Getting Started Guide

Chapter 13
Getting Started with Macros

Using the Macro Recorder … and Beyond

Copyright

Contributors

This book is updated from *Getting Started with LibreOffice 6.0*.

To this edition

Jean Hollis Weber
Steve Fanning
Kees Kriek

To previous editions

Andrew Pitonyak
Ron Faile Jr.
Valerii Goncharuk
Peter Schofield
Hazel Russman
Andrew Jensen
Martin Fox
Olivier Hallot
Jean Hollis Weber

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.

Publication date and software version

Published September 2020. Based on LibreOffice 6.4.

Note for macOS users

Some keystrokes and menu items are different on macOS 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	*macOS equivalent*	*Effect*
Tools > Options	**LibreOffice > Preferences**	Access setup options
Right-click	Control+click and/or right-click depending on computer setup	Open a context menu
Ctrl (Control)	⌘ *(Command)*	Used with other keys
F5	*Shift*+⌘+*F5*	Open the Navigator
F11	⌘+*T*	Open the Sidebar Styles deck

Contents

Introduction

A macro is a set of commands or keystrokes that are stored for later use. An example of a simple macro is one that enters your address into an open document. You can use macros to automate both simple and complex tasks. Macros are very useful when you have to repeat the same task in the same way.

The simplest way to create a macro is to record a series of actions through LibreOffice's user interface. LibreOffice saves recorded macros using the open source LibreOffice Basic scripting language, which is a dialect of the well-known BASIC programming language. Such macros can be edited and enhanced after recording using the built-in LibreOffice Basic Integrated Development Environment (IDE).

The most powerful macros in LibreOffice are created by writing code using one of the four supported scripting languages (LibreOffice Basic, BeanShell, JavaScript, and Python). This chapter provides an overview of LibreOffice's macro facilities, mostly focused on its default macro scripting language, LibreOffice Basic. Some examples are included for the BeanShell, JavaScript, and Python scripting languages but fuller descriptions of the facilities for these languages are beyond the scope of this chapter.

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 should create a library and module to contain your macro; see "Macro organization" on page 15 for more information.

Listing 1: Simple macro that says hello

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

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

1) Go to **Tools > Macros > Organize Macros > Basic** to open the LibreOffice Basic Macros dialog (Figure 1).
2) Click **Organizer** to open the LibreOffice Basic Macro Organizer dialog (Figure 2) and select the *Libraries* tab.
3) Set the *Location* drop-down to **My Macros & Dialogs**, which is the default location.
4) Click **New** to open the New Library dialog (not shown here).
5) Enter a library name, for example *TestLibrary*, and click **OK**.
6) On the LibreOffice Basic Macro Organizer dialog, select the *Modules* tab (Figure 3).
7) In the *Module* list, expand *My Macros* and select your library (in the 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 4). The IDE is a text editor and associated facilities that are built into LibreOffice and allow you to create, edit, run, and debug macros.

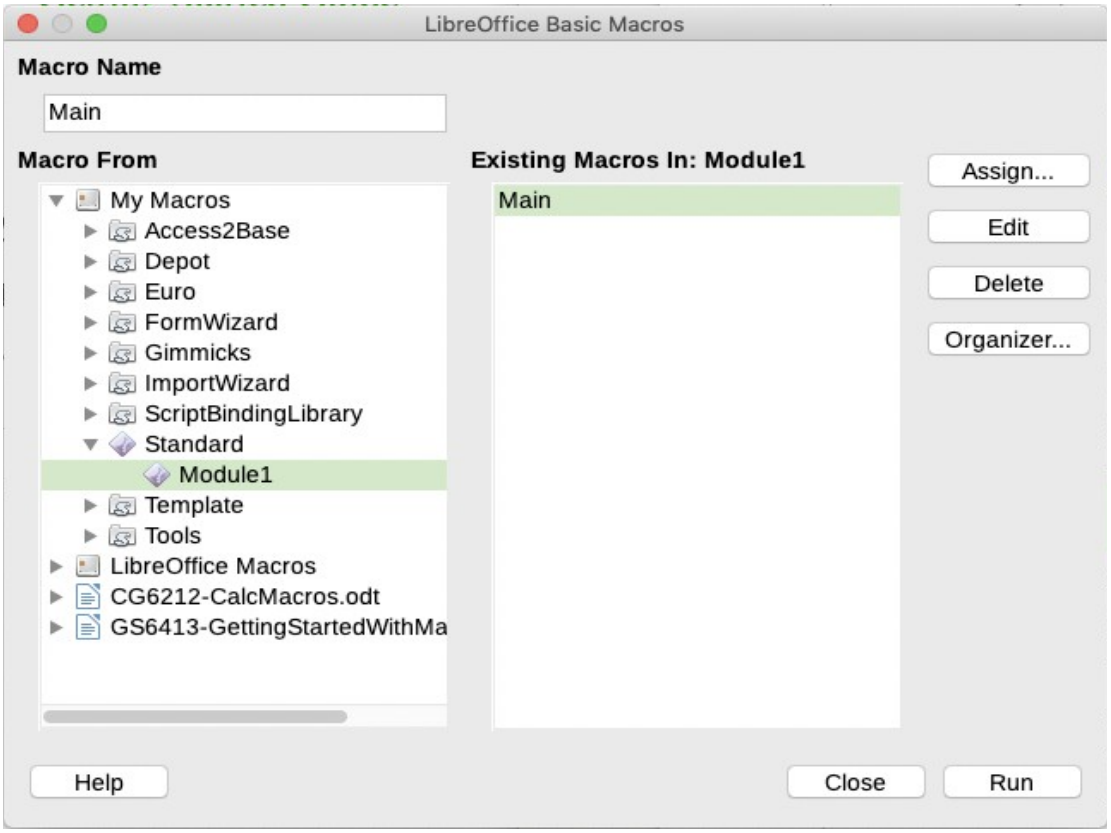

Figure 1: LibreOffice Basic Macros dialog

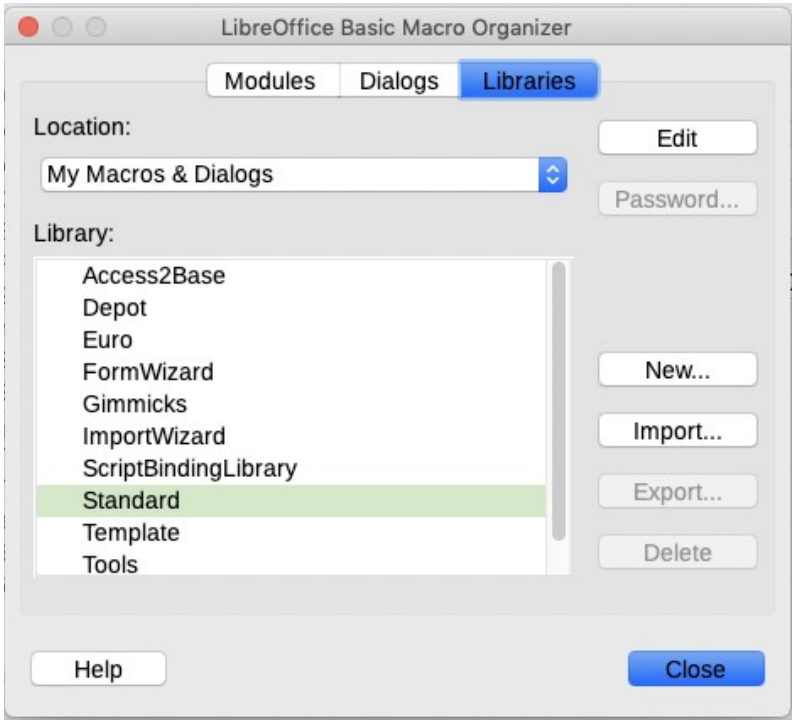

Figure 2: LibreOffice Basic Macro Organizer dialog, Libraries tab

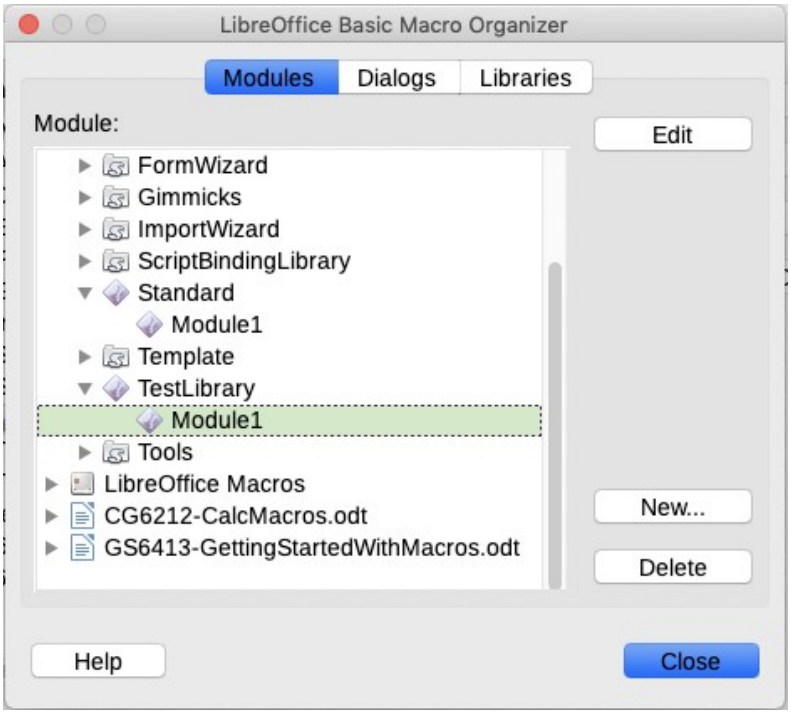

Figure 3: LibreOffice Basic Macro Organizer dialog, Modules tab

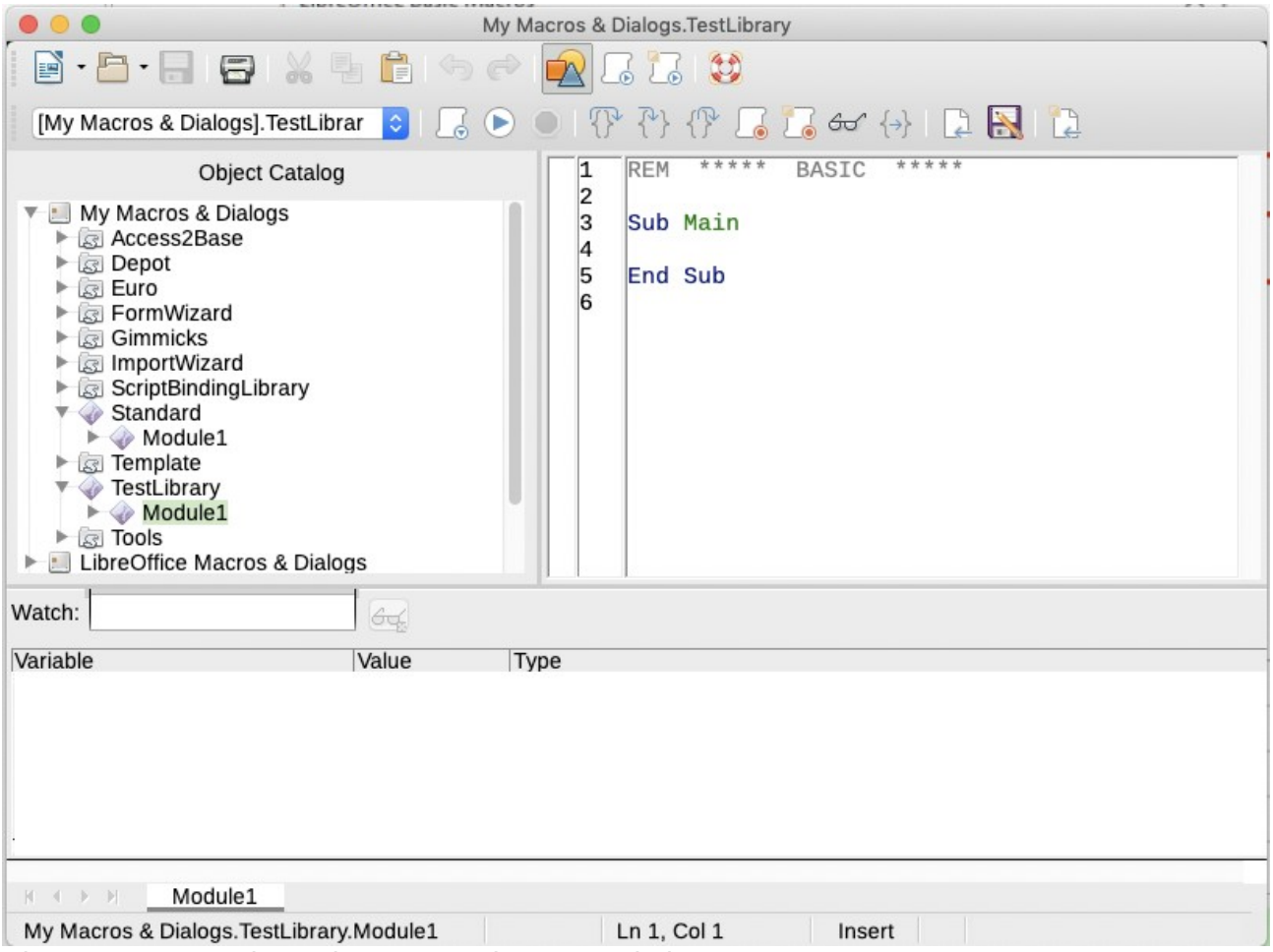

Figure 4: Integrated Development Environment window

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`.
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** icon on the Macro toolbar, or press the *F5* key, to run the `HelloMacro` subroutine 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 > Organize Macros > Basic**.
15) Select a macro and then click **Run**.

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 of 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 cases when you want to repeatedly enter information into a document, it may be more convenient to create an AutoText. See Chapter 2, Working with Text: Basics, in the *Writer Guide* for more information.

Make sure macro recording is enabled by going to **Tools > Options > LibreOffice > Advanced** and selecting the option **Enable macro recording** under *Optional Features*. By default, this feature is turned off in LibreOffice.

1) Go to **Tools > Macros > Record Macro** to start recording a macro. A small dialog with a **Stop Recording** button 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 dialog and the LibreOffice Basic Macros dialog opens (similar to Figure 1 on page 5, but with different action buttons).
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 *RecordsInsertText*, 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 *RecordsInsertText* 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** to open the Macro Selector dialog (Figure 5).
2) For example, select your newly created macro *EnterMyName* and click **Run**.
3) Alternatively, go to **Tools > Macros > Organize Macros > Basic** to open the LibreOffice Basic Macros dialog (Figure 1), select your macro and click **Run**.

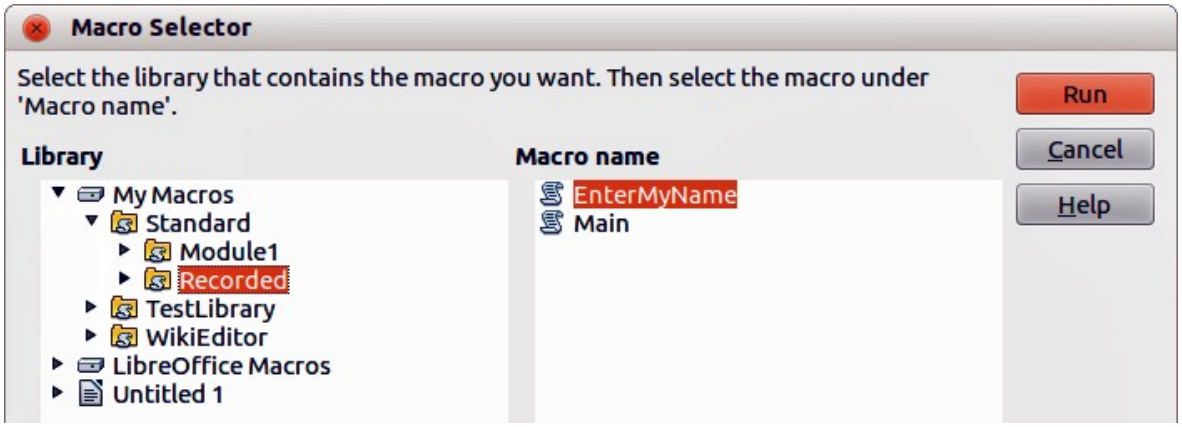

Figure 5: Macro Selector dialog

Viewing and editing macros

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

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

This first macro is not complicated. A little explanation will significantly help you in understanding macros. The discussion starts with first line of the macro and describes features through the whole listing.

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 Basic macro coding begin with REM, which stands for remark. All text after REM on the same line is ignored.

Tip

As a short cut, you can use the single quote character (') 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.

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 can return a value as a result of its work. Functions are defined by the keyword `FUNCTION` at the beginning. Recorded macros in LibreOffice create subroutines only.

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 `EnterMyName` macro.

```
dim document   as object
```

Defines `document` as an object variable. Objects are a specific variable type with multiple fields (sometimes they are called properties) and actions (also they are called methods). The fields can be perceived like variables (including an object) and actions like subroutines which allow us to operate with the object.

Note

Sometimes the word service is used. A service is supplied by a type of object which are distinguished in order to point out how they are used.

```
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 `DispatchHelper` service, which provides an easy way to dispatch a URL using one call instead of multiple ones and 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 the `args1` 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.

The `com.sun.star.beans.PropertyValue` expression is a Universal Network Objects (UNO) structure. Structures are special variable types that contain other variables united by logical basis. They can be convenient to operate with sets of heterogeneous information that should be treated as a single whole. An explanation of UNO and user structures goes far beyond the scope of this book. For more information on creating and using structures, see the LibreOffice Help system and other Basic guides.

```
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 `document`) with the command `.uno:InsertText`. The next two arguments, frame name and search flags, are beyond the scope of this book. 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 website and format them as a table in a text document as follows:

1) Copy the data from the website 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**.

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 website is shown in Figure 6. 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.

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 6: Example of copied data

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.

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** and selecting the option **Enable macro recording**. By default, this feature is turned off when LibreOffice is installed on your computer.
2) Go to **Tools > Macros > Record Macro** 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 7.

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 steps 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** when the macro can be run from the IDE (Figure 4 on page 6).

Here are a few tips to do the work easier. If you create a temporary macro using the same name, you can add that macro into the *Main* subroutine and delete your macro after using.

1) Go to **Tools > Macros > Organize Macros > Basic** to open the LibreOffice Basic Macros dialog (Figure 1 on page 5).
2) Select your macro and click **Edit** to open the macro in the IDE.
3) Unless you changed the first macro, it is the empty macro named *Main*. Modify *Main* so that it reads as shown in Listing 5.
4) Now, you can run *CopyNumToCol1* when you run the *Main* macro by clicking the **Run** icon in the Macro toolbar of the IDE, or pressing *F5*.
5) Go to **Tools > Customize** and select the *Keyboard* tab in the Customize dialog.
6) Select a shortcut key in the *Shortcut Keys* list. Then select *LibreOffice Macros*, and further, a library container, a library, and a module in the *Category* list where your modified *Main* macro is. Select the *Main* macro in the *Function* list and click the **Modify** button.

Now you can run your macro by the shortcut. This is very fast and easy, especially for temporary macros that will be used a few times and then discarded. See “How to run a macro” on page 19 for more information.

Listing 5: Modify Main to call CopyNumToCol1

```
Sub Main
  CopyNumToCol1
End Sub
```

Macro recorder failures

Sometimes the macro recorder has a failure and knowledge of 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**, the shortcut keys *Ctrl+S*, or clicking the **Save** icon on the Standard toolbar are all 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** 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.

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 7 shows an example of the hierarchical structure of macro libraries in LibreOffice.

Go to **Tools > Macros > Organize Macros > Basic** 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 Macros* 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.

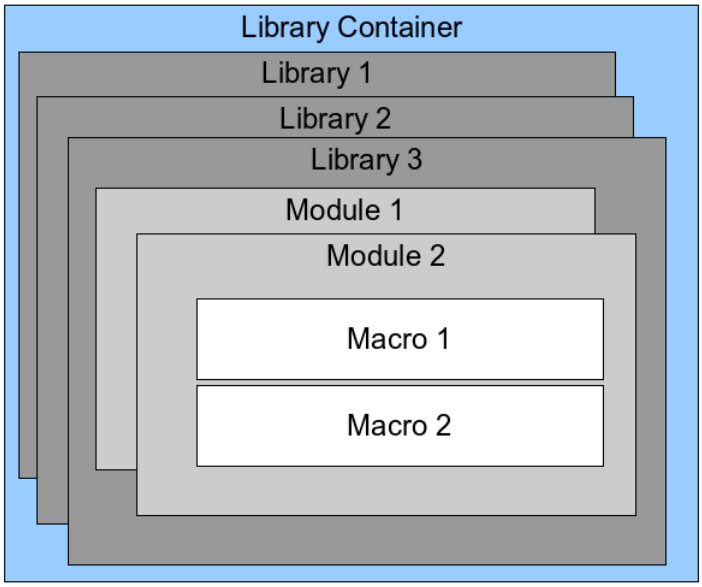

Figure 7: Macro Library hierarchy

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. However, in contrast to other libraries, the *Standard* and *Template* libraries 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 subdirectory of the home directory for the user. The location is operating system specific. Go to **Tools > Options > LibreOffice > Paths** to view where other configuration data are stored. User macros written in Basic 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** to open the LibreOffice Basic Macro Organizer dialog (Figure 2 on page 5). Alternatively, go to **Tools > Macros > Organize Macros > Basic** to open the LibreOffice Basic Macros dialog (Figure 1 on page 5) and then click the **Organizer** button.

Importing macros

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

1) On the *Libraries* tab, select the library container to use and then click **Import** to import macro libraries.
2) Navigate to the directory containing the library to import (Figure 8). 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*.

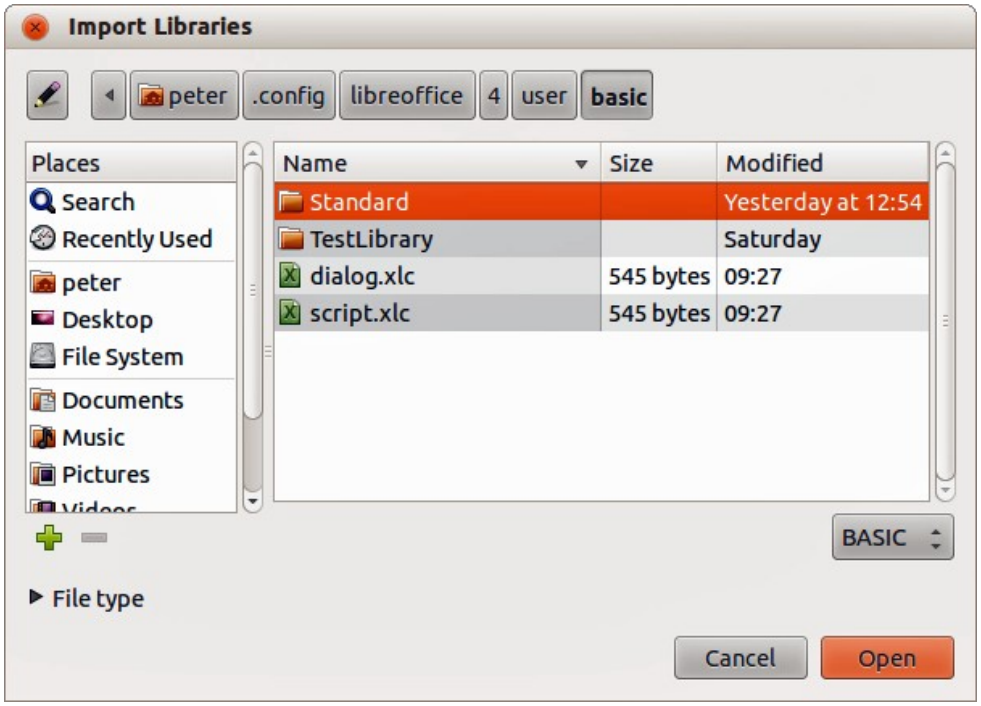

Figure 8: Navigating to a macro library

Tip

On Linux, the LibreOffice-specific files are stored in the home directory of a user, in a subdirectory whose name begins with a dot prefix (usually .config/). Directories and files with names beginning with a dot prefix 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 9).

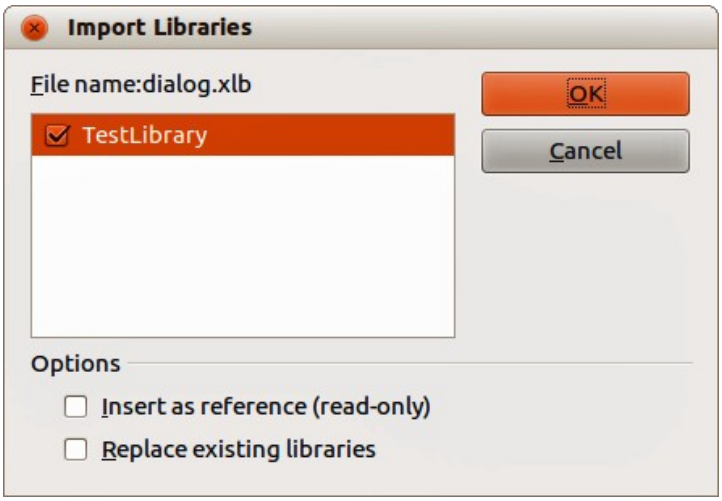

Figure 9: Choose library import options

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 (read-only)** if you want to use the library as reference, but not import it into the document. When a library is used as a reference, it remains in its current location and is fully functional, but cannot be modified in the Basic IDE.
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 8 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://forum.openoffice.org/en/forum/	Forums, with many examples and help.

How to run a macro

Although you can use **Tools > Macros > Run** to run all macros, this is not efficient for frequently run macros. See “Running a macro” on page 8 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.

The Customize dialog contains pages to configure menus, keyboard shortcuts, toolbars, and events. To open this dialog, go to **Tools > Customize**. Use of the *Menus*, *Toolbars*, *Context Menus*, and *Keyboard* tabs are covered in Chapter 14, Customizing LibreOffice.

Events

Whenever something happens in LibreOffice, it is called an event. For example, opening a document, changing status of modified, 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 chapter, 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 document is modified, but you make a mistake so the event is not properly handled. One possible result is that your event handler will force you to kill LibreOffice.

1) Go to **Tools > Customize** 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.
2) In the *Save In* drop-down, select **LibreOffice**, or a specific document from the 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 (similar to Figure 5 on page 8 but with different action buttons).

4) Select the desired macro and click **OK** to assign the macro to the event. The *Events* tab will show 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.

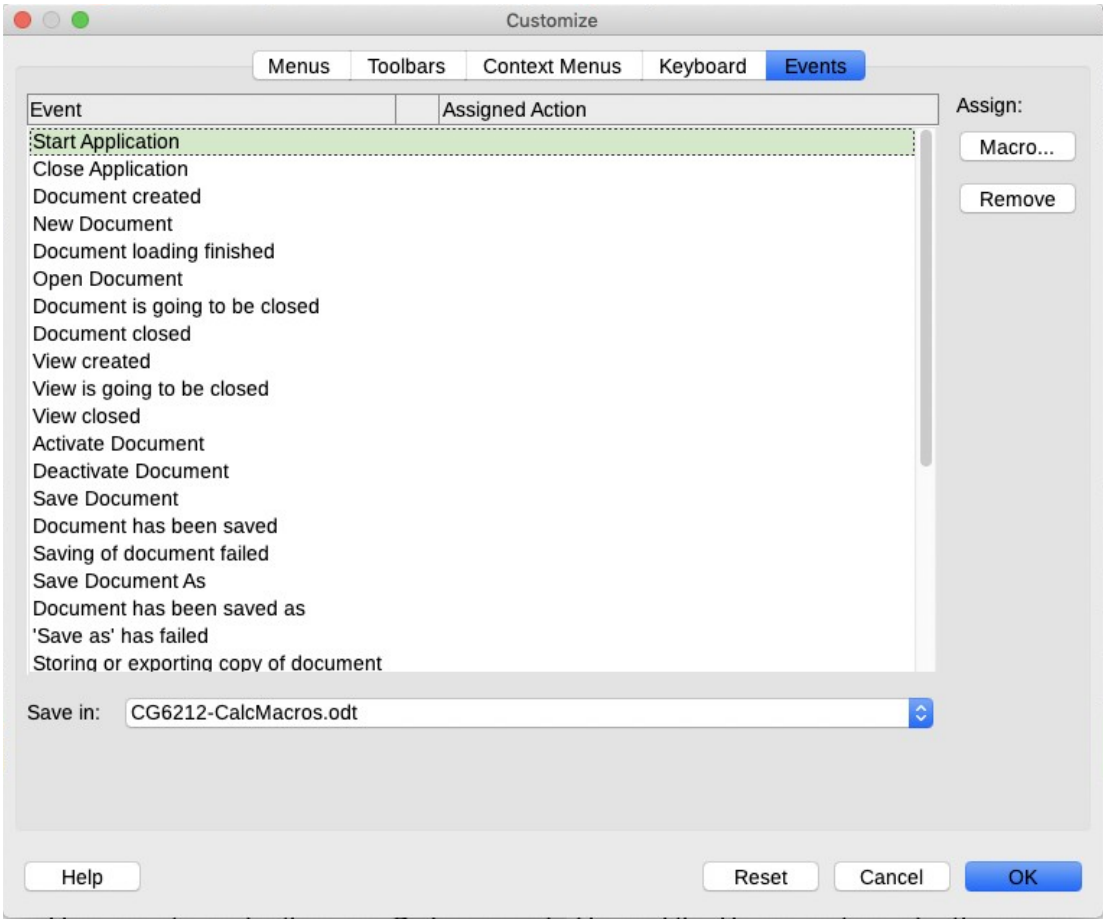

Figure 10: Events tab in Customize dialog

Using 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, for example:

- 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.
- 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
```

Overview of BeanShell, JavaScript, and Python macros

Many programmers may not be familiar with LibreOffice Basic, so LibreOffice supports macros written in three other languages that may be more familiar: BeanShell, JavaScript, and Python.

Macros are organized in the same way for all four scripting languages. The *LibreOffice Macros* container holds all the macros that are supplied in the LibreOffice installation. The *My Macros* library container holds your macros that are available to any of your LibreOffice documents. Each document can also contain your macros that are not available to any other document.

When you use the macro recording facility, LibreOffice creates the macro in LibreOffice Basic. To use the other available scripting languages, you must write the code yourself.

When you select to run a macro using **Tools > Macros > Run Macro**, LibreOffice displays the Macro Selector dialog. This dialog enables selection and running of any available macro, coded in any of the available languages (Figure 11).

When you select to edit a macro using **Tools > Macros > Edit Macros**, LibreOffice displays the LibreOffice Basic IDE. This dialog enables selection and editing of any available LibreOffice Basic macro, but not macros in other languages.

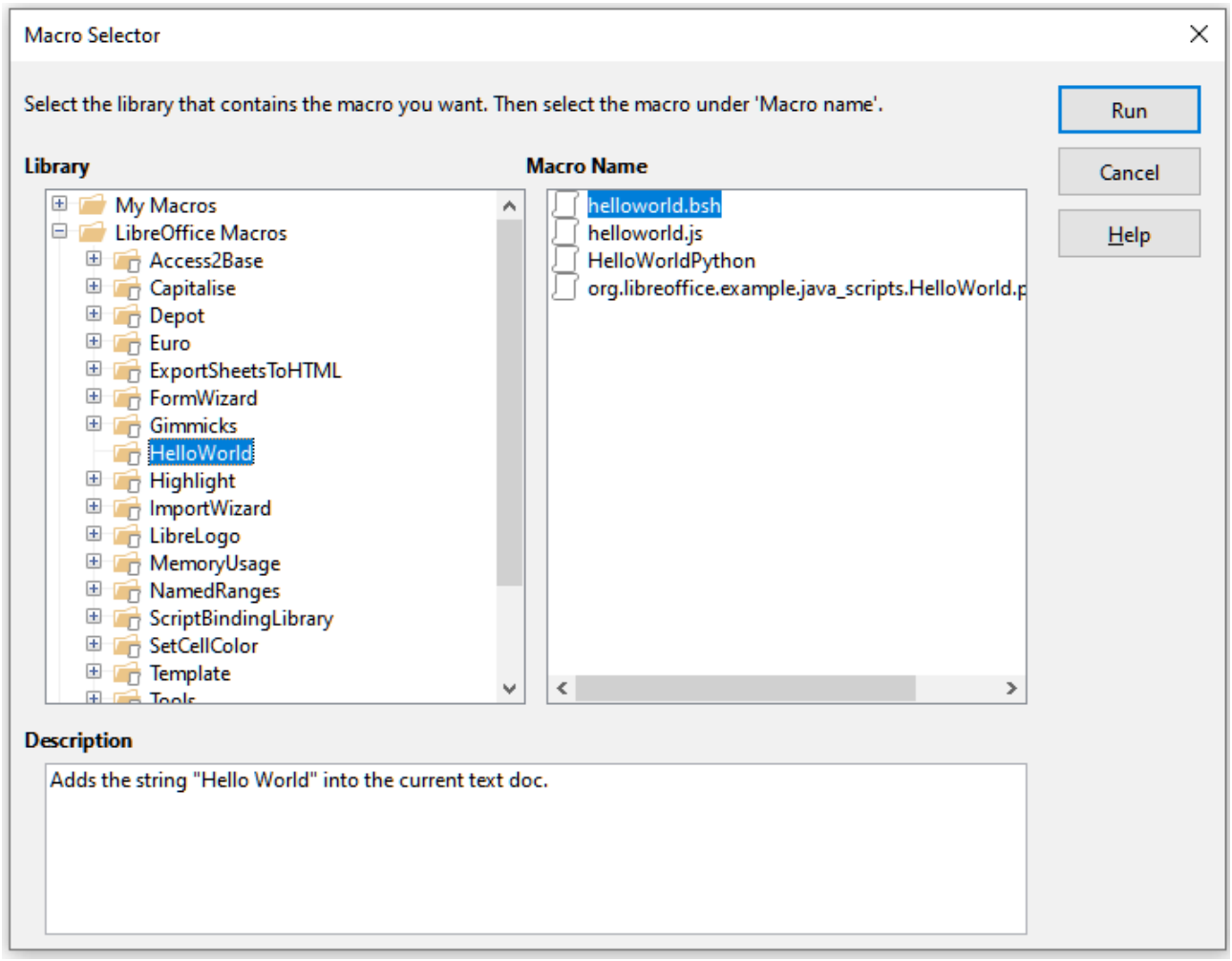

Figure 11: Macro Selector dialog

The component model used in LibreOffice is known as Universal Network Objects or UNO. LibreOffice macros in any scripting language use a UNO runtime application programming interface (API). The XSCRIPTCONTEXT interface is provided to macro scripts in all four languages, and provides a means of access to the various interfaces which they might need to perform some action on a document.

BeanShell macros

BeanShell is a Java-like scripting language that was first released in 1999.

When you select **Tools > Macros > Organize Macros > BeanShell**, LibreOffice displays the BeanShell Macros dialog (Figure 12).

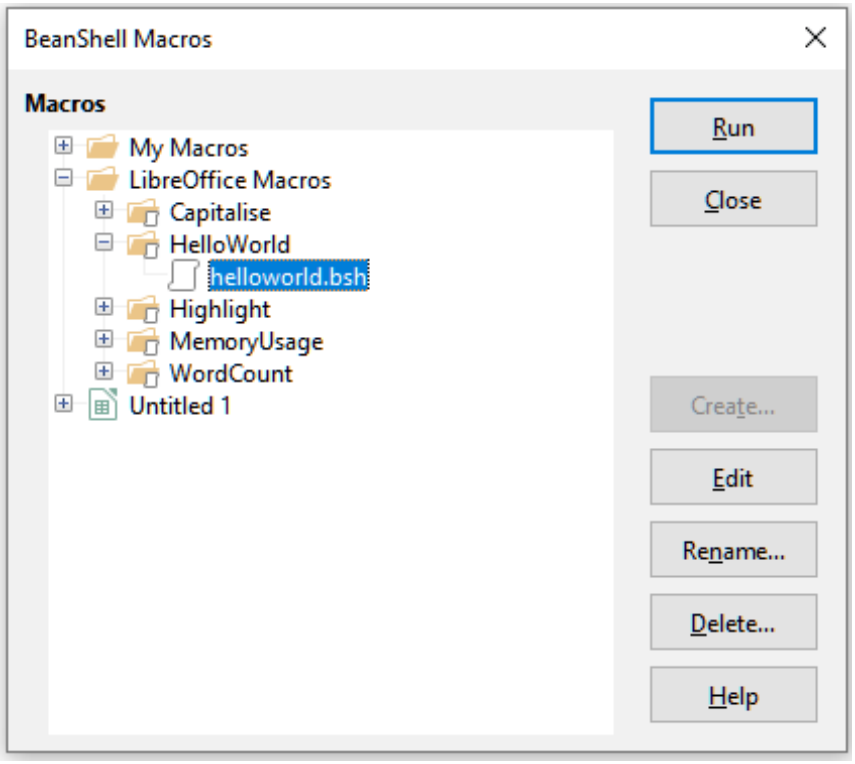

Figure 12: BeanShell Macros dialog

Click the **Edit** button on the BeanShell Macros dialog to access the BeanShell Debug Window (Figure 13).

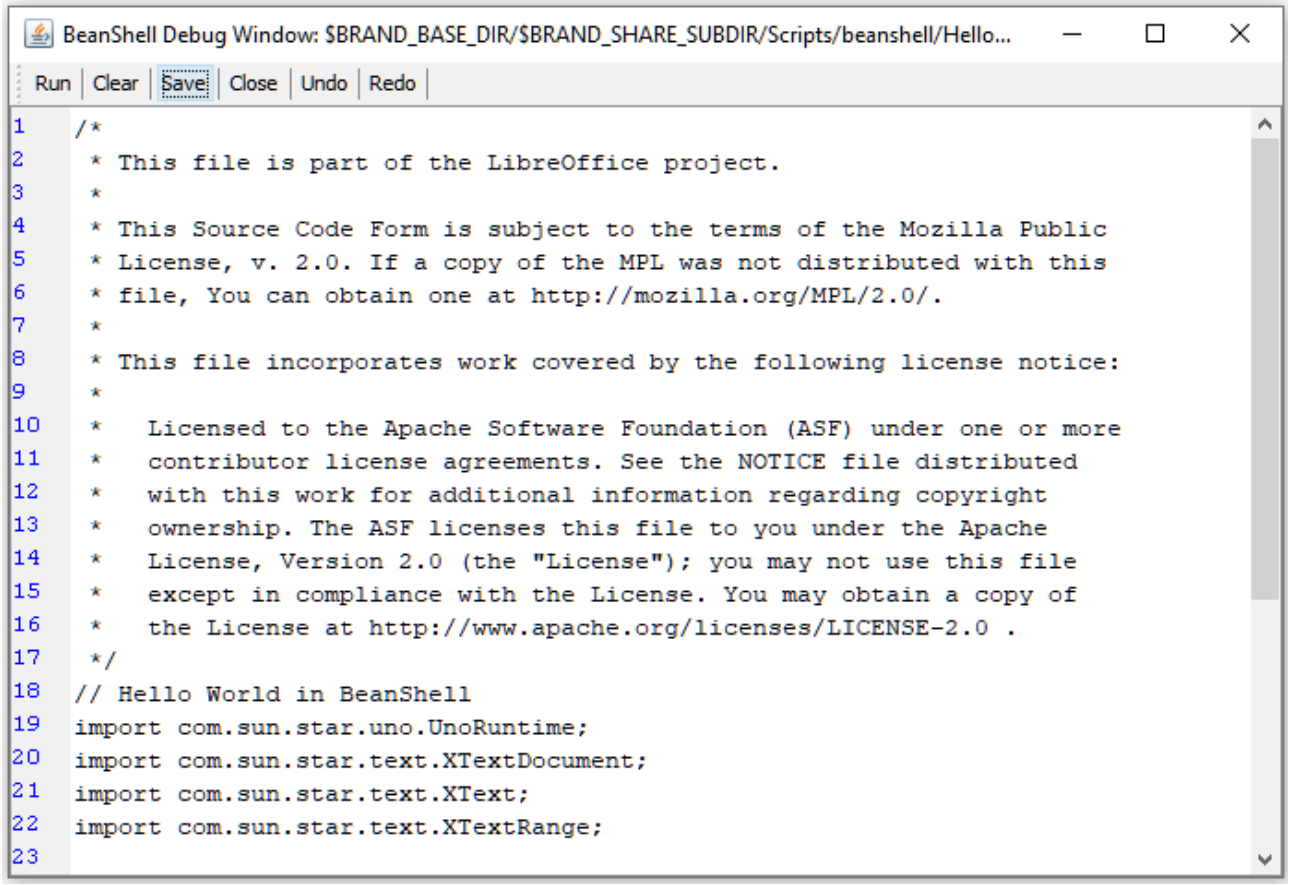

Figure 13: BeanShell Debug Window

Listing 7 is an example of a BeanShell macro that inserts the text “Hello World from BeanShell” in cell A1 of the active Calc spreadsheet.

Listing 7: Sample BeanShell macro

```
import com.sun.star.uno.UnoRuntime;
import com.sun.star.sheet.XSpreadsheetView;
import com.sun.star.text.XText;
model = XSCRIPTCONTEXT.getDocument();

controller = model.getCurrentController();

view = UnoRuntime.queryInterface(XSpreadsheetView.class, controller);

sheet = view.getActiveSheet();

cell = sheet.getCellByPosition(0, 0);

cellText = UnoRuntime.queryInterface(XText.class, cell);

textCursor = cellText.createTextCursor();

cellText.insertString(textCursor, "Hello World from BeanShell", true);

return 0;
```

JavaScript macros

JavaScript is a high-level scripting language that was first released in 1995.

When you select **Tools > Macros > Organize Macros > JavaScript**, LibreOffice displays the JavaScript Macros dialog (Figure 14).

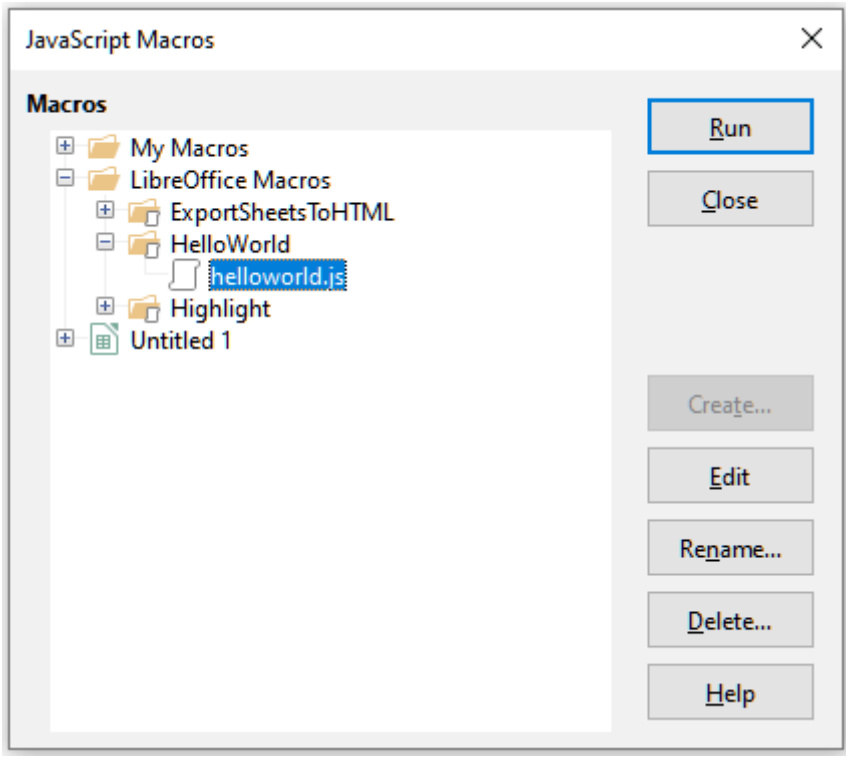

Figure 14: JavaScript Macros dialog

Click the **Edit** button on the JavaScript Macros dialog to access the Rhino JavaScript Debugger (Figure 15). Detailed instructions for using this tool can be found on Mozilla's website at https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/Debugger.

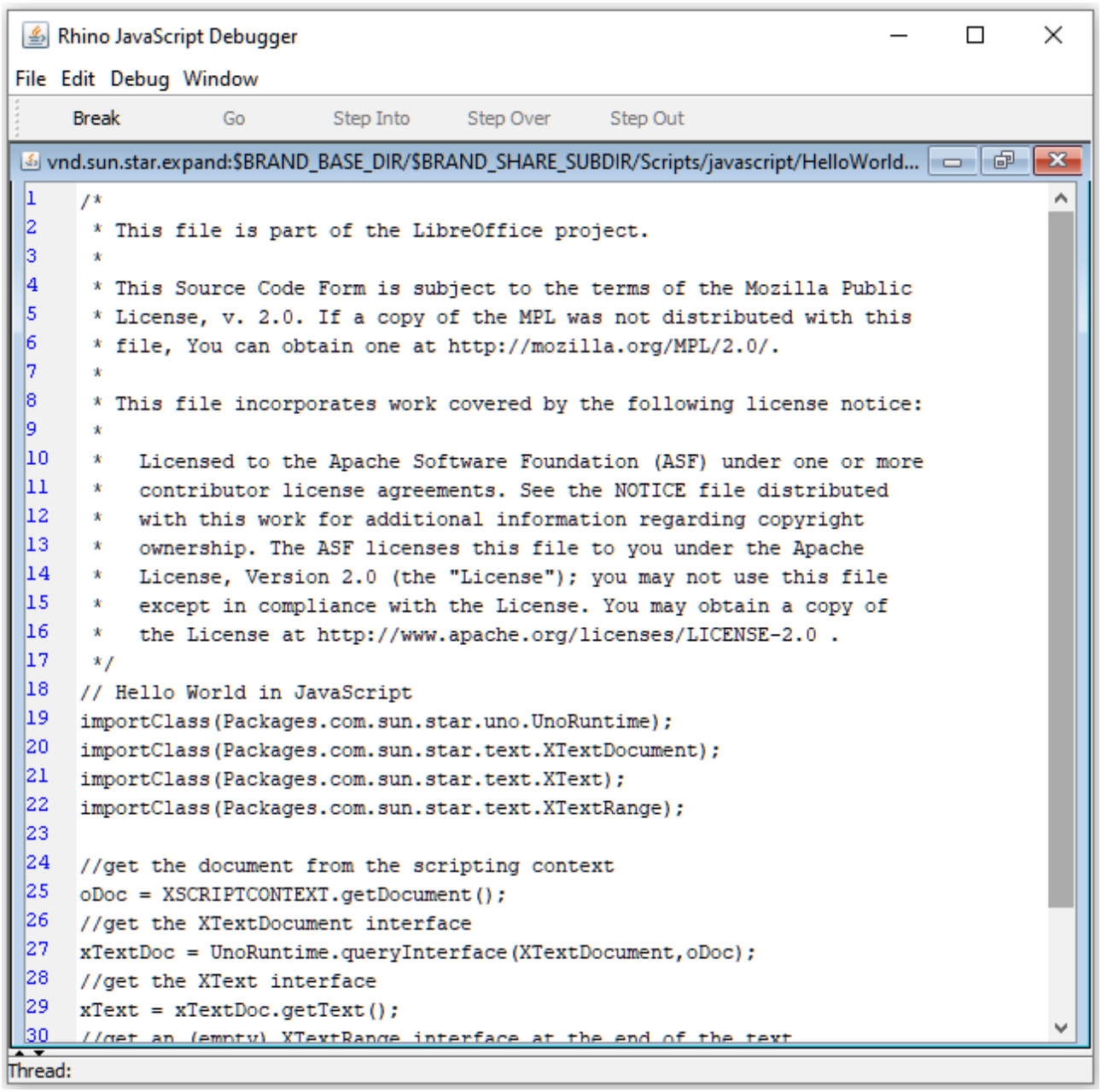

Figure 15: Rhino JavaScript Debugger

Listing 8 is an example of a JavaScript macro that inserts the text "Hello World from JavaScript" in cell A1 of the first sheet in a Calc spreadsheet.

Listing 8: Sample JavaScript macro

```
importClass(Packages.com.sun.star.uno.UnoRuntime);
importClass(Packages.com.sun.star.sheet.XSpreadsheetDocument);
importClass(Packages.com.sun.star.container.XIndexAccess);
importClass(Packages.com.sun.star.table.XCellRange);
importClass(Packages.com.sun.star.table.XCell);

documentRef = XSCRIPTCONTEXT.getDocument();
```

```
spreadsheetInterface = UnoRuntime.queryInterface(XSpreadsheetDocument,
documentRef);

allSheets = UnoRuntime.queryInterface(XIndexAccess,
spreadsheetInterface.getSheets());

theSheet = allSheets.getByIndex(0);

Cells = UnoRuntime.queryInterface(XCellRange,theSheet);

cellA1 = Cells.getCellByPosition(0,0);

theCell = UnoRuntime.queryInterface(XCell,cellA1);

theCell.setFormula("Hello World from JavaScript");
```

Python macros

Python is a high-level, general-purpose programming language that was first released in 1991.

When you select **Tools > Macros > Organize Macros > Python**, LibreOffice displays the Python Macros dialog (Figure 16).

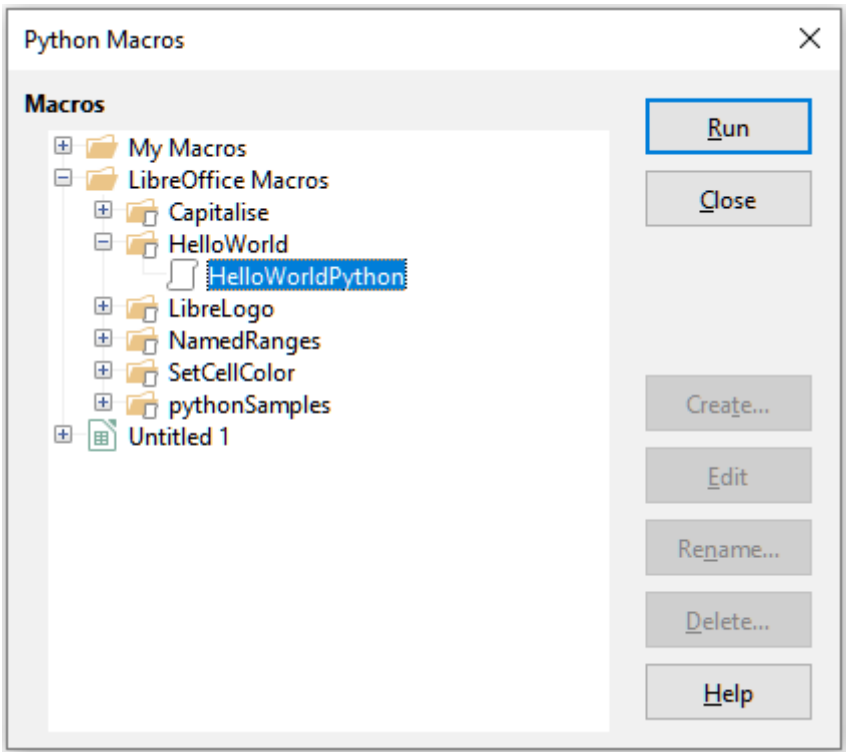

Figure 16: Python Macros dialog

Facilities to edit and debug Python scripts are not currently integrated into the standard LibreOffice user interface. However, you can edit Python scripts with your preferred text editor or an external IDE. The Alternative Python Script Organizer (APSO) extension eases the editing of Python scripts, in particular when embedded in a document. Using APSO you can configure your preferred source code editor, start the integrated Python shell and debug Python scripts. For more information search for Python in the LibreOffice Help system and visit the *Designing & Developing Python Applications* section of The Document Foundation's wiki, https://wiki.documentfoundation.org/Macros/Python_Design_Guide.

Listing 9 is an example of a Python macro that sets cell A1 of the first sheet in a Calc spreadsheet to the text “Hello World from Python”.

Listing 9: Sample Python macro

```
import uno

def HelloWorld():
    doc = XSCRIPTCONTEXT.getDocument()
    cell = doc.Sheets[0]['A1']
    cell.setString('Hello World from Python')
    return
```

Finding more information

Numerous resources are available that provide help with writing macros. Use **Help > LibreOffice Help,** or press the *F1* key, 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 **Basic** from this list.

Included material

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

Online resources

The following links and references contain information regarding macro programming:

https://wiki.documentfoundation.org/Macros

https://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)

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)

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.