LibreOffice RefCard

LibreOffice BASIC
Runtime Parameters

Advanced

v. 2.11 – 31/12/2023

Written using LibreOffice v. 7.4.0 – Platform : All

Knowing LibreOffice paths

User file paths

These paths can be modified in the UI menu **Tools > Options > LibreOffice > Paths**
In code, use the PathSettings service:

```
Dim oPaths As Object
Dim Dirs As Variant         'directories array
oPaths = CreateUnoService("com.sun.star.util.PathSettings")
Dirs = Split(oPaths.Xxx, ";")
```

where Xxx is the property associated to the wanted directory, among:

The property...	... points to
`Addin`	The directory with the old add-ins.
`AutoCorrect`	The autocorrection dialog parameters.
`AutoText`	The autotexts storage directory name.
`Backup`	The backup directory name.
`Basic`	Here are the BASIC files used for autopilots.
`Bitmap`	Toolbar icons.
`Config`	Configuration files.
`Dictionary`	Storage for the provided dictionaries.
`Favorite`	Path for storing files bookmarks.
`Filter`	Filters storage.
`Gallery`	Multimedia and Gallery storage.
`Graphic`	The directory displayed when a graphic is opened or saved.
`Help`	Path to help files.
`Linguistic`	Language checking files storage.
`Module`	Paths to the modules.
`Palette`	Paths to the palettes (`.sob` to `.sof`).
`Plugin`	Plugins storage paths.
`Storage`	Mail or newsgroup files storage (ex : FTP server).
`Temp`	Base URL to temporary files.
`Template`	Template storage directories.
`UIConfig`	Global directories for configuration files storage.
`UserConfig`	The user's configuration parameters directory.
`Work`	Path to the work directory. Can be modified to follow the user's needs.
`BasePathShareLayer`	Shared folder between users.
`BasePathUserLayer`	User's folder.

Extension installation path

Use the "package provider" singleton:
`"/singletons/com.sun.star.deployment.PackageInformationProvider"`

```
Dim oInfo As Object, Path As String
oInfo =
    GetDefaultContext.getByName("/singletons/com.sun.star.deployment
    .PackageInformationProvider")
If Not IsNull(oInfo) Then
  Path = oInfo.getPackageLocation(ExtID)
End If
If (Path <> "") Then Path = Path & "/"
```

where ExtID is the unique identifier for the given extension (ex : `"com.company.AName"`)

☞ Path either holds the directory (in URL form) or a zero-length string if not found.
☞ You may also use the strings expander with the UNO_USER_PACKAGES_CACHE macro.

Knowing LibreOffice execution parameters

Two complementary services are available: `PathSubstitution` and `MacroExpander`.

Using the `PathSubstitution` service

```
Dim oSubst As Object, Result As String
oSubst = CreateUnoService("com.sun.star.util.PathSubstitution")
Result = oSubst.getSubstituteVariableValue("$(var_name)")
```

☞ The substituted variable is a string which format is `$(var_name)`
☞ The result is in URL form.

The variable...	... is a substitute for
`$(inst)`	The LibreOffice installation path.
`$(prog)`	The soffice program path.
`$(user)`	The user's installation path.
`$(work)`	The user's work directory. Under Windows, it is `My Documents`. Under Unix, it is the "`home`" dir.
`$(home)`	The user's directory. Under Unix, it is the "`home`" dir. Under Windows, it is the directory that `CSIDL_PERSONAL` point to, for ex : `"Documents and Settings\<username>\Documents"`.
`$(temp)`	The current temporary directory.
`$(path)`	The contents of the environment PATH variable.
`$(username)` (since LibO 5.2)	The user name for the current session (without domain name under Windows).
`$(langid)`	Then language code LibreOffice uses. Ex : `1036` for French (France).
`$(vlang)`	The language code LibreOffice uses, in text form. Ex : "fr" for French.

Using the "Macro" (string) expander singleton

☠ Such "macros" have *nothing* to see with the BASIC macros we're dealing with in other parts of this refcard.

`"/singletons/com.sun.star.util.theMacroExpander"`
and call its `ExpandMacros()` method:

```
Dim oContext as Object              'context object
Dim oMacroExpand as Object          'macro expander
Dim Result As String

oContext = getProcessServiceManager().DefaultContext
oMacroExpand =
    oContext.getValueByName("/singletons/com.sun.star.util.theMacroE
    xpander")

Result = oMacroExpand.ExpandMacros("$UNO_USER_PACKAGES_CACHE")
```

☠ A "macro" string must start with the $ symbol.
☞ Directory and file names are returned in URL form.

"Macros"

There're plenty of them. For a full list, see:
https://wiki.documentfoundation.org/Development/Environment_variables
and, for the bootstrap file:

`ORIGIN`	LibreOffice installation directory.
`SYSUSERCONFIG`	The user's parameters directory (profile) in his session.
`UNO_USER_PACKAGES_CACHE`	The extensions directory.
`USERNAME`	The user's account name.

Knowing LibreOffice command-line parameters

Here, we only describe the parameters that are useful in macro development mode. A full list may be found here (checked 2023/12):
https://dnimruoynepo.blogspot.fr/2016/12/command-line-arguments-in-libreoffice.html
From which these are extracted:

Help and information

`--version`	Displays the version number.
`--nstemporarydirectory`	(only for MacOS X sandbox) Returns the temporary directory path for the current user. Overrides all other arguments.

General parameters

`--quickstart[=no]`	Disables/Enables the quick starter. Only one value at the right of "=":no which disables the quick start.
`--nolockcheck`	Disables check for remote instances using one installation.
`--infilter={filter}`	Force an input filter type if possible. If it isn't possible, LibreOffice uses the available filter for the document. Example : `--infilter="Calc Office Open XML"` `--infilter="Text (encoded):UTF8,LF,,,"` Note that filter names may change, these examples show the use of the argument. Unfortunately, there is no easy way to know all the available filters.
`--pidfile={file}`	Store `soffice.bin` pid in `{file}`.
`--display {display}`	Sets the `DISPLAY` environment variable on UNIX-like platforms to the value `{display}` (only supported by a start script for the LibreOffice).

UI control

`--nologo`	Disables the splash screen at program start.
`--minimized`	Starts minimized. The splash screen is not displayed.
`--nodefault`	Starts without displaying anything except the splash screen (do not display initial window).
`--invisible`	Starts in invisible mode. Neither the start-up logo nor the initial program window will be visible. LibreOffice can be controlled, and documents and dialogs can be controlled and opened via the API. Using the parameter, LibreOffice can only be ended using the taskmanager (Windows) or the `kill` command (UNIX-like systems). `--invisible` cannot be used with `--quickstart`.
`--headless`	Starts in "headless mode" which allows using the application without GUI. This special mode can be used when the application is controlled by external clients via the API.

☞ `invisible` vs `headless`.
`--invisible` does **not** disable the GUI: documents and dialogs are displayed.
`--headless` calls a "silent mode" everytime a GUI is not needed.

`--norestore`	Disables restart and file recovery after a system crash.
`--safe-mode`	Starts in a safe mode, i.e. starts temporarily with a fresh user profile and helps to restore a broken configuration.
`--accept={UNO-URL}`	Specifies an `UNO-URL` connect-string to create an UNO acceptor through which other programs can connect to access the API. `{UNO-URL}` is a string like `uno:connection-type,params;protocol-name,params;ObjectName`. At the same time, according to the LibreOffice code, the `ObjectName` is ignored.
`--unaccept={UNO-URL}`	Closes an acceptor that was created with `--accept`. Use `--unaccept=all` to close all open acceptors.

Developer parameters

`--terminate_after_init`	Exit after initialization complete (no documents loaded).
`--eventtesting`	Exit after loading documents.

Creating documents

These arguments create an empty document of the specified kind. Only one of them may be used in one command line. If filenames are specified after an argument, then it tries to open those files in the specified component. If it is impossible to open in the selected component, LibreOffice loads the document.
The options below create empty documents of the specified type:

`--writer`	`--draw`	`--base`	`--math`
`--calc`	`--impress`	`--global`	`--web`

Opening files

The arguments define how following filenames are treated. New treatment begins after the argument and ends at the next argument. The default treatment is to open documents for editing, and create new documents from document templates.

`-n`	Treats following files as templates for creation of new documents.
`-o`	Opens following files for editing, regardless whether they are templates or not.
`--pt {Printer}`	Prints following files to the printer `{Printername}`, after which those files are closed. The splash screen does not appear. If used multiple times, only last `{Printername}` is effective for all documents of all `--pt` runs. Also, `--printer-name` argument of `--print-to-file` switch interferes with `{Printername}`.
`-p`	Prints following files to the default printer, after which those files are closed. The splash screen does not appear. If the file name contains spaces, then it must be enclosed in quotation marks.
`--view`	Opens following files in viewer mode (read-only).
`--show`	Opens and starts the following presentation documents of each immediately. Files are closed after the showing. Files other than Impress documents are opened in default mode, regardless of previous mode.

`--convert-to OutExt[:OutFilterName] [--outdir output_dir]`	Batch convert files (implies `--headless`).
	`OutExt` : target extension.
	`OutFilterName` : the conversion filter.
	If `--outdir` isn't specified, then current working directory is used as `output_dir`.
	If `--convert-to` is used more than once, last value of `OutputFileExtension[:OutputFilterName]` is effective.
	If `--outdir` is used more than once, only its last value is effective.
	Examples :
	`-- convert-to pdf *.doc`
	`-- convert-to pdf:writer_pdf_Export --outdir /home/user *.doc`
	`-- convert-to "html:XHTML Writer File:UTF8" *.doc`
	`-- convert-to "txt:Text (encoded):UTF8" *.doc`
	Unfortunately, now there is no easy way to know all the possible filter values. Thus, the use of this argument is difficult, in spite of its potential usefulness.
	☞ Check also online help, at: File Conversion Filter Names.

PDF export options (v.7.4+)

Syntax:

```
--convert-to 'pdf:draw_pdf_Export:{option:{"type":"T","value":"V"}}'
```

☞ If several options, use commas as delimiters (see encryption ex.).

☠ Beware to quotes!

https://vmiklos.hu/blog/pdf-convert-to.html

Details for `{option}`, `"T"` and `"V"`

- Pages: `{"PageRange":{"type":"string","value":"2-"}}'`
- Watermark: `{"TiledWatermark":{"type":"string","value":"draft"}}'`
- Encryption: `{"EncryptFile":{"type":"boolean","value":"true"}, "DocumentOpenPassword":{"type":"string","value":"secret"}}'`
- PDF v.1.5 : `{"SelectPdfVersion":{"type":"long","value":"15"}}'`

https://ask.libreoffice.org/en/question/2641/convert-to-command-line-parameter/ which points to: https://cgit.freedesktop.org/libreoffice/core/tree/filter/source/config/fragments/filters (checked 2023/12)

`--print-to-file [--printer-name printer_name] [--outdir output_dir]`	Batch print files to file. If `--outdir` is not specified, then current working directory is used as `output_dir`. If `--printer-name` or `--outdir` used multiple times, only last value of each is effective. Also, `{Printername}` of `--pt` switch interferes with `--printer-name`.
`--cat`	Dump text content of the following files to console (implies `--headless`). Cannot be used with `--convert-to`.
`-env:var[=value]`	Set a bootstrap variable. For example: to set a non-default user profile path: `-env:UserInstallation=file:///tmp/test` Unfortunately, now there is no easy way to get all the possible variables for this flag.

Calling a macro through the command-line

Syntax

☞ The `--headless` option triggers a silent execution (see above).

Calling a Global macro

```
{soffice} "macro:///library/module/macro[(params)]"
```

Calling a Macro stored in an ODF document

```
{soffice} path/to/doc.odf "macro://./library/module/macro[(params)]"
```

The `{soffice}` form

Windows	`%programfiles%\libreoffice 5\program\soffice.exe`
GNU/Linux	`/opt/LibreOffice 5/program/soffice`

Installing a macro… by macro

☞ We don't install a macro, we install a **library** that contains it.

In a nutshell

A container file (Writer, Calc, etc.) holds both the macro to install and an installer macro:

- The installer macro is stored in the document `Standard` library,
- The macro to install is *separated from the installer* and stored within its own library. We'll install the latter.

☞ Generally, the container file type (Writer, Calc, etc.) is not connected to the macro to install capabilities. Writer is a good container as we may use it to document the process.

The macro to install

Store it in its own library in the container document. We'll install that library.

The installer macro

Its purpose is to copy the contained library to the global container `My Macros`. Here's an example of a typical installation process for a code library:

```
Dim oSrcLib As Object      'source library (container document)
Dim oDestLib As Object     'target library (in 'My Macros')
Dim i As Integer
Dim SrcModules() As String
'we create the target library if it doesn' exist yet
If Not GlobalScope.BasicLibraries.hasByName("MyTargetLib") Then
  GlobalScope.BasicLibraries.createLibrary("MyTargetLib")
End If
'we copy the modules
If BasicLibraries.hasByName("MySourceLib") Then
  BasicLibraries.loadLibrary("MySourceLib")
  oSrcLib  = BasicLibraries.getByName("MySourceLib")
  oDestLib = GlobalScope.BasicLibraries.getByName("MyTargetLib")
  SrcModules = oSrcLib.getElementNames()
  'installation des modules 1 par 1
  i = LBound(SrcModules())
  Do While (i <= uBound(SrcModules()))
    If Not oDestLib.hasByName(SrcModules(i)) Then
      oDestLib.insertByName(SrcModules(i), _
                     oSrcLib.getByName(SrcModules(i)))
    End If
    i = i + 1
  Loop
End If
```

☞ To install a dialog library, replace `BasicLibraries` with `DialogLibraries` and `GlobalScope.BasicLibraries` with `GlobalScope.DialogLibraries`.

Beyond macros: extensions

The next step would be to turn our macro into an extension for ease of distribution or use.

✰ This is a difficult task. Bernard Marcelly's **ExtensionCompiler** brings a precious help in that area: http://berma.pagesperso-orange.fr/Files_en/ExtensionCompiler.ott. (the Orange pagesperso website is offline since 2023/09).

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
2.0	04/06/2021	Formatting update.
2.1	02/15/2021	Added: PDF export options.
2.11	12/31/2023	Added: Conversion references.