LibreOffice RefCard

LibreOffice Basic
Calc

v. 2.0 – 06/04/2021

Beginner

Written using LibreOffice v. 7.0.5 – Platform: All

LibreOffice documents

Current document

```
Dim oDoc As Object
oDoc = ThisComponent
```

Opening an existing document

In visible mode

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

In hidden mode

```
Dim oDoc 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
oDoc = StarDesktop.loadComponentFromURL(DocPath, "_blank", 0, Props())
```

Make visible a hidden document

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

Creating a new calc document

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

```
Dim oDoc As Object, Templte As String
Dim Props()   'here, an unitialized table
Template = "private:factory/scalc"        '(1)
'or
Templte = "C:\Path\To\SomeTemplate.ots"  '(2)
oDoc = StarDesktop.loadComponentFromURL(Templte, "_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 **File > Save As**)

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

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

Saving a copy

As above, but calling `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?

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

```
IsCalc = oDoc.SupportsService("com.sun.star.sheet.SpreadsheetDocument")
```

Calc – General

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

Automatic calculation

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

Protecting the spreadsheet

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

Sheets

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

Getting sheets

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

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

Modifiying sheets

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

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

Managing sheets

`oSheet` is a sheet object.

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

Linking a sheet

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

Finding the last used row/col

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

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

Cells

Below, `oCell` is a cell object.

Getting cells

`oSheet` is a sheet object. You can get the cell object:

Through its R/C coordinates	`oCell = oSheet.getCellRangeByName("A4")`
Through its name	`oCell = oSheet.getCellRangeByName("VAT")`
Through its XY coordinates	`oCell = oSheet.getCellByPosition(0,3)` `'with X=0 (col.A) ; Y=3 (row 4)`

Knowing the active cell

oDoc is a document object and `oCell` is the active cell we're looking for.

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

Selecting a cell

```
ThisComponent.CurrentController.select(oCell)
```

Cell coordinates

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

(Un)Protecting cells

The `oCell.CellProtection` object can get the boolean properties:

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

Getting a cell contents

Properties

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

Content types (`Type` property)

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

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

Writing in a cell

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

Ranges

oRange = a set of cells (one single cell is a range, too): `Dim MyRange As Object`

Getting Ranges

`oSheet` is a sheet object. You get the oRange range object:

By its coordinates	`oRange = oSheet.getCellRangeByName("C2:G14")`
By its name	`oRange = oSheet.getCellRangeByName("RangeName")`
By its coordinates (X1, Y1, X2, Y2)	`oRange = oSheet.getCellRangeByPosition(2, 1, 6, 13)`
Arbitrament (ex 3rd sheet)	`oRange = 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(oRange)` where oRange is an object.

Range coordinates

Row/col indices are `Longs`. Sheets indices are `Integers`.

Coordinates (Object)	`oCoord = oRange.RangeAddress`
Sheet index (Integer)	`Rang = oRange.RangeAddress.Sheet`
Column index (top/left)	`NumCTL = oRange.RangeAddress.StartColumn`
Row index (top/left)	`NumLTL = oRange.RangeAddress.StartRow`
Column index (bottom/right)	`NumCBR = oRange.RangeAddress.EndColumn`
Row index (bottom/right)	`NumLBR = oRange.RangeAddress.EndRow`
Sheet container object	`oSheet = oRange.Spreadsheet`
Absolute coordinates (String)	`Coord = oRange.AbsoluteName`

Named ranges

The oDoc object points to the document. Also `oRanges As Object`

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

Clearing a range

Clearing oRange contents	`oRange.clearContents(DelMode)`

`DelMode` specifies the deletion type. `com.sun.star.sheet.CellFlags.XXX` integer 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

`oRange.DataArray` is the oRange cell values array.

Copying a range contents to another range

Two ranges `Source` and `Target`, **of the same dimensions**.

Copying the contents (**values**) of `Source` into `Target`. `oTarget.DataArray = oSource.DataArray`

Writing values into a range

oRange 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 = oRange.DataArray 'Table has the same dimensions as the range
'(set the array items values)
oRange.DataArray = Table
```

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

browsing a range cells

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

```
Dim oRanges As Object, oEnum As Object
oRanges = oDoc.createInstance("com.sun.star.sheet.SheetCellRanges")
oRanges.insertByName("some name", oRange)
oEnum = oRanges.Cells.CreateEnumeration
Do While oEnum.hasMoreElements
  oTheCell = oEnum.NextElement
  'do smthg with the cell object
Loop
```

Empty cells are not browsed!

Ranges misc.

Merging oRange object cells	`oRange.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 = oRange.Rows`
Columns (oCols object)	`oCols = oRange.Columns`
Number	`NumRows = oRange.Rows.Count` `NumCols = oRange.Columns.Count`
A Row (oRow object) (base 0)	`oRow = oRange.Rows(index)`
A Column (oCol object) (base 0)	`oCol = oRange.Columns(index)`

Rows/Columns properties

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

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

Inserting/Deleting rows or 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")
Result = 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
  TrapezoidSurface = ((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
2.0	04/06/2021	Formatting revision; corrections.