



Getting Started Guide

Chapter 13

Getting Started with Macros

Using the Macro Recorder ... and Beyond

Copyright

This document is Copyright © 2017 by the LibreOffice Documentation Team. Contributors are listed below. You may distribute it and/or modify it under the terms of either the GNU General Public License (<http://www.gnu.org/licenses/gpl.html>), version 3 or later, or the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), version 4.0 or later.

All trademarks within this guide belong to their legitimate owners.

Contributors

Andrew Pitonyak
Ron Faile Jr.

Peter Schofield
Hazel Russman

Martin Fox
Olivier Hallot

Feedback

Please direct any comments or suggestions about this document to the Documentation Team's mailing list: documentation@global.libreoffice.org

Note: Everything you send to a mailing list, including your email address and any other personal information that is written in the message, is publicly archived and cannot be deleted.

Acknowledgments

This chapter is based on Chapter 13 of *Getting Started with OpenOffice.org 3.3*. The contributors to that chapter are:

Andrew Pitonyak

Jean Hollis Weber

Publication date and software version

Published 16 February 2017. Based on LibreOffice 5.2.

Note for Mac users

Some keystrokes and menu items are different on a Mac from those used in Windows and Linux. The table below gives some common substitutions for the instructions in this chapter. For a more detailed list, see the application Help.

Windows or Linux	Mac equivalent	Effect
Tools > Options menu selection	LibreOffice > Preferences	Access setup options
<i>Right-click</i>	Control+click and/or right-click depending on computer setup	Open a context menu
<i>Ctrl (Control)</i>	⌘ (<i>Command</i>)	Used with other keys
<i>F5</i>	<i>Shift+⌘+F5</i>	Open the Navigator
<i>F11</i>	⌘+T	Open the Styles and Formatting window

Contents

Copyright.....	2
Contributors.....	2
Feedback.....	2
Acknowledgments.....	2
Publication date and software version.....	2
Note for Mac users.....	2
Introduction.....	4
Your first macros.....	4
Adding a macro.....	4
Recording a macro.....	6
Running a macro.....	7
Viewing and editing macros.....	7
Commenting with REM.....	8
Defining subroutines with SUB.....	8
Defining variables using Dim.....	9
Explaining macro code.....	9
Creating a macro.....	10
A more complicated example of a macro.....	10
Running a macro quickly.....	13
Macro recorder failures.....	13
Dispatch framework.....	13
How the macro recorder uses the dispatch framework.....	14
Other options.....	14
Macro organization.....	14
Where are macros stored?.....	15
Importing macros.....	16
Downloading macros to import.....	17
How to run a macro.....	18
Toolbars, menu items, and keyboard shortcuts.....	18
Events.....	19
Extensions.....	20
Writing macros without the recorder.....	20
Finding more information.....	21
Included material.....	21
Online resources.....	21
Printed and eBook materials.....	21

Introduction

A macro is a saved sequence of commands or keystrokes that are stored for later use. An example of a simple macro is one that “types” your address. The LibreOffice macro language is very flexible, allowing automation of both simple and complex tasks. Macros are very useful when you have to repeat the same task in the same way over and over again.

LibreOffice macros are usually written in a language called LibreOffice Basic, sometimes abbreviated to Basic. Although you can learn Basic and write macros, there is a steep learning curve to writing macros from scratch. The usual methods for a beginner are to use macros that someone else has written or use the built-in macro recorder, which records keystrokes and saves them for use.

Most tasks in LibreOffice are accomplished by “dispatching a command” (sending a command), which is intercepted and used. The macro recorder works by recording the commands that are dispatched (see “Dispatch framework” on page 13).

Your first macros

Adding a macro

The first step in learning macro programming is to find and use existing macros. This section assumes that you have a macro that you want to use, which may be in an email, on a web page, or even in a book. For this example, the macro in Listing 1 is used. You must create a library and module to contain your macro; see “Macro organization” on page 14 for more information.

Listing 1: Simple macro that says hello

```
Sub HelloMacro
  Print "Hello"
End Sub
```

Use the following steps to create a library to contain your macro:

- 1) Use **Tools > Macros > Organize Macros > LibreOffice Basic** to open the LibreOffice Basic Macro dialog (Figure 1).
- 2) Click **Organizer** to open the Basic Macro Organizer dialog (Figure 2) and select the *Libraries* tab.
- 3) Set the *Location* to *My Macros & Dialogs*, which is the default location.
- 4) Click **New** to open the New Library dialog.
- 5) Enter a library name, for example TestLibrary, and click **OK**.
- 6) Select the *Modules* tab.
- 7) In the *Module* list, expand *My Macros* and select, for example TestLibrary. A module named Module1 already exists and can contain your macro. If you wish, you can click **New** to create another module in the library.
- 8) Select Module1, or the new module that you created, and click **Edit** to open the Integrated Development Environment (IDE) (Figure 3). The IDE is a text editor included with LibreOffice that allows you to create and edit macros.
- 9) When a new module is created, it contains a comment and an empty macro named Main, which does nothing.
- 10) Add the new macro either before Sub Main or after End Sub. Listing 2 shows the new macro has been added before Sub Main.

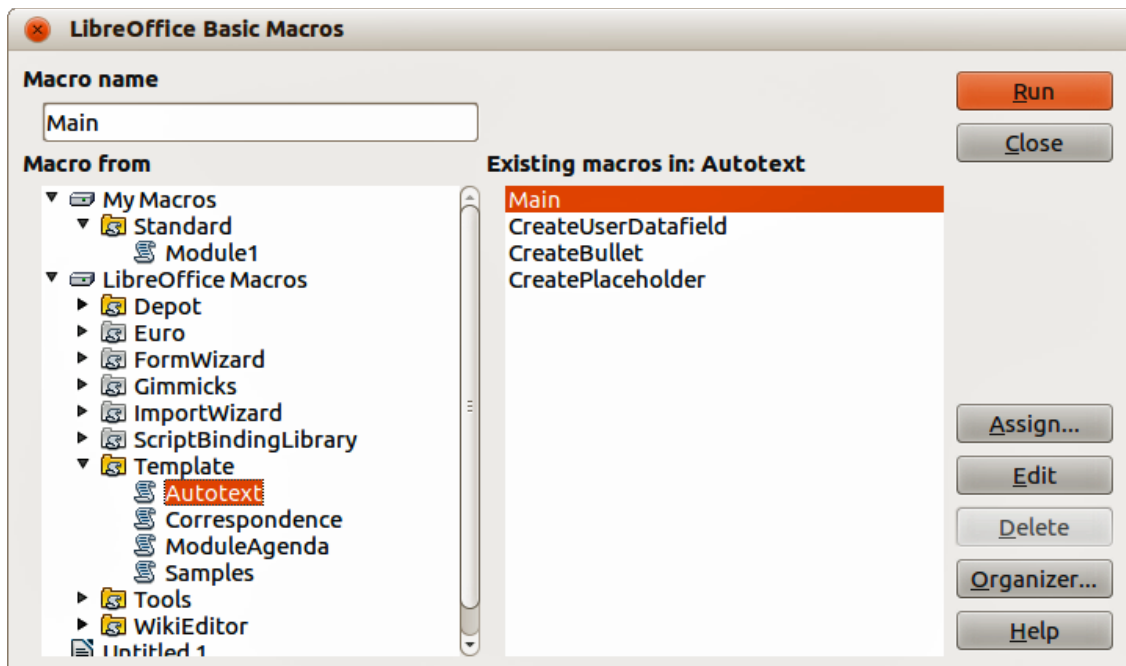


Figure 1: LibreOffice Basic Macros dialog

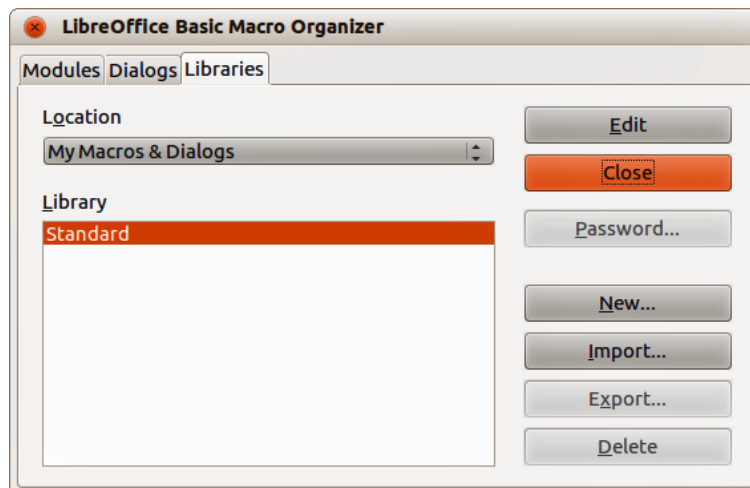





Figure 2: LibreOffice Basic Macro Organizer dialog

- 11) Click the **Compile** icon  on the Macro toolbar to compile the macro.
- 12) Place the cursor in the HelloMacro subroutine and click the **Run BASIC** icon  on the Macro toolbar, or press the *F5* key, to run the HelloMacro in the module. A small dialog will open with the word "Hello" displayed. If the cursor is not in a subroutine or function, a dialog will open; select the macro to run.
- 13) Click **OK** to close this small dialog.
- 14) To select and run any macro in the module, click the **Select Macro** icon  on the Standard toolbar or go to **Tools > Macros > Organize Macros > LibreOffice Basic**.
- 15) Select a macro and then click **Run**.

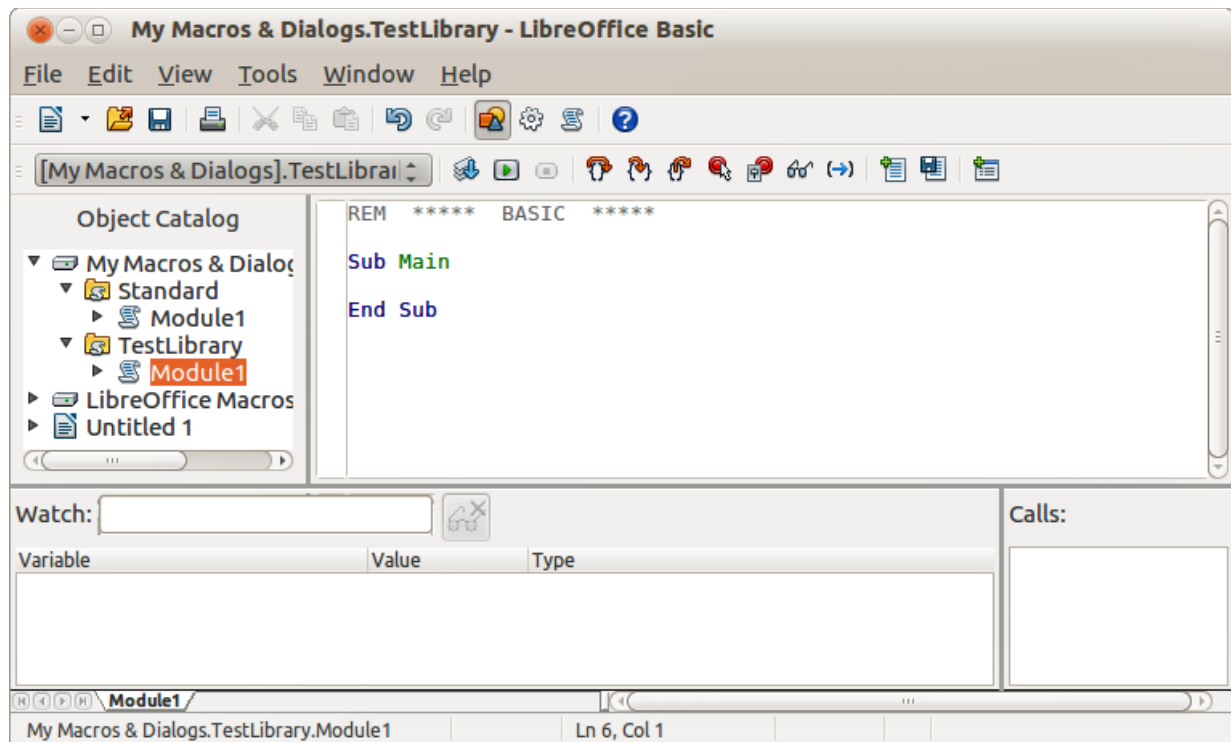


Figure 3: Integrated Development Environment dialog

Listing 2: Module1 after adding the new macro.

```
REM ***** BASIC *****

Sub HelloMacro
    Print "Hello"
End Sub

Sub Main

End Sub
```

Recording a macro

If you have to repeatedly enter the same information, you can copy this information after it has been entered into your document for the first time, then paste the information into your document each time you want to use it. However, if something else is copied to the clipboard, the contents on the clipboard are changed. This means that you have to re-copy your repeated information. To overcome this problem, you can create a macro that enters your repeated information.



Note

For some types of information that you want to repeatedly enter into a document, it may be more convenient to create an AutoText file. See the *Writer Guide Chapter 3 Working with Text* for more information.

Make sure macro recording is enabled by going to **Tools > Options > LibreOffice > Advanced** on the main menu bar and selecting the option **Enable macro recording**. By default, this feature is turned off when LibreOffice was installed on your computer.

- 1) Go to **Tools > Macros > Record Macro** on the main menu bar to start recording a macro. A small dialog is displayed indicating that LibreOffice is recording a macro.

- 2) Type the desired information or perform an appropriate series of operations. As an example, type your name.
- 3) Click **Stop Recording** on the small Recording dialog to stop recording and the LibreOffice Basic Macros dialog opens (Figure 1 on page 5).
- 4) Open the library container *My Macros*.
- 5) Find the library named *Standard* in My Macros. Note that every library container has a library named *Standard*.
- 6) Select the *Standard* library and click **New Module** to create a new module to contain the macro. This opens the New Module dialog.
- 7) Type a descriptive name for the new module, for example *Recorded*, and click **OK** to create the module. The LibreOffice Basic Macros dialog now displays the name of the new module in the *Standard* library.
- 8) In the **Macro name** text box, type a name for the macro you have just recorded, for example *EnterMyName*.
- 9) Click **Save** to save the macro and close the LibreOffice Basic Macros dialog.

If you followed all of the above steps, the *Standard* library now contains a module named *Recorded* and this module contains the *EnterMyName* macro.



Note

When LibreOffice creates a new module, it automatically adds the macro named *Main*.

Running a macro

- 1) Go to **Tools > Macros > Run Macro** on the main menu bar to open the Macro Selector dialog (Figure 4).
- 2) For example, select your newly created macro *EnterMyName* and click **Run**.
- 3) Alternatively, go to **Tools > Macros > Organize Macros > LibreOffice Basic** on the main menu bar to open the LibreOffice Basic Macros dialog, select your macro and click **Run**.

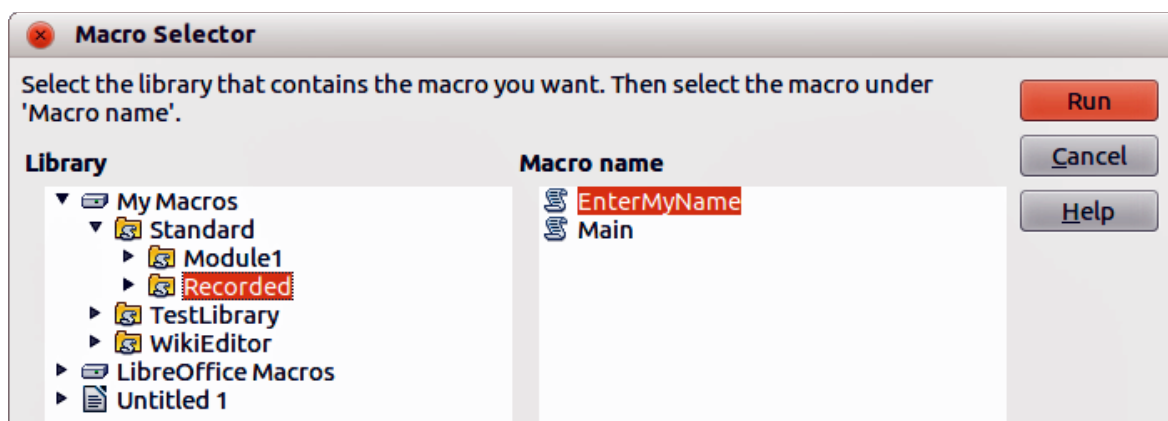


Figure 4: Macro Selector dialog

Viewing and editing macros

To view and/or edit the macro that you created:

- 1) Go to **Tools > Macros > Organize Macros > LibreOffice Basic** to open the LibreOffice Basic Macros dialog.
- 2) Select your new macro *EnterMyName* and click **Edit** to open the macro in the Basic IDE. The macro *EnterMyName* is shown in Listing 3.

The EnterMyName macro is not as complicated as it first appears. Learning a few things helps significantly in understanding macros. The discussion starts with features near the top of the macro listing and describes them.

Listing 3: Generated “EnterMyname” macro

```
REM ***** BASIC *****
Sub Main

End Sub

sub EnterMyName
rem -----
rem define variables
dim document as object
dim dispatcher as object
rem -----
rem get access to the document
document = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

rem -----
dim args1(0) as new com.sun.star.beans.PropertyValue
args1(0).Name = "Text"
args1(0).Value = "Your name"

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args1())
end sub
```

Commenting with REM

All comments in macro coding begin with REM, which stands for remark. All text after REM and on the same line is ignored. As a short cut, the single quote character (') can also be used to start a comment.

LibreOffice Basic is not case-sensitive for keywords, so REM, Rem, and rem can all start a comment. If you use symbolic constants defined by the Application Programming Interface (API), it is safer to assume that the names are case-sensitive. Symbolic constants are an advanced topic not covered by this user guide and are not required when using the macro recorder in LibreOffice.

Defining subroutines with SUB

Individual macros are stored in subroutines and these subroutines begin with the keyword SUB. The end of a subroutine is indicated by the words END SUB. The code starts by defining the subroutine named Main, which is empty and does nothing. The next subroutine, EnterMyName, contains the generated code for your macro.



Note

LibreOffice always creates an empty subroutine named Main when it creates a module.

There are advanced topics that are beyond the scope of this user guide, but knowing about them might be of interest:

- You can write a macro so that values can be passed to the subroutine. The values are called arguments. However, recorded macros in LibreOffice do not accept arguments.

- Another kind of subroutine is called a function, which is a subroutine that returns a value. Functions are defined by the keyword `FUNCTION` at the beginning. However, recorded macros in LibreOffice always create subroutines, not functions.

Defining variables using Dim

You can write information on a piece of paper so that you can look at it later. A variable, like a piece of paper, contains information that can be changed and read. The `Dim` keyword originally stood for Dimension and was used to define the dimensions of an array. The `dim` statement used in the `EnterMyName` macro is similar to setting aside a piece of paper to be used to store a message or note.

In the `EnterMyName` macro, the variables `document` and `dispatcher` are defined as the type *object*. Other common variable types include *string*, *integer*, and *date*. A third variable, named `args1`, is an array of property values. A variable of type *array* allows a single variable to contain multiple values, similar to storing multiple pages in a single book. Values in an array are usually numbered starting from zero. The number in the parentheses indicates the highest usable number to access a storage location. In this example, there is only one value, and it is numbered zero.

Explaining macro code

The following is an explanation of the code used in the `EnterMyName` macro. You may not understand all the details, but the explanation of each line of code may give you some idea of how a macro works.

```
sub EnterMyName
```

Defines the start of the macro

```
dim document as object
```

Defines `document` as a variable. Objects are a specific variable type with multiple fields representing properties and actions.

```
dim dispatcher as object
```

Defines `dispatcher` as an object variable.

```
document = ThisComponent.CurrentController.Frame
```

`ThisComponent` refers to the current document.

`CurrentController` is a property referring to a service that controls the document. For example, when you type, it is the current controller that takes note of what you type. `CurrentController` then dispatches the changes to the document frame.

`Frame` is a controller property that returns the main frame for a document. Therefore, the variable named `document` refers to a document's frame, which receives dispatched commands.

```
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
```

Most tasks in LibreOffice are accomplished by dispatching a command. LibreOffice includes a dispatch helper service, which does most of the work when using dispatches in macros. The method `CreateUnoService` accepts the name of a service and it tries to create an instance of that service. On completion, the `dispatcher` variable contains a reference to a `DispatchHelper`.

```
dim args1(0) as new com.sun.star.beans.PropertyValue
```

Declares an array of properties. Each property has a name and a value. In other words, it is a name/value pair. The created array has one property at index zero.

```
args1(0).Name = "Text"
```

```
args1(0).Value = "Your name"
```

Gives the property the name "Text" and the value "Your name", which is the text that is inserted when the macro is run.

```
dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args1())
```

This is where the magic happens. The dispatch helper sends a dispatch to the document frame (stored in the variable named document) with the command .uno:InsertText. The next two arguments, frame name and search flags, are beyond the scope of this document. The last argument is the array of property values to be used while executing the command InsertText.

```
end sub
```

The last line of the code ends the subroutine.

Creating a macro

When creating a macro, it is important to ask two questions before recording:

- 1) Can the task be written as a simple set of commands?
- 2) Can the steps be arranged so that the last command leaves the cursor ready for the next command or entering text or data into the document?

A more complicated example of a macro

A common task is to copy rows and columns of data from a web site and format them as a table in a text document as follows:

- 1) Copy the data from the web site to the clipboard.
- 2) To avoid strange formatting and fonts, paste the text into a Writer document as unformatted text.
- 3) Reformat the text with tabs between columns so that it can be converted into a table using **Table > Convert > Text to Table** on the main menu bar.

With the two questions given above in mind, inspect the text to see if a macro can be recorded to format the text. An example of copied data showing the FontWeight constants group from the API web site (Figure 5). The first column in this example indicates a constant name and each name is followed by a space and a tab, and each line has two trailing spaces.

The first column in the table should contain a numeric value, the second column the name, and the third column the description. This conversion is easily accomplished for every row except for DONTKNOW and NORMAL, which do not contain a numeric value, but the values are between 0 and 100 and can be entered manually.

DONTKNOW	The font weight is not specified/known.
THIN	specifies a 50% font weight.
ULTRALIGHT	specifies a 60% font weight.
LIGHT	specifies a 75% font weight.
SEMILIGHT	specifies a 90% font weight.
NORMAL	specifies a normal font weight.
SEMIBOLD	specifies a 110% font weight.
BOLD	specifies a 150% font weight.
ULTRABOLD	specifies a 175% font weight.
BLACK	specifies a 200% font weight.

Figure 5: Example of copied data

The data can be cleaned up in several ways, all of them easy to accomplish. The example given below uses keystrokes that assume the cursor is at the start of the line with the text THIN.

- 1) Make sure macro recording is enabled by going to **Tools > Options > LibreOffice > Advanced** on the main menu bar and selecting the option **Enable macro recording**. By default, this feature is turned off when LibreOffice was installed on your computer.
- 2) Go to **Tools > Macros > Record Macro** on the main menu bar to start recording.
- 3) Press *Ctrl+Right Arrow* to move the cursor to the start of “specifies”.
- 4) Press *Backspace* twice to remove the tab and the space.
- 5) Press *Tab* to add the tab without the space after the constant name.
- 6) Press *Delete* to delete the lower case s and then press *Shift+S* to add an upper case S.
- 7) Press *Ctrl+Right Arrow* twice to move the cursor to the start of the number.
- 8) Press *Ctrl+Shift+Right Arrow* to select and move the cursor before the % sign.
- 9) Press *Ctrl+C* to copy the selected text to the clipboard.
- 10) Press *End* to move the cursor to the end of the line.
- 11) Press *Backspace* twice to remove the two trailing spaces.
- 12) Press *Home* to move the cursor to the start of the line.
- 13) Press *Ctrl+V* to paste the selected number to the start of the line.
- 14) Pasting the value also pasted an extra space, so press *Backspace* to remove the extra space.
- 15) Press *Tab* to insert a tab between the number and the name.
- 16) Press *Home* to move to the start of the line.
- 17) Press *down arrow* to move to the next line.
- 18) Stop recording the macro and save the macro, see “Recording a macro” on page 6.

It takes much longer to read and write the steps than to record the macro. Work slowly and think about the steps as you do them. With practice this becomes second nature.

The generated macro code in Listing 4 has been modified to contain the step number in the comments to match the code to the step above.

Listing 4: Copying numeric value to start of the column

```
sub CopyNumToCol1
rem -----
rem define variables
dim document as object
dim dispatcher as object
rem -----
rem get access to the document
document = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

rem (3) Press Ctrl+Right Arrow to move the cursor to the start of "specifies".
dispatcher.executeDispatch(document, ".uno:GoToNextWord", "", 0, Array())

rem (4) Press Backspace twice to remove the tab and the space.
dispatcher.executeDispatch(document, ".uno:SwBackspace", "", 0, Array())

rem -----
dispatcher.executeDispatch(document, ".uno:SwBackspace", "", 0, Array())

rem (5) Press Tab to add the tab without the space after the constant name.
```

```

dim args4(0) as new com.sun.star.beans.PropertyValue
args4(0).Name = "Text"
args4(0).Value = CHR$(9)

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args4())

rem (6) Press Delete to delete the lower case s ....
dispatcher.executeDispatch(document, ".uno:Delete", "", 0, Array())

rem (6) ... and then press Shift+S to add an upper case S.
dim args6(0) as new com.sun.star.beans.PropertyValue
args6(0).Name = "Text"
args6(0).Value = "S"

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args6())

rem (7) Press Ctrl+Right Arrow twice to move the cursor to the number.
dispatcher.executeDispatch(document, ".uno:GoToNextWord", "", 0, Array())

rem -----
dispatcher.executeDispatch(document, ".uno:GoToNextWord", "", 0, Array())

rem (8) Press Ctrl+Shift+Right Arrow to select the number.
dispatcher.executeDispatch(document, ".uno:WordRightSel", "", 0, Array())

rem (9) Press Ctrl+C to copy the selected text to the clipboard.
dispatcher.executeDispatch(document, ".uno:Copy", "", 0, Array())

rem (10) Press End to move the cursor to the end of the line.
dispatcher.executeDispatch(document, ".uno:GoToEndOfLine", "", 0, Array())

rem (11) Press Backspace twice to remove the two trailing spaces.
dispatcher.executeDispatch(document, ".uno:SwBackspace", "", 0, Array())

rem -----
dispatcher.executeDispatch(document, ".uno:SwBackspace", "", 0, Array())

rem (12) Press Home to move the cursor to the start of the line.
dispatcher.executeDispatch(document, ".uno:GoToStartOfLine", "", 0, Array())

rem (13) Press Ctrl+V to paste the selected number to the start of the line.
dispatcher.executeDispatch(document, ".uno:Paste", "", 0, Array())

rem (14) Press Backspace to remove the extra space.
dispatcher.executeDispatch(document, ".uno:SwBackspace", "", 0, Array())

rem (15) Press Tab to insert a tab between the number and the name.
dim args17(0) as new com.sun.star.beans.PropertyValue
args17(0).Name = "Text"
args17(0).Value = CHR$(9)

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args17())
rem (16) Press Home to move to the start of the line.
dispatcher.executeDispatch(document, ".uno:GoToStartOfLine", "", 0, Array())

rem (17) Press Down Arrow to move to the next line.
dim args19(1) as new com.sun.star.beans.PropertyValue

```


```
args19(0).Name = "Count"
args19(0).Value = 1
args19(1).Name = "Select"
args19(1).Value = false

dispatcher.executeDispatch(document, ".uno:GoDown", "", 0, args19())
end sub
```

Cursor movements are used for all operations (as opposed to searching). If run on the DONTKNOW line, the word *weight* is moved to the front of the line, and the first “The” is changed to “She”. This is not perfect, but you should not run the macro on the lines that did not have the proper format. You need to do these manually.

Running a macro quickly

It is tedious to repeatedly run the macro using **Tools > Macros > Run Macro** on the main menu bar when the macro can be run from the IDE (Figure 3 on page 6).

- 1) Go to **Tools > Macros > Organize Macros > LibreOffice Basic** on the main menu bar to open the Basic Macro dialog (Figure 1 on page 5).
- 2) Select your macro and click **Edit** to open the macro in the IDE.
- 3) Click the **Run BASIC** icon  on the Macro toolbar, or press the *F5* key, to run the macro.
- 4) Unless you change the first macro, it is the empty macro named Main. Modify Main so that it reads as shown in Listing 5.
- 5) Now, you can run CopyNumToCol1 by repeatedly clicking the **Run Basic** icon in the toolbar of the IDE. This is very fast and easy, especially for temporary macros that will be used a few times and then discarded.

Listing 5: Modify Main to call CopyNumToCol1.

```
Sub Main
  CopyNumToCol1
End Sub
```

Macro recorder failures

Sometimes the macro recorder has a failure and understanding LibreOffice internal workings helps to understand how and why the macro recorder sometimes fails. The primary offender is related to the dispatch framework and its relationship to the macro recorder.

Dispatch framework

The purpose of the dispatch framework is to provide uniform access to components (documents) for commands that usually correspond to menu items. Using **File > Save** from the main menu bar, the shortcut keys *Ctrl+S*, or clicking the **Save** icon are all of commands that are translated into the same “dispatch command”.

The dispatch framework can also be used to send “commands” back to the User Interface (UI). For example, after saving a new document, the list of recent files is updated.

A dispatch command is text, for example `.uno:InsertObject` or `.uno:GoToStartOfLine`. The command is sent to the document frame and this passes on the command until an object is found that can handle the command.

How the macro recorder uses the dispatch framework

The macro recorder records the generated dispatches. The recorder is a relatively simple tool to use and the same commands that are issued are recorded for later use. The problem is that not all dispatched commands are complete. For example, inserting an object generates the following code:

```
dispatcher.executeDispatch(document, ".uno:InsertObject", "", 0, Array())
```

It is not possible to specify what kind of object to create or insert. If an object is inserted from a file, you cannot specify which file to insert.

If while recording a macro you use **Tools > Options** on the main menu bar to open and modify configuration items, the generated macro does not record any configuration changes. In fact, the generated code is commented so it will not even be run.

```
rem dispatcher.executeDispatch(document, ".uno:OptionsTreeDialog", "", 0, Array())
```

If a dialog is opened, a command to open the dialog is likely to be generated. Any work done inside the dialog is not usually recorded. Examples of this include macro organization dialogs, inserting special characters, and similar types of dialogs. Other possible problems using the macro recorder include things such as inserting a formula, setting user data, setting filters in Calc, actions in database forms, and exporting a document to an encrypted PDF file. You never know for certain what will work unless you try it. For example, the actions from the search dialog are properly captured.

Other options

When the macro recorder is not able to solve a specific problem, the usual solution is to write code using the LibreOffice objects. Unfortunately, there is a steep learning curve for these LibreOffice objects. It is usually best to start with simple examples and then increase the scope of macros as you learn more. Learning to read generated macros is a good place to start.

If you record Calc macros, and the recorder can correctly generate a macro, there is an add-in available which converts Calc macros when they are recorded. The final code manipulates LibreOffice objects rather than generating dispatches. This can be very useful for learning the object model and can be downloaded directly from the web site:

<http://www.paolo-mantovani.org/downloads/DispatchToApiRecorder/>

Macro organization

In LibreOffice, macros are grouped in modules, modules are grouped in libraries, and libraries are grouped in library containers. A library is usually used as a major grouping for either an entire category of macros, or for an entire application. Modules usually split functionality, such as user interaction and calculations. Individual macros are subroutines and functions. Figure 6 shows an example of the hierarchical structure of macro libraries in LibreOffice.

Go to **Tools > Macros > Organize Macros > LibreOffice Basic** on the main menu bar to open the LibreOffice Basic Macros dialog (Figure 1 on page 5). All available library containers are shown in the *Macro from* list. Every document is a library container, capable of containing multiple libraries. The application itself acts as two library containers, one container for macros distributed with LibreOffice called LibreOffice Macros, and one container for personal macros called My Macros.

The LibreOffice Macros are stored with the application runtime code, which may not be editable to you unless you are an administrator. This helps protect these macros because they should not be changed and you should not store your own macros in the LibreOffice container.

Unless your macros are applicable to a single document, and only to a single document, your macros will probably be stored in the My Macros container. The My Macros container is stored in your user area or home directory.

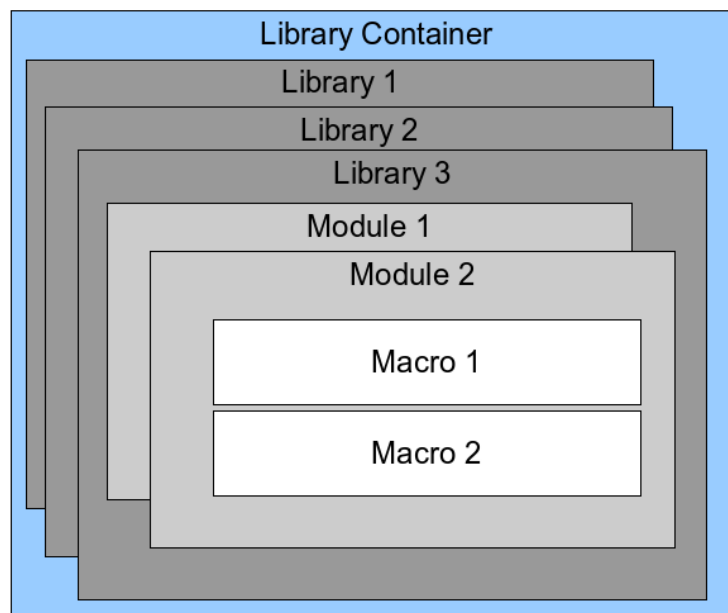


Figure 6: Macro Library hierarchy

If a macro is contained in a document, then a recorded macro will attempt to work on that document; because it primarily uses “ThisComponent” for its actions.

Every library container contains a library named *Standard*. It is better to create your own libraries with meaningful names than to use the Standard library. Not only are meaningful names easier to manage, but they can also be imported into other library containers whereas the Standard library cannot.



Caution

LibreOffice allows you to import libraries into a library container, but it will not allow you to overwrite the library named *Standard*. Therefore, if you store your macros in the *Standard* library, you cannot import them into another library container.

Just as it makes good sense to give your libraries meaningful names, it is prudent to use meaningful names for your modules. By default, LibreOffice uses names such as *Module1*, *Module2* and so on.

As you create your macros, you must decide where to store them. Storing a macro in a document is useful if the document will be shared and you want the macro to be included with the document. Macros stored in the application library container named *My Macros*, however, are globally available to all documents.

Macros are not available until the library that contains them is loaded. The *Standard* library and *Template* library, however, are automatically loaded. A loaded library is displayed differently from a library that is not loaded. To load the library and the modules it contains, double-click on the library.

Where are macros stored?

LibreOffice stores user-specific data in a directory in the home directory for each user. The location is operating system specific. Go to **Tools > Options > LibreOffice > Paths** on the main menu bar to view where other configuration data are stored. For example, on computer running Windows XP, this is `C:\Documents and Settings\<user name>\Application Data`. User macros are

stored in LibreOffice\4\user\basic. Each library is stored in its own directory off the basic directory.

For casual use, it is not necessary to understand where macros are stored. If you know where they are stored, however, you can create a backup, share your macros, or inspect them if there is an error.

Go to **Tools > Macros > Organize Dialogs** on the main menu bar to open the LibreOffice Macro Organizer dialog (Figure 2 on page 5). Alternatively, go to **Tools > Macros > Organize Macros > LibreOffice Basic** on the main menu bar to open the LibreOffice Macros dialog (Figure 1 on page 5) and then click the **Organizer** button.

Importing macros

The LibreOffice Macro Organizer dialog allows you to import macro libraries into your document as well as creating, deleting, and renaming libraries, modules, and dialogs.

- 1) Select the library container to use and then click **Import** to import macro libraries (Figure 2 on page 5).
- 2) Navigate to the directory containing the library to import (Figure 7). There are usually two files from which to choose, dialog.xlb and script.xlb. It does not matter which of these two files you select; both will be imported. Macros can be stored in libraries inside LibreOffice documents. Select a document rather than a directory on disk to import libraries contained in a document.



Note

You cannot import the library named Standard.

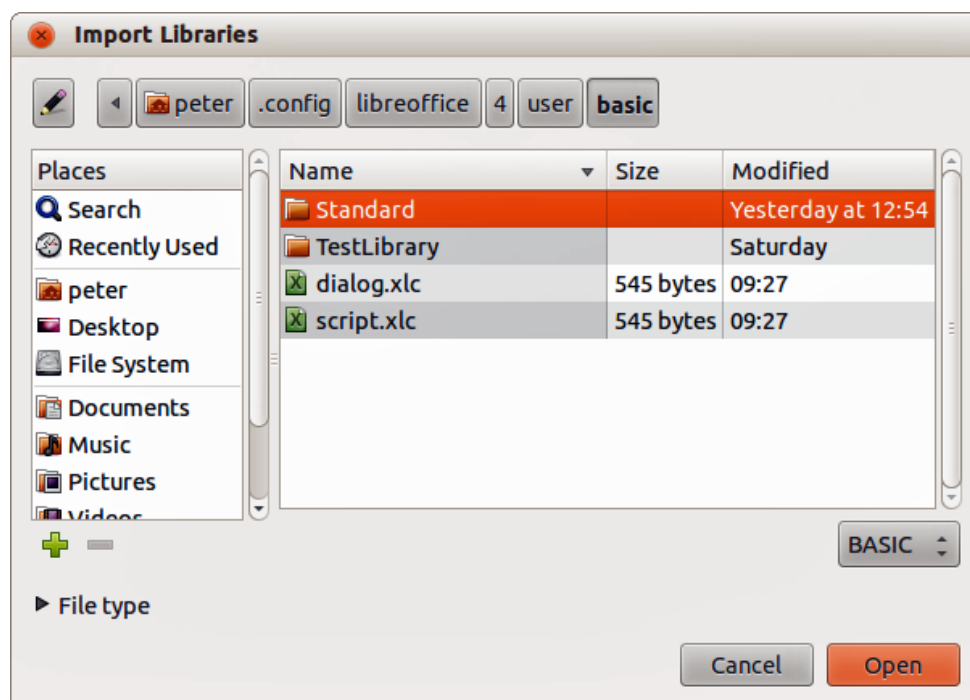


Figure 7: Navigating to a macro library

Tip

On a computer operating Linux, the LibreOffice specific files are stored in the home directory of a user, in a subdirectory whose name begins with a period (usually .config/). Directories and files with names beginning with a period may be hidden and not shown in a normal selection dialog. If using LibreOffice dialogs, rather than the operating system specific dialogs, type the name of the desired directory in the Name field.

- 3) Select a file and click **Open** to continue and open the Import Libraries dialog (Figure 8).

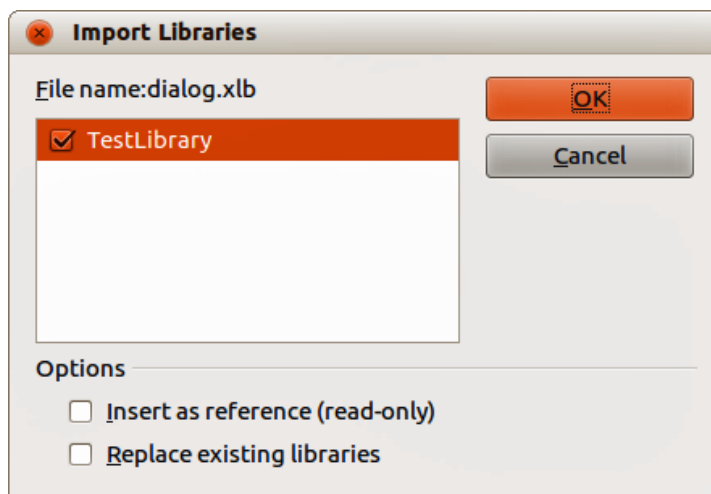


Figure 8: Choose library import options

- 4) Select the following options for importing libraries:
 - a) If no options are selected, the library is copied to your user macro directory. However, if the library you are importing has the same name and you are importing into the same location, it will not be copied.
 - b) Select **Replace existing libraries** if the library you want to import has the same name and you want to replace the existing library.
 - c) Select **Insert as reference** if you want to use the library as reference, but not import the library into your document. When a library is used as a reference, it remains in its current location and is read only.
- 5) Click **OK** to import the macro library you selected.

Downloading macros to import

Macros are available for download. Some macros are contained in documents, some as regular files that you must select and import, and some as macro text that should be copied and pasted into the Basic IDE. See “Adding a macro” on page 4 on how to add macros to your macro library and “Viewing and editing macros” on page 7 on how to edit macros using the Basic IDE.

Some macros are available as free downloads on the Internet (see Table 1).

Table 1. Macro examples

Location	Description
http://www.pitonyak.org/oo.php	Reference materials regarding macros.
http://www.pitonyak.org/database/	Reference materials regarding database macros.
https://wiki.documentfoundation.org/Macros	Lots of links to macros.
http://en.libreofficeforum.org/ http://forum.openoffice.org/en/forum/	Forums, with many examples and help.

How to run a macro

Although you can use **Tools > Macros > Run Macro** to run all macros, this is not efficient for frequently run macros. See “Running a macro” on page 7 for more information.

A more common technique for frequently used macros is to link the macro to a toolbar icon, menu item, keyboard shortcut, or a button embedded in a document. While choosing a method, it is also good to ask questions such as:

- Should the macro be available for only one document, or globally for all documents?
- Is the macro for a specific document type, such as a Calc document?
- How frequently will the macro be used?

The answers will determine where to store the macro and how to make it available. For example, you will probably not add a rarely used macro to a toolbar. To help determine your choices, see Table 2.

Table 2. Where to store a macro

Type of macro	LibreOffice (for all components)	A specific document type	One document
Toolbar	No	Yes	Yes
Menu	No	Yes	Yes
Shortcut	Yes	Yes	No
Event	Yes	No	Yes

Toolbars, menu items, and keyboard shortcuts

To add a menu item, keyboard shortcut, or toolbar icon that calls a macro, use the Customize dialog (Figure 9).

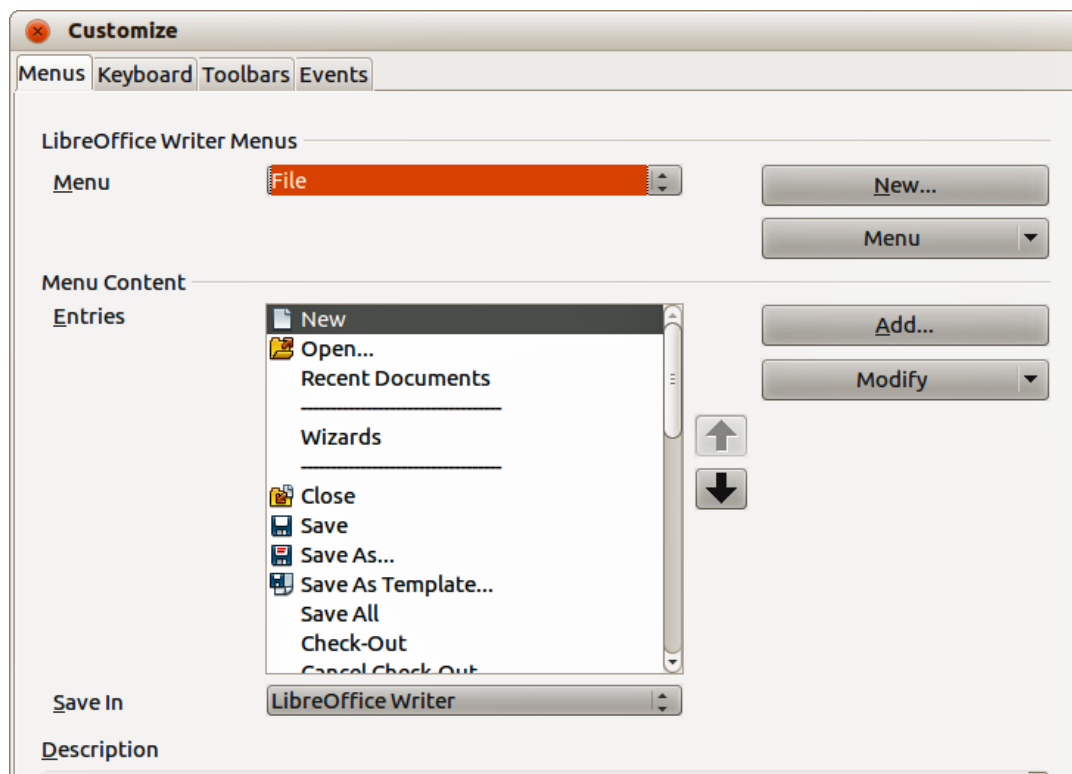


Figure 9: Menus page in Customize dialog

The Customize dialog contains pages to configure menus, keyboard shortcuts, toolbars, and events. To open this dialog, go to **Tools > Customize** on the main menu bar or right-click in an empty space on a toolbar and select **Customize Toolbar** from the context menu.

Complete coverage of the Customize dialog is beyond the scope of this chapter. Click the **Help** button to access the help pages included with LibreOffice or see *Chapter 14 Customizing LibreOffice*.

Events

Whenever something happens in LibreOffice, it is called an event. For example, opening a document, pressing a key, or moving the mouse cursor are all events. LibreOffice allows events to trigger the execution of a macro; the macro is then called an event handler. Full coverage of event handlers is well beyond the scope of this document, but a little knowledge can accomplish much.



Caution

Be careful when you configure an event handler. For example, assume that you write an event handler that is called every time that a key is pressed, but you make a mistake so the event is not properly handled. One possible result is that your event handler will consume all key presses, forcing you to kill LibreOffice.

- 1) Go to **Tools > Customize** on the main menu bar to open the Customize dialog and select the Events tab (Figure 10). The events in the Customize dialog are related to the entire application and specific documents.

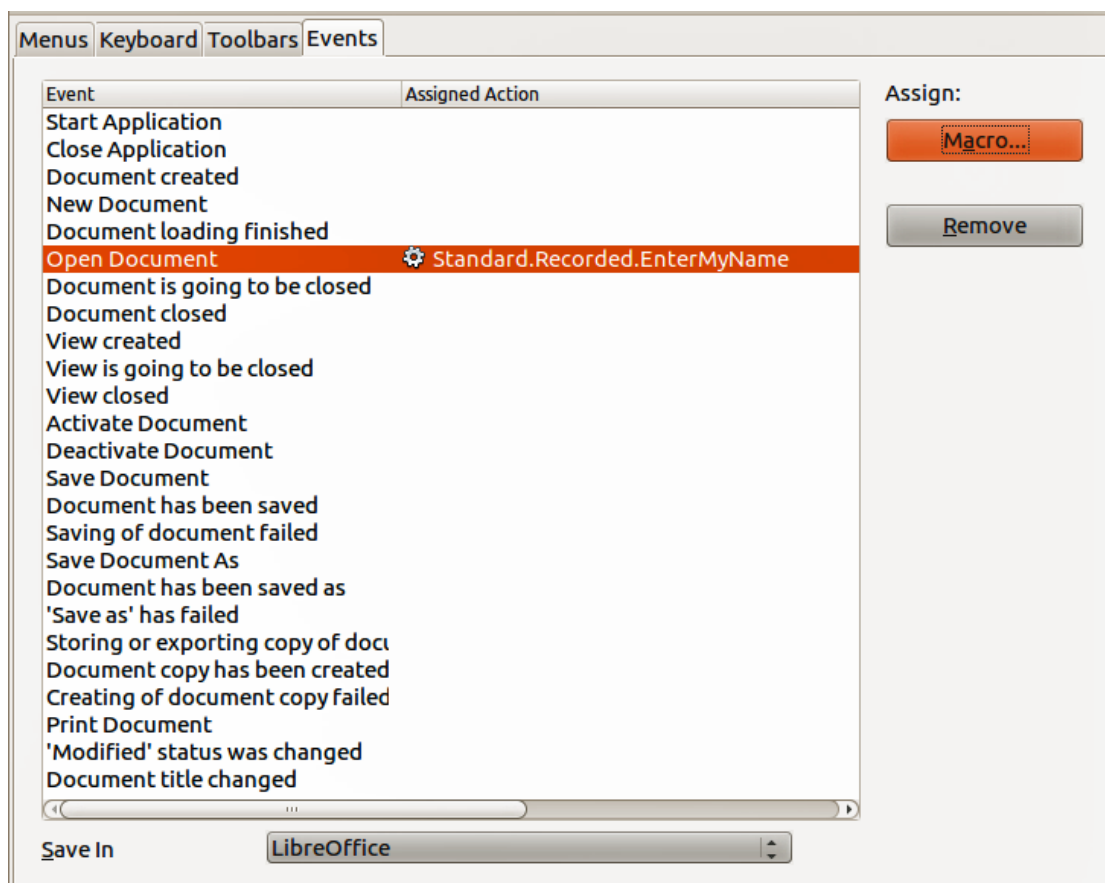


Figure 10: Events page in Customize dialog

- 2) In **Save In**, select LibreOffice, or a specific document from the drop down menu to save your event.

- 3) A common use is to assign the Open Document event to call a specific macro. The macro then performs certain setup tasks for the document. Select the desired event and click **Macro** to open the Macro Selector dialog (Figure 4 on page 7).
- 4) Select the desired macro and click **OK** to assign the macro to the event. The Events page shows that the event has been assigned to a macro.

Many objects in a document can be set to call macros when events occur. The most common use is to add a control, such as a button, into a document. Even double-clicking on a graphic opens a dialog with a Macros tab that allows you to assign a macro to an event.

Extensions

An extension is a package that can be installed into LibreOffice to add new functionality. Extensions can be written in almost any programming language and may be simple or sophisticated. Extensions can be grouped into types:

- Calc Add-Ins, which provide new functionality for Calc, including new functions that act like normal built-in functions
- New components and functionality, which normally include some level of User Interface (UI) integration such as new menus or toolbars
- Pivot Tables that are used directly in Calc
- Chart Add-Ins with new chart types
- Linguistic components such as spelling checkers
- Document templates and images

Although individual extensions can be found in several places, there is currently an extension repository at: <http://extensions.libreoffice.org/> and some documentation at <http://libreplanet.org/wiki/Group:OpenOfficeExtensions/List>.

For more about obtaining and installing extensions, see Chapter 14, Customizing LibreOffice.

Writing macros without the recorder

The examples covered in this chapter are created using the macro recorder and the dispatcher. You can also write macros that directly access the objects that comprise LibreOffice if you are confident in writing computer code. In other words, you can create a macro that directly manipulates a document.

Directly manipulating LibreOffice internal objects is an advanced topic that is beyond the scope of this chapter. A simple example, however, demonstrates how this works.

Listing 6: Append the text "Hello" to the current document.

```
Sub AppendHello
    Dim oDoc
    Dim sTextService$
    Dim oCurs

    REM ThisComponent refers to the currently active document.
    oDoc = ThisComponent

    REM Verify that this is a text document
    sTextService = "com.sun.star.text.TextDocument"
    If NOT oDoc.supportsService(sTextService) Then
        MsgBox "This macro only works with a text document"
        Exit Sub
    End If
```

```

REM Get the view cursor from the current controller.
oCurs = oDoc.currentController.getViewCursor()

REM Move the cursor to the end of the document
oCurs.gotoEnd(False)

REM Insert text "Hello" at the end of the document
oCurs.Text.insertString(oCurs, "Hello", False)
End Sub

```

Finding more information

Numerous resources are available that provide help with writing macros. Use **Help > LibreOffice Help** to open the LibreOffice help pages. The upper left corner of the LibreOffice help system contains a drop-down list that determines which help set is displayed. To view the help for Basic, choose *LibreOffice Basic* from this list.

Included material

Many excellent macros are included with LibreOffice. Use **Tools > Macros > Organize Macros > LibreOffice Basic** to open the Macro dialog. Expand the Tools library in the LibreOffice library container. Inspect the Debug module—some good examples include `WritedbgiInfo(document)` and `printdbgiInfo(sheet)`.

Online resources

The following links and references contain information regarding macro programming:

<http://ask.libreoffice.org/> (a Q & A site where volunteers answer questions related to LibreOffice)

<http://forum.openoffice.org/en/forum/> (Apache OpenOffice community forum; volunteers answer questions about LibreOffice as well)

<http://en.libreofficeforum.org/> (Unofficial LibreOffice community forum)

<http://api.openoffice.org/docs/common/ref/com/sun/star/module-ix.html> (official IDL reference; here you will find almost every command with a description)

https://wiki.documentfoundation.org/Documentation/Other_Documentation_and_Resources (look in Programmers section for *BASIC Programmers' Guide* and *Developers' Guide*; the latter contains a detailed explanation)

<http://www.pitonyak.org/oo.php> (macro page for Andrew Pitonyak)

<http://www.pitonyak.org/AndrewMacro.odt> (numerous examples of working macros)

http://www.pitonyak.org/OOME_3_0.odt (Andrew Pitonyak's book on macros)

<http://www.pitonyak.org/database/> (numerous macro examples using Base)

Printed and eBook materials

There are currently no books specific to LibreOffice macros that are available for download.

Information in the following books is mostly applicable to LibreOffice; the books are available for purchase in both printed and eBook form from their publishers:

Dr. Mark Alexander Bain's *Learn OpenOffice.org Spreadsheet Macro Programming*.

See <http://www.packtpub.com/openoffice-ooobasic-calc-automation/book>.

Roberto Benitez's *Database Programming with OpenOffice.org Base & Basic*.

See <http://www.lulu.com/product/paperback/database-programming-with-openofficeorg-base-basic/3568728>