheet `Name`	`Doc.Sheets.insertNewByName(Name, p)`
Deleting a sheet	`Doc.Sheets.removeByName(SheetName)`
Duplicating a sheet	`Doc.Sheets.copyByName(SrcName, TgtName, p)`
Moving a sheet	`Doc.Sheets.moveByName(SheetName, p)`

Managing sheets

`ASheet` is a sheet object.

Activating a sheet	`Doc.CurrentController.ActiveSheet = ASheet`
Hiding/showing a sheet	`ASheet.IsVisible = False 'True`
Checking protection	`Protected = ASheet.IsProtected`
Protecting a sheet (Pwd can be empty)	`ASheet.protect(Pwd)`
Unprotecting a sheet	`ASheet.unprotect(Pwd)`
Tab coloring	`ASheet.tabColor = RGB(255, 255, 0)`

Linking a sheet

Linking to a file (ex: `ASheet.link(URL, "", "Text - txt - csv (StarCalc)", _`
CSV)   `Filtre, com.sun.star.sheet.SheetLinkMode.VALUE)`

Destroying a link `ASheet.setLinkMode(com.sun.star.sheet.SheetLinkMode.NONE)`

Finding the last used row/Col

`ASheet` is a sheet object.. `Row` and `Col` are the values to retrieve.

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

Cells

Below, `ACell` is a cell object.

Getting cells

`ASheet` is asheet object. You can get the cell object:

Through its R/C coordinates	`ACell = ASheet.getCellRangeByName("A4")`
Through its name	`ACell = ASheet.getCellRangeByName("VAT")`

Through its XY coordinates	`ACell = ASheet.getCellByPosition(0,3)`
	`with X=0 (col.A) ; Y=3 (row 4)`

Knowing the active cell

`Doc` is a document object and `ACell` is the active cell we're looking for.

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

Selecting a cell

`ThisComponent.CurrentController.select(ACell)`

Cell Coordinates

Coordinates (Object)	`Coord   = ACell.CellAddress`
Sheet index (Integer)	`NumSh   = ACell.CellAddress.Sheet`
Column index (Long)	`NumCol  = ACell.CellAddress.Column`
Row index (Long)	`NumRow  = ACell.CellAddress.Row`
Container sheet object	`ASheet  = ACell.Spreadsheet`
Absolute coordinates (String)	`Coord   = ACell.AbsoluteName`

(Un)Protecting cells

The `ACell.CellProtection` property can get the boolean values:

No modifications	`CellProtection.IsLocked = True`
Hide formula	`CellProtection.IsFormulaHidden = True`
Hide cell	`CellProtection.IsHidden = True`
Don't print cell	`CellProtection.IsPrintHidden = True`

Getting a cell contents

Properties

Getting text contents	`MyText   = ACell.String`
Getting numerical contents	`ANumber  = ACell.Value`
Getting a formula (US)	`AFormula = ACell.Formula`
Getting a formula (local lang.)	`AFormula = ACell.FormulaLocal`
Knowing content type (below)	`TheType  = ACell.Type`
Emptying a cell	`ACell.String = ""`

Content Types (Type property)

The `com.sun.star.table.CellContentType.XXX` constants can tell the type of contents (`ACell.Type`, above):

`EMPTY`	Empty cell	`VALUE`	Contents is numerical
`TEXT`	Text contents	`FORMULA`	Contents is a formula

Writing in a cell

Replacing existing text	`ACell.String  = "Hello!"`
Replacing existing value	`ACell.Value   = 1,234`
Replacing existing formula	`ACell.Formula = "=AND(A1="YES";A2="OK")"`
Replacing existing formula (FR)	`ACell.FormulaLocal = "=ET(A1="YES";A2="OK")"`

Ranges

Range = a set of cells (one cell is a range): `Dim MyRange As Object`

Getting Ranges

`ASheet` is asheet object. You get the `ARange` range object:

Ny its coordinates	`ARange = ASheet.getCellRangeByName("C2:G14")`
By its name	`ARange = ASheet.getCellRangeByName("NomDePlage")`
By its coordinates (X1, Y1, X2, Y2)	`ARange = ASheet.getCellRangeByPosition(2, 1, 6, 13)`
Arbitrament (ex 3rd sheet)	`ARange = ThisComponent.Sheets.getCellRangeByPosition(2, 2, 1, 6, 13)`

Getting the active range

Like for the active cell, but verify the `"com.sun.star.sheet.SheetCellRange"` or `"[...].SheetCellRanges"` services.

Selecting a range

`ThisComponent.CurrentController.select(ARange)` where `ARange` is an object.

Range Coordinates

Coordinates (Object)	`Coord  = ARange.RangeAddress`
Sheet index (Integer)	`Rang   = ARange.RangeAddress.Sheet`
Column index (Long) top/left	`NumCTL = ARange.RangeAddress.StartColumn`
Row index (Long) top/left	`NumLTL = ARange.RangeAddress.StartRow`
Column index (Long) bottom/right	`NumCBR = ARange.RangeAddress.EndColumn`
Row index (Long) bottom/right	`NumLBR = ARange.RangeAddress.EndRow`
Containing sheet object	`ASheet = ARange.Spreadsheet`
Absolute coordinates (String)	`Coord  = ARange.AbsoluteName`

Named ranges

The `Doc` object points to the document. Also `Ranges As Object`

Named ranges	`Ranges = Doc.NamedRanges`
Ranges count (Long)	`Count  = Ranges.Count`
Getting a range (by index)	`ARange = Ranges(index)`
Checking existence (by name)	`Existe = Ranges.hasByName(Name)`
Getting a Range (by name)	`ARange = Ranges.getByName(Name)`
Adding a range	`Ranges.addNewByName(Name, Coord, _ CellRef.CellAddress, 0)`
`Coord` : Range coordinates `CellRef` : reference cell object.	
Removing (by name)	`Ranges.removeByName(Name)`

Clearing a range

Clearing `ARange` contents	`ARange.clearContents(DelMode)`

`DelMode` is a value that specifies the deletion type. `com.sun.star.sheet.CellFlags.XXX` constants give that choice (combine with `+`) :

`ANNOTATION`	Comments	`STRING`	Text
`DATETIME`	Number formatted as date-time	`VALUE`	Numbers (except date-time)
`FORMULA`	Formulas		

Getting a range cells Contents

`ARange.DataArray` est un tableau des valeurs des cellules pour `ARange`.

Copying a range contents to another range

Two ranges `Source` and `Target`, **of the same dimensions**.
Copying the contents (**values**) of `Source` into `Target`. `Target.DataArray = Source.DataArray`

Writing values into a range

`ARange` is a range object, `Table` is a table of the same dimensions, which values have to be copied into the range.

```
Dim Table As Variant
Table = ARange.DataArray 'Table has the same dimensions as the range
'(ste the array items values)
ARange.DataArray = Table
```

☞ `.DataArray` is a nested array: use `.DataArray(i)(j)`

browsing a range cells

From a collection (`Ranges.Cells`), we create an `enumeration`. It is then browsed by calling its `hasMoreElements` et `NextElement` properties:

```
Dim Ranges As Object
Ranges =
    ThisComponent.createInstance("com.sun.star.sheet.SheetCellRanges")
Ranges.insertByName("some name", ARange)
TheEnum = Ranges.Cells.CreateEnumeration
Do While TheEnum.hasMoreElements
  TheCell = TheEnum.NextElement
  'do smthg with the cell object
Loop
```

☠ Empty cells are not browsed!

Ranges misc.

Merging `ARange` object cells	`ARange.Merge`

Ranges types

The range access mode defines which service it implements:

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`

☠ The implemented service implies how the ranges should be used.
Check using their `supportsService()` method (ex. below)

Range or cell?

To know an object type, test `supportsService()` on a Range or Cell:
`If MyObj.supportsService(Service_Name) Then …`
Replace `Service_Name` with:

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

☠ Always check for a Cell type **before** a Simple Range as a cell is **also** a Simple Range!

Rows/Columns

Rows and columns are properties of `Sheet` and `Range` objects.

Overview

Rows (oRows object)	`oRows = ARange.Rows`
Columns (oCols object)	`oCols = ARange.Columns`
Number	`NumL = ARange.Rows.Count`
	`NumC = ARange.Columns.Count`
A Row (oRow object) (base 0)	`oRow = ARange.Rows(index)`
A Column (oCol object) (base 0)	`oCol = ARange.Columns(index)`

Rows/Columns Properties

Apply to Row or Rows objects (resp. Column or Columns).

Visible/Hidden (Boolean)	`IsVisible = True`
Optimal width or not (Boolean)	`OptimalWidth = True`

Inserting/Deleting rows/Columns

Given the objects `RorC`, `FirstPos` and `LastPos` the starting and ending positions for the rows set (resp. columns) to insert or delete (Long).

Inserting	`RorC.insertByIndex(FirstPos, LastPos)`
Erasing	`RorC.clearContents(EraseMode)` [EraseMode: see **Clearing a range**]
Deleting	`RorC.removeByIndex(FirstPos, LastPos)`

Fixing rows or columns

☞ Only on a visible spreadsheet.

Use the controller object: `oContrlr = ThisComponent.CurrentController`

Is there any?	`Fixed = oContrlr.hasFrozenPanes`
Fix	`oContrlr.freezeAtPosition(1, 2)`
Delete	`oContrlr.freezeAtPosition(0, 0)`

Calling a Calc function

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

Overview

```
Dim FCalc As Object
Dim Result As (according to context)
Dim Params As (according to context)
Dim FuncName As String
FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
Result = FCalc.callFunction(FuncName, Params)
```

☞ The name, arguments and result type depend upon the target function.
☠ The function name **must** be its English name.
To get it in a non-EN environment, temporarily swap the function names in Calc using **Tools > Options > LibreOffice Calc > Formula, Use English function names.**

Example 1 (`SUM()`)

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

Example 2 (`ROUND()`)

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

Creating a Calc function

Creation

Example : trapezoid surface calculation (`S = ((B + b) / 2)  × H`)

```
Function TrapezoidSurface(LB As Double, SB As Double, H As Double) As
    Double
  SurfaceTrapeze = ((LB + SB) / 2) * H
End Function
```

Using in Calc spreadsheets

Given `A2` is the large base, `A3` the small one and `A4` the height, a trapezoid surface is obtained by inserting the following formula into a cell: `=TRAPEZOIDSURFACE(A2;A3;A4)`

☞ The macro receives arguments as **values**, not the source cell objects.
☞ The macro **returns a value**. It **can't** act upon a cell object.
☞ The function must be placed within a library that is available at work time (ex : `Standard` in the document or in the user's container) (otherwise: `#VALUE!` error).

Credits

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

We are like dwarves perched on the shoulders of giants, and thus 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.])

History

Version	Date	Comments
1.15	12/14/2019	Minor updates