

LibreOffice®

LibreOffice RefCard

LibreOffice BASIC

Dialogs

v. 1.05 – 12/02/2018

Written using LibreOffice v. 5.3.3 – Platform : All

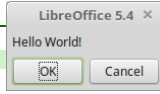
Advanced

Dialogs In BASIC

Displaying A Simple Message

Print "Hello World!"

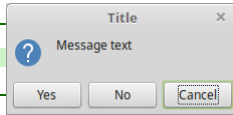
The Cancel choice stops the program.



Displaying Information

MsgBox(MessageText[, Dialog Code[, Title]])

Line breaks in MessageText with Chr(10) or Chr(13).



Display A Message And Wait For A Response

Response = MsgBox(MessageText[, DialogCode[, Title]])

where

- Response is an integer value that reflects the user's choice.
- DialogCode : the sum of button codes + icon + default button (as below).

Buttons to display			
0	OK	3	Yes, No, Cancel
1	OK, Cancel	4	Yes, No
2	Stop, Retry, Ignore	5	Retry, Cancel

Icon	
0	(none)
16	⚠ Critical message
32	❓ Question
48	⚠ Caution
64	i Information (OK only)

Default button			
0	First	256	Second
512	Last		

Return values (user's selection)			
1	OK	3	Stop
2	Cancel	4	Retry
5	Ignore	6	Yes
7	No		

InputDialog() Function

Function InputBox(Message[, Title[, DefaultValue]])
returns a string. On cancellation, returns a zero-length string.

API Dialogs

The aspect of FilePicker and FolderPicker types below depend upon Tools > Options > LibreOffice > General, Use LibreOffice dialogs

API Dialog Types

File Selection: FilePicker Objects

com.sun.star.ui.dialogs.FilePicker From above configuration option.
com.sun.star.ui.dialogs.OfficeFilePicker Forces LibreOffice style.
com.sun.star.ui.dialogs.SystemFilePicker Forces native OS style.

Directory Selection: FolderPicker Objects

com.sun.star.ui.dialogs.FolderPicker From above configuration option.
com.sun.star.ui.dialogs.OfficeFolderPicker Forces LibreOffice style.
com.sun.star.ui.dialogs.SystemFolderPicker Forces native OS style.

The FilePicker Object (or OfficeFilePicker Or SystemFilePicker)

oFilePicker = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
AppendFilter() By pairs: appendFilter("LiteralName", "*.xyz")
Ex: oFilePicker.appendFilter("ODF Documents", "*.odt;*.ods")

CurrentFilter Sets the default filter from the ones added using AppendFilter (literal name) or the user's filter selection.
DefaultName Default name for the file to save.
DisplayDirectory The starting directory or the user's directory selection.
Execute Transfers the execution stream to the dialog and reads the return code (see return constants values below).
Files An array of selected files.
initialize() Dialog type selection (see type constants below).
Dim FPType(0) As Integer
FPType(0) = 'a type constant
oFilePicker.initialize(FPType())
MultiSelectionMode Disables/Enables the multi-selection mode (defaults to False).
Title The dialog window title.

FilePicker Type Constants

com.sun.star.ui.dialogs.TemplateDescription.XXX :

FILEOPEN_SIMPLE	0	Simple open file dialog.
FILESAVE_SIMPLE	1	Simple save file dialog.
FILESAVE_AUTOEXTENSION_PASSWORD	2	Enhanced save dialog: automatic extension + password.
FILESAVE_AUTOEXTENSION_PASSWORD_FILTEROPTIONS	3	Enhanced save dialog: automatic extension + password + filter options.
FILESAVE_AUTOEXTENSION_SELECTION	4	Enhanced save dialog: automatic extension + selection.
FILESAVE_AUTOEXTENSION_TEMPLATE	5	Enhanced save dialog: automatic extension + templates.
FILEOPEN_LINK_PREVIEW_IMAGE_TEMPLATE	6	Enhanced open dialog: insert as link + preview + template.
FILEOPEN_PLAY	7	Enhanced open dialog: play.
FILEOPEN_READONLY_VERSION	8	Enhanced open dialog: read-only + version.
FILEOPEN_LINK_PREVIEW	9	Enhanced open dialog: link + preview.
FILESAVE_AUTOEXTENSION	10	Enhanced save dialog: automatic extension
FILEOPEN_PREVIEW	11	Enhanced open dialog: preview.
FILEOPEN_LINK_PLAY	12	Enhanced open dialog: insert as link + play.

Return Values Constants

com.sun.star.ui.dialogs.ExecutableDialogResults.XXX
CANCEL 0 Canceled OK 1 Validated

The FolderPicker Object (Or OfficeFolderPicker Or SystemFolderPicker)

oFldrPicker = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
Description Help text to display on the dialog. Does nothing on an OfficeFolderPicker.
DisplayDirectory Starting directory.
Execute Transfers the execution stream to the dialog and reads the return code (see return code constants above).
Title Dialog title.
Directory User's selection.

Opening A Unique File (FilePicker)

- Create a FilePicker. The default type usually fits (FILEOPEN_SIMPLE),
- set its properties and methods (see above),
- execute,
- read the return values in theCurrentFilter, DisplayDirectory and Files (vector) properties (Files(0) only has a value).

```
Dim oFilePicker As Object, FileName As String
FileName = ""
'FilePicker initialization
oFilePicker = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
oFilePicker.DisplayDirectory = ConvertToURL("C:\Path\To\SomeDir")
oFilePicker.appendFilter("Calc Documents", "*.ods")
oFilePicker.CurrentFilter = "Calc Documents"
oFilePicker.Title = "Select a Calc document"
'execution and return check (OK?)
If oFilePicker.execute =
    com.sun.star.ui.dialogs.ExecutableDialogResults.OK Then
    FileName = oFilePicker.Files(0)
End If
```

Opening Several Files (FilePicker)

- As above,
- set its properties and methods (esp. with MultiSelectionMode = True),
- execute,
- read the Files() vector that holds the user's choices.

Saving A File (FilePicker)

- Create a FilePicker,
- set its properties and methods (type FILESAVE_XXX) (see above),
- execute,
- read the return values in theCurrentFilter, DisplayDirectory and Files (vector) properties (Files(0) alone holds a value).

Selecting A Directory (FolderPicker)

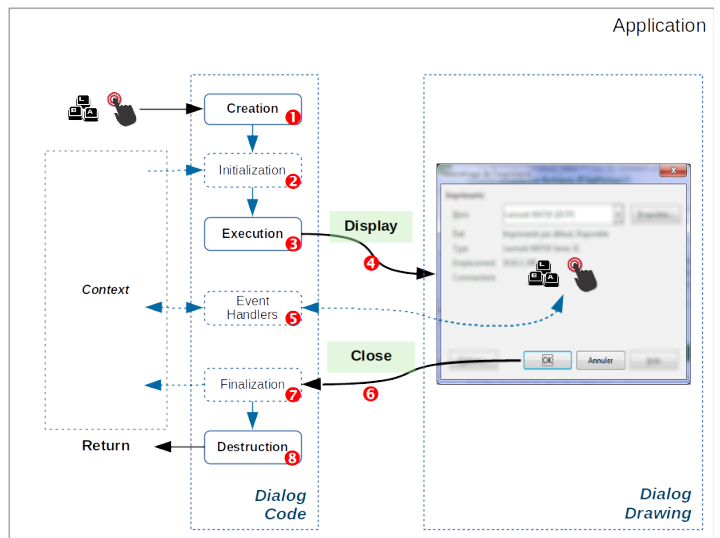
- Create a FolderPicker,
- set its properties and methods (see above),
- execute,
- read the return value in Directory.

```
Dim oFP As Object, DirName As String
DirName = ""
oFP = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
oFP.DisplayDirectory = ConvertToURL("C:\Path\To\SomeDir")
oFP.Description = "Select a directory"
oFP.Title = "Select the backup directory"
If oFP.execute =
    com.sun.star.ui.dialogs.ExecutableDialogResults.OK Then
    DirName = oFP.Directory
End If
```

Custom Dialogs 101

A BASIC dialog = a dialog module (drawing) + (at least) one code module.

Dialog Execution Sequence



The figure above illustrates a typical dialog execution sequence:

- As a response to an application event, you create the dialog,
- (initialize dialog controls from the application context if necessary),
- run the dialog that receives the execution flow:
 - display,
 - (dialog controls events management),
 - some events imply the dialog close (OK, Cancel) ;
- (finalize to the application context if necessary),
- the dialog is destroyed and the flow returns to the calling application.

- ☞ Creation, initialization, execution, finalization and destruction: processed in your code. Display, closing: automatic operations that follow the latter.
- ☞ Think to the responses to control **events** ("Associating an event to a macro" and RefCard #4).

Loading Dialog Libraries

- ☞ Much code or several dialogues? You may want to store them in dedicated libraries.
- ☞ Dialog libraries are **never** automatically loaded.
- ☞ Loading libraries: beware to the **typcase!**

Modal Vs Non-modal

- Modal** A modal dialog takes full control upon the keyboard, mouse and screen, waiting for some action from the user. The underlying application is then not accessible.
- ☞ By default, dialogues are modal.
- Non-modal** A non-modal dialog doesn't block access to the application.
- Ex : the LibreOffice **Search & replace** dialogue.
 - ☞ Multiple calls to a non-modal dialog may block the application.

Standard Custom Dialogs (modal)

This is the most frequent use.
Given a dialog module MyDlg and a code module MyDlgCode in a MyDlgLib library. In a code module Sub, we instantiate a dialog object (oDlg) from the dialog.

Creating / Loading In Memory

```
DialogLibraries.loadLibrary("MyDlgLib")
oLib = DialogLibraries.getByNamed("MyDlgLib")
oModule = oLib.getByNamed("MyDlg")
oDlg = CreateUnoDialog(oModule)
'on now manipulate the oDlg object
```

Calling The Dialog

oDlg.execute ☞ The execution flow is transferred to the dialog.

Calling And Testing The Return Value

```
If oDlg.execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
Then ...
```

☞ The execution flow is transferred to the dialog and the return value is checked (did the user select OK?).

Terminating / Destroying The Dialog

```
oDlg.dispose
```

Wrap-up Example (Code Module)

This example doesn't show any event management.

```
Sub ShowDialog()
Dim oLib As Object, oModule As Object, oDlg As Object

DialogLibraries.loadLibrary("MyDlgLib")
oLib = DialogLibraries.getByNamed("MyDlgLib")
oModule = oLib.getByNamed("MyDlg")
oDlg = CreateUnoDialog(oModule)
'initializeDlg() 'code to initialize the dialog contents
If oDlg.execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
Then
'finalizeDlg() 'code to do something with the user's input
End If
oDlg.dispose
End Sub
```

Non-modal Custom Dialogs

Given a dialog module MyNMDlg and a code module MyNMDlgCode in MyNMDlgLib library. In a Sub of the code module, we instantiate an object (oDlg) from the dialog. Apply the same technique as above, with some subtleties:

- The dialog **display** is ensured using oDlg.SetVisible(True) instead of oDlg.execute,
- we set two global Boolean flags:
 - gRunning that prevents multiple executions,
 - gShowMe that controls the dialog display,
- events responses** (controls) must set gShowMe to False to close the dialog.

Displaying The Dialog

oDlg.SetVisible(True) ☞ The dialog is **displayed**. The execution flow is **not** transferred to the dialog.

Wrap-up Example (Code Module)

```
Dim gShowMe As Boolean 'dialog display flag.
Dim gRunning As Boolean 'execution flag to prevent multiple runs.

Sub ShowNonModalDialog()
'manages the dialog creation and display

Dim oLib As Object, oModule As Object, oDlg As Object

'check for multiple runs
If Not gRunning Then
gRunning = True
gShowMe = True
DialogLibraries.loadLibrary("MyNMDlgLib")
oLib = DialogLibraries.getByNamed("MyNMDlgLib")
oModule = oLib.getByNamed("MyNMDlg")
oDlg = CreateUnoDialog(oModule)
'initializeDlg() 'code to initialize the dialog contents

'display the dialog as long as the flag is True
Do While gShowMe
Wait 20 'allow other software execution
oDlg.SetVisible(True) 'keep on screen
Loop
'if we are here, the dialog was closed (see OnBtnOKClick)
'finalizeDlg() 'code to do something with the user's input
oDlg.dispose
gRunning = False
End If
End Sub ' ShowNonModalDialog
```

```
Sub OnBtnOKClick(ByRef pEvt As Object)
'Response to a click on a OK button on the non-modal dialog

'set the appropriate actions
'then end with:
gShowMe = False '=> the ShowNonModalDialog while loop ends
'thus the dialog closes
End Sub 'OnBtnOKClick
```

Associating An Event To A Macro

A dialog communicates with the application through **events** (⚡ on the figure). You thus have to write macros to respond to events occurrences. Extract from RefCard #4:

- Create the macro** to call, according to this template:

```
Sub MacroName()
End Sub
```

☞ Hint: name the macro from the object and event type.

```
Example : Sub OnOKButtonClick()
```

That Sub may get a parameter. See below "Getting Information",

- select the object** that carries the event to intercept,
- go to its settings (differs according to the object),
- select the event** to intercept,
- point to the macro** that should be run when the event fires (point 1).

☞ More information about events in RefCard #4.

Getting Information About The Triggered Event

The event management macro can read the input parameter to get more information about the event itself:

```
Sub EventResponse(ByRef Event As Object)
End Sub
```

The Event input object properties and methods depend from the type of event that triggered the macro call.

Most Frequent Cases For Controls

To gain access to the calling...

Calling **control** object

Control **model** object

Dialog object that owns the control

Interrogate

Event.Source

Event.Source.Model

Event.Source.Context

Initialization And Finalization

Initialization

(⚡ in the figure) A dialog often requires information from the execution context. The initialization macro configures the dialogue contents from this data.

Finalization

(⚡ in the figure) Here, we have the opposite process: setting context data from what was input in the dialogue.

Managing Dialog Modules

LibreOffice manages dialog modules independently from code (see RefCard #1). We may copy such modules from a document to another.

Copying Modules From A Library To Another

(within the same document or between documents/containers)

- In the IDE, open both source and target documents/containers,
- open the **Macro organizer** (🔍 button),
- go to the **Dialogs** tab, drag/drop from the source to the target.

☞ By default, modules are **moved**. To **copy**: **Ctrl** + drag/drop.

Saving A Dialog (Drawing Alone)

- In the IDE, open the dialog module to save,
- click the toolbar button **Export Dialog**,
- name the file and save it.

The document is in XML format with an .xdl extension.

☞ Import is the reciprocal process, using the **Import Dialog** button.

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 of Chartres [attr.])

History

Version	Date	Comments
1.05	02/12/2018	Minor updates.

License

This RefCard is distributed under the **Creative Commons BY-SA v3 license** Information

<https://creativecommons.org/licenses/by-sa/3.0/fr/>

