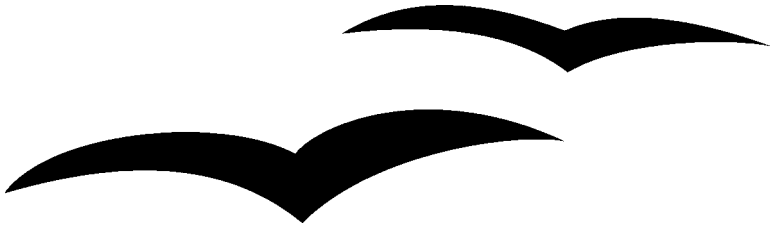
Getting Started with Macros:

Extract from "OpenOffice.org Macros Explained"



Title:	Getting Started with Macros: Extract from "OpenOffice.org Macros Explained"
Version:	1.0
First edition:	January 2005
First English edition:	January 2005

Contents

Overview

In OpenOffice.org, macros and dialogs are stored in documents and libraries. The included integrated development environment (IDE) is used to create and debug macros. This chapter introduces the basic concepts of starting the IDE and creating macros by showing the steps to produce a simple macro, which displays the text “Hello World” on the screen.

Copyright and trademark information

Feedback

Please direct any comments or suggestions about this document to: authors@user-faq.openoffice.org.

Acknowledgments

This chapter is reprinted, with the permission of the author and the publisher, from Chapter 1 of Andrew Pitonyak’s book *OpenOffice.org Macros Explained*, published by Hentzenwerke, 2004. More information about the book, including a table of contents, is available from http://www.hentzenwerke.com/catalog/oome.htm.

Modifications and updates

Version	Date	Description of Change
1.0	17 Jan 2005	First published edition in this format.

The OpenOffice.org macro language

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. Macros support commands that allow a variety of advanced functions, such as making decisions (for example, if the balance is less than zero, color it red; if not, color it black), looping (if the balance is greater than zero, subtract 10 from it), and even interacting with a person (asking the user for a number). Some of these commands are based on the BASIC programming language. (BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code.) It is common to assign a macro to a keystroke or toolbar icon so that it can be quickly started.

The OpenOffice.org macro language is very flexible, allowing automation of both simple and complex tasks. Although writing macros and learning about the inner workings of OpenOffice.org can be a lot of fun, it is not always the best approach. Macros are especially useful when you have to do a task the same way over and over again, or when you want to press a single button to do something that normally takes several steps. Once in a while you might write a macro to do something you can't otherwise do in OpenOffice.org, but in that case you should investigate thoroughly to be sure OOo cannot do it. For instance, a common request on some of the OpenOffice.org mailing lists is for a macro that removes empty paragraphs. This functionality is provided with AutoFormat (select **Tools > AutoCorrect/ AutoFormat > Options** tab). It is also possible to use regular expressions to search for and replace empty space. There is a time and a purpose for macros and a time for other solutions. This chapter will begin to prepare you for the times when a macro is the solution of choice.

Note OpenOffice.org is abbreviated as OOo. "OpenOffice.org Basic" is therefore abbreviated as "OOo Basic."

The OpenOffice.org macro language is based on the BASIC programming language. OOo Basic runs one line at a time. However, you usually need more than one line to get anything done, so you will typically write routines—also known as procedures—that consist of a number of lines that, when all are run, do a particular thing. For instance, you might write a routine that deletes a header from a file and inserts your preferred header. In OpenOffice.org, routines that are logically related are stored in a module. For example, a module might contain routines for finding common mistakes that require editing. Logically related modules are stored in a library, and libraries are stored in library containers. The OpenOffice.org application can act as a library container, as can any OOo document. Simply stated, the OpenOffice.org application and every OpenOffice.org document can contain libraries, modules, and macros.

Note A dialog is a window that appears on the screen, usually to request input or present information. Dialogs usually disappear after the requested input is entered. User-created dialogs are stored in dialog libraries in the same way that macros are stored in macro libraries. Each library can contain multiple dialogs. Library containers can store both macro and dialog libraries. See Chapter 17, "Dialogs and Controls" *in OpenOffice.org Macros Explained* for more about dialogs.

Storing a macro in a document library

Each OpenOffice.org document is a library container able to contain macros and dialogs. When a document contains the macros that it uses, possession of the document implies possession of the macros. This is a convenient distribution and storage method. Send the document to another person or location and the macros are still available and usable.

The traditional method of introducing a programming language is by writing a program that somehow outputs the message "Hello World." Entire Web sites exist with the sole purpose of showing "Hello World" programs in as many different programming languages as possible (for example, see http://www2.latech.edu/~acm/HelloWorld.shtml). Choosing not to break with tradition, my first macro shows a variation of "Hello World."

Step 1. Create a library

All OOo documents, regardless of document type, may contain macros. To add a macro to any OOo document, the document must be open for editing. Start by opening a new text document, which will be named "Untitled1"—assuming that no other untitled document is currently open. When a document is created, OpenOffice.org creates an empty library named Standard. The Standard library, however, remains empty until a new module is manually created. Use the Macro dialog to organize libraries and modules: select **Tools > Macros > Macro** (see Figure 1).

The "Macro from" list shows the available library containers; this includes every open document as well as the application library container. The application library container is named "soffice", but this is not shown in Figure 1. The document library containers are listed below the "soffice" container using the document's assigned name. Most library containers already have a library named Standard. Double-click a library container icon to toggle the display of the contained libraries. Double-click a library to toggle the display of the contained modules.

Note Before version 2.0, OOo displayed "My Macros" and "OpenOffice.org Macros" in the same list. The new dialogs are more intuitive while retaining a very similar look and feel. Support for editing and running macros in languages other than OOo Basic have also been added; see **Tools > Macros > Organize Macros > JavaScript**, for example.

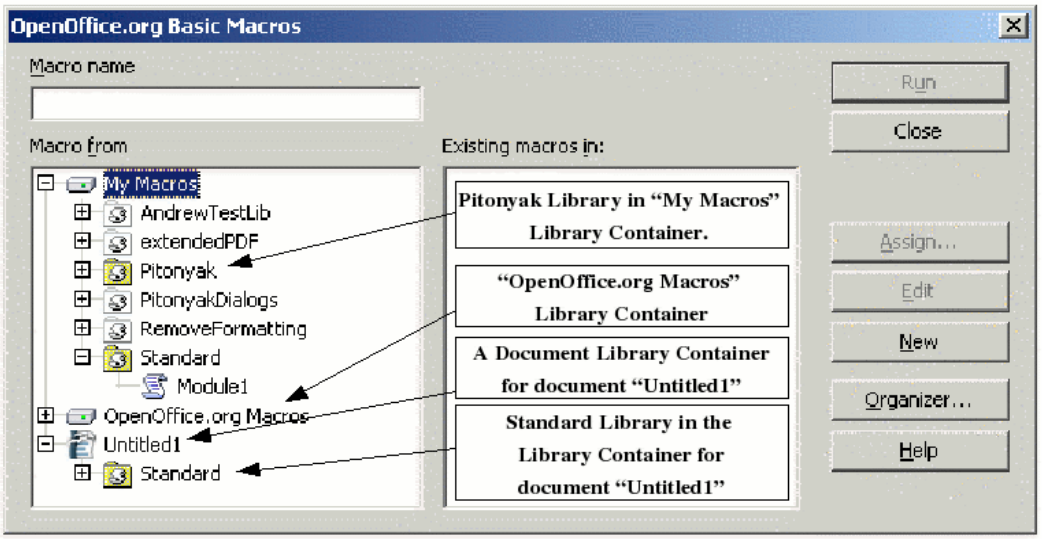


Figure 1. Use the Macro dialog to create new macros and organize libraries.

In Figure 1, the Standard library in the untitled document is highlighted. This library was automatically created when the new document was created. The document currently contains no modules—remember that macros are stored in modules. Although you could click the **New** button to create a new module, don't! The point of this section is to create a new library.

TIP Do not store your macros in the Standard library. Create a new library with a descriptive name and store your macros there. When a library is appended it will overwrite an existing library with the same name. If all of your libraries are named Standard, this prevents you from appending your libraries to other library containers.

Click the Organizer button to open the Macro Organizer dialog (see Figure 2). As with the Macro dialog, all of the library containers are listed. In other words, each document is listed, as is the "soffice" application library container. In Figure 2, the Standard library is highlighted in the document "Untitled1"; scroll down the list to find "Untitled1" if required. The Macro Organizer dialog is a tabbed dialog, and the tab in focus is Modules. As the name implies, the Modules tab deals with modules. Here's a description of the items in this dialog:

- The **New Module** button creates a new module in the selected library.
- The **New Dialog** button creates a new dialog in the selected library.
- The **Delete** button deletes the currently selected module; it's not available unless a module is selected.
- The **Edit** button opens the currently selected module for editing in the IDE; it's not available unless a module is selected.
- The **Close** button closes the Macro Organizer dialog.

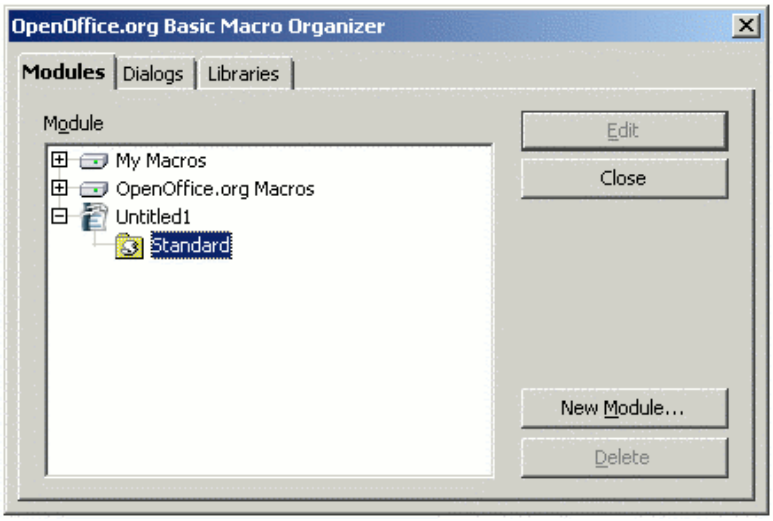


Figure 2. Use the Macro Organizer dialog to organize modules.

The purpose of this section is to create a meaningfully named library that is contained in the "Untitled1" document. Click the Libraries tab to deal with libraries (see Figure 3).

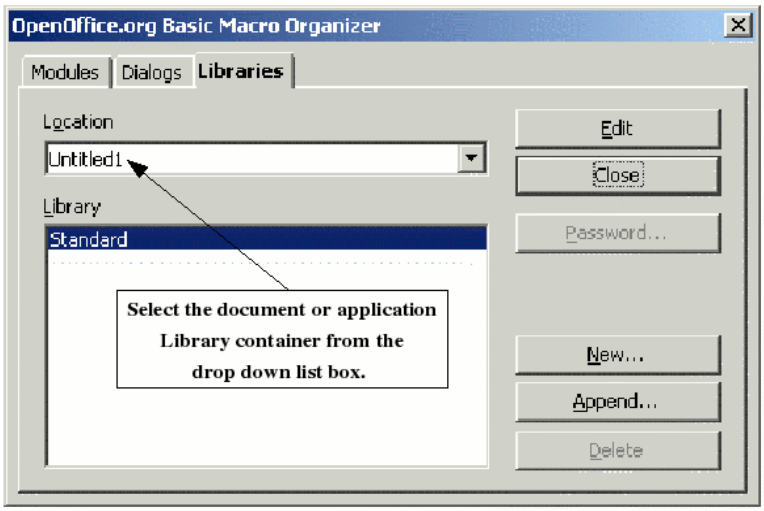


Figure 3. Use the Macro Organizer dialog to organize libraries.

When this portion of the dialog is displayed, the application library container (soffice) is selected in the Application/Document list. Select the "Untitled1" document so that the changes are made to the untitled document. The buttons displayed on the Libraries tab affect libraries, not modules. Here are their descriptions:

- The **New** button creates a new library in the selected document or application.
- The **Password** button allows you to assign or change the password for the selected library. Note that you cannot assign a password to the default library.
- The **Delete** button deletes the currently selected module; it's not available unless a module is selected.

- The **Append** button provides a mechanism for copying a library from another library container (document or application) to the library container selected in the Application/Document list. Library management is discussed later in this chapter.
- The **Edit** button opens the currently selected library for editing in the IDE.
- The **Close** button closes the Macro Organizer dialog.

Click the **New** button to create a new library (see Figure 4). Although the default name is "Library1," it is better to choose a meaningful name such as "MyFirstLibrary" or "TestLibrary." Click **OK** to create it.

Figure 4. Choose a meaningful name for the library.

The Macro Organizer now contains the newly created library in the Library list (see **Figure 5**).

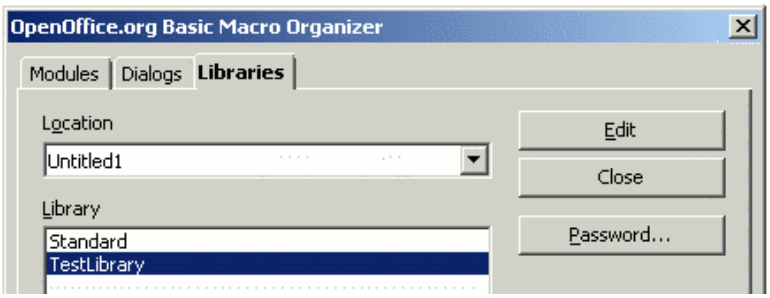


Figure 5. The document now contains the library named TestLibrary.

Step 2. Create a module

Macros are stored in a module, so the next step is to create a module in the newly created library. Assuming that the Macro Organizer (see Figure 3) is still open, select the Modules tab (see Figure 6).

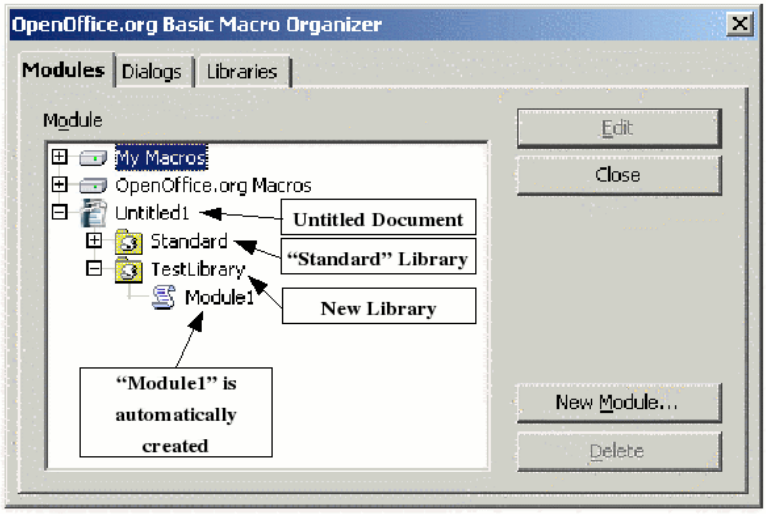


Figure 6. The TestLibrary contains one module named Module1.

The newly created TestLibrary is now displayed in the Macro Organizer. Select TestLibrary or any module contained in that library, and then click the New Module button to create a new module (see Figure 7). The default name is Module1; choose a more descriptive name for the module and click **OK** to create it.

Figure 7. Choose a meaningful module name.

TIP	Use descriptive module names to avoid confusion. This is important when moving between modules.

A common mistake is to highlight the wrong library container in either the Macro dialog or the Macro Organizer dialog. The most common mistake is to select a library or module in the application container (soffice) rather than a specific document. Find the document name in the list. The document name is determined by the title as set in the document's Properties dialog. Use **File > Properties** to open the document's Properties dialog. The title is set from the Description tab. If no title is set, the file name is used instead.

Note	Two documents with the same title in the document's Properties dialog use the same name in the Macro dialog, the Macro Organizer dialog, and the window title. This is confusing, so try to avoid it.

Step 3. Enter your first macro

If the Macro Organizer dialog is still open, you can highlight the newly created module and click the **Edit** button. This will open the Basic IDE. Another option is to use the Macro dialog. If the Macro Organizer dialog is open, click the **Close** button to open the Macro dialog. If the Macro Organizer dialog is not open, select **Tools** > **Macros** > **Macro** to open the Macro dialog (see Figure 8).

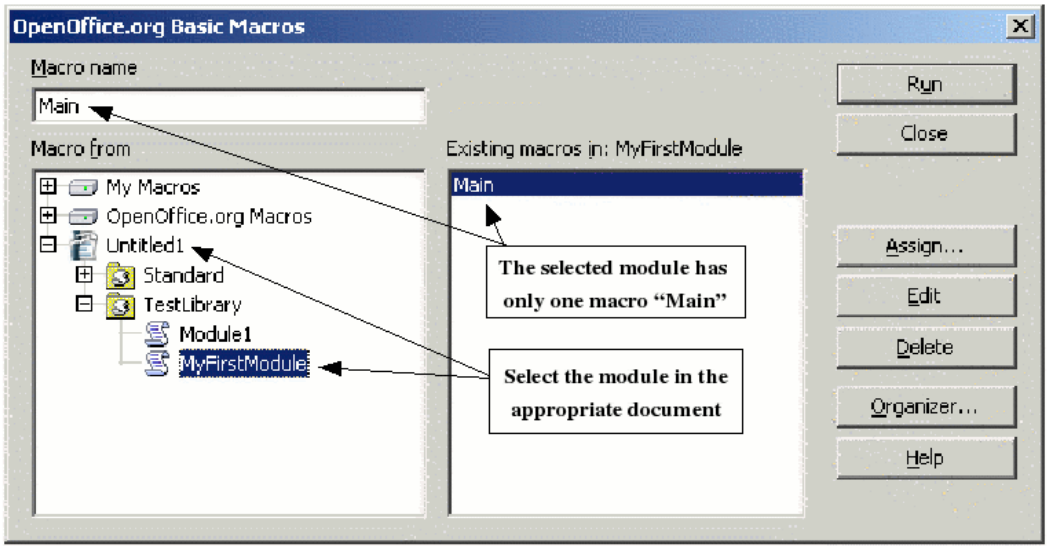


Figure 8. Select a specific macro.

Here's a description of the buttons in the Macro dialog:

- The **Run** button runs the selected macro. The macro is selected in the right-hand list, and its name also appears in the top-left input box labeled "Macro name."
- The **Close** button closes the Macro dialog.
- The **Assign** button associates a macro with a specific event. Assigning macros to events is discussed later.
- The **Edit** button opens the IDE and edits the selected macro.
- The **Delete** button deletes the selected macro. This button is present only if a module is selected. If a library or document is selected in the "Macro from" list, the **Delete** button changes to **New**. The **New** button creates a new macro in the selected library.
- The **Organizer** button opens the Macro Organizer dialog.
- The **Help** button opens the help system.

The purpose of the Macro dialog is to operate on individual macros. Select MyFirstModule and click the Edit button to open the Basic IDE; see Figure 9). One empty subroutine, Main, is automatically created when a module is created. The **New** button on the Macro dialog creates a second empty subroutine, Macro1. The IDE shown in Figure 9 was opened by clicking the library and then clicking the New button. Delete these subroutines and replace them with the code in Listing 1.

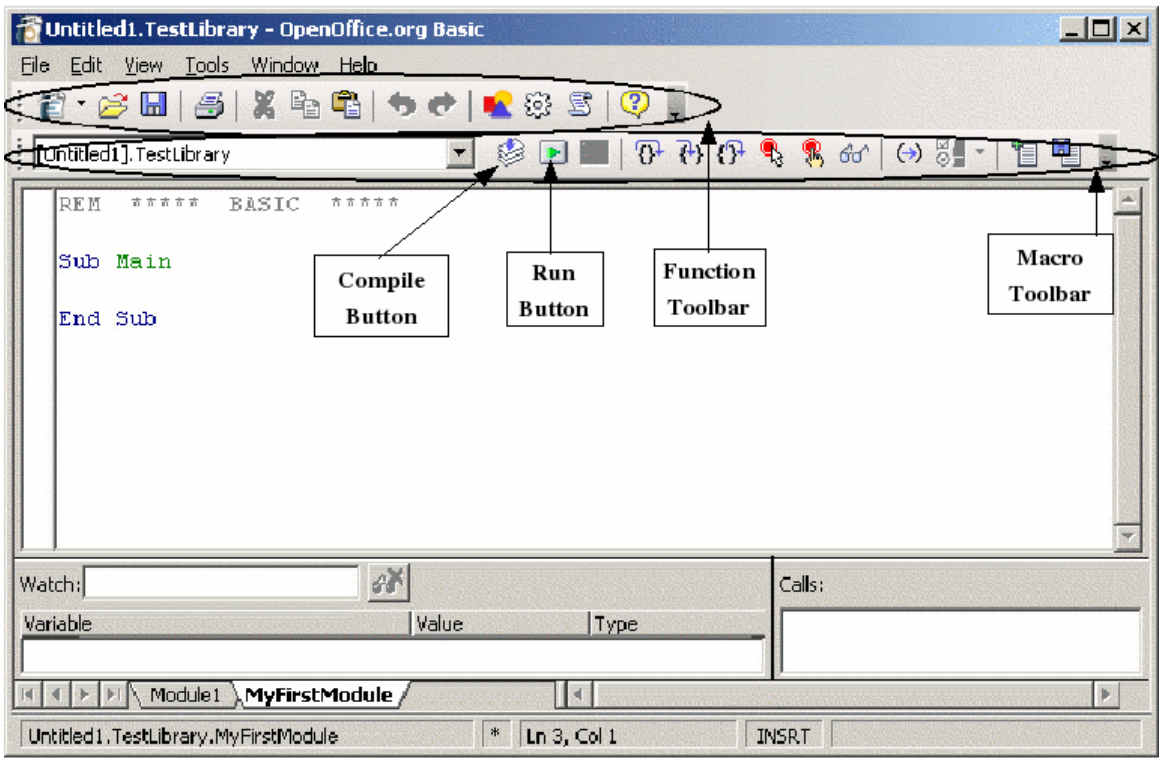


Figure 9. Develop macros in the IDE.

***Listing 1**. The "Hello World" routines are found in this chapter's source code files as SC01.sxw.*

```
Sub main
  HelloWorld2()
End Sub
Sub HelloWorld1
  Print "Hello World One"
End Sub
Sub HelloWorld2
  Print "Hello World Two"
End Sub
```

The IDE contains a Macro toolbar and a Function toolbar as labeled in Figure 9. (Most of the icons on the Macro toolbar are identified in Figure 13.) Rest your mouse cursor on a toolbar icon for a few seconds to read the text that appears; this provides a hint at what that icon does.

Click the **Compile** icon to check the macro for syntax errors. No message is displayed unless an error is found (see Figure 10). The Compile icon compiles only the current module.

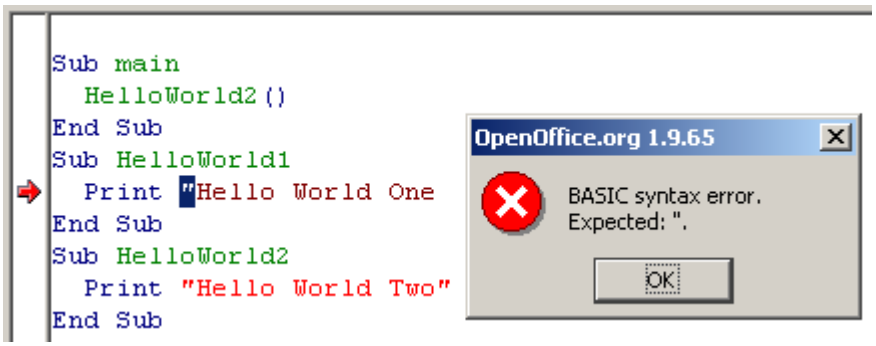


Figure 10. Click the Compile icon to find syntax errors such as a missing double quotation mark.

Modify the code in Listing 1 to demonstrate an error. Remove the second double quotation mark from the Print statement in HelloWorld1 (see Figure 10). Then click the **Compile** icon. A dialog displays a relevant error message for the first error encountered. The error message in Figure 10 indicates that a double quotation mark was expected but not found. The first double quotation character is highlighted, and a red arrow marks the line with the error. Click the **OK** button to close the error dialog, fix the line by adding a double quotation mark at the end, and then compile the code again.

Click the **Run** icon to run the first routine in the current module. It is not necessary to click the **Compile** icon first, because clicking the **Run** icon automatically compiles every module in the current library. Clicking the **Run** icon runs only the first routine in the module. For Listing 1, the Run icon runs the first subroutine, which is named “main.” The main subroutine calls the HelloWorld2 subroutine, displaying the dialog shown in Figure 11. Click **OK** to close the dialog, or click **Cancel** to to stop the macro.

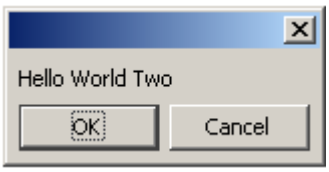


Figure 11. Click OK to close the dialog.

The **Run** icon always runs the first macro in the current module. As a result, a different approach is required to run HelloWorld1. To run HelloWorld1, you can use one of the following methods:

- Place HelloWorld1 first in the module and click the **Run** icon.
- Modify the main subroutine to call HelloWorld1 rather than HelloWorld2.
- Use the Macro dialog (shown in Figure 8) to run any routine in the module.
- Add a button to your OpenOffice.org document that calls HelloWorld1. This method is discussed later.

- Assign the macro to a keystroke. To do this, click **Tools > Configure** to open the Configuration dialog, and then select the *Keyboard* tab. Macro libraries are at the bottom of the Category list. You can also find this by clicking **Tools > Macros > Macro**, selecting the specific macro, and then clicking the **Assign** button to launch the Configuration window. Various tabs in this dialog allow you to assign the macro to execute as a menu item, from a keyboard key, a toolbar icon, or a system event.
- Add an icon to the toolbar that calls HelloWorld1.

To use the Macro dialog to run any subroutine in a module, follow these steps:

1) Select **Tools > Macros > Macro** to open the Macro dialog (see Figure 8).
2) Find the document that contains the module in the "Macro from" list.
3) Double-click a library to toggle the display of the contained modules.
4) Select the module to display the contained subroutines and functions in the "Existing macros in: <selected module name>" list.
5) Select the desired subroutine or function to run—for example, HelloWorld1.
6) Click the **Run** button to run the subroutine or function.

TIP When developing a subroutine, first place it in a module so you can quickly run it by clicking the **Run** icon. Another solution is to use the first subroutine to call another, as shown in Listing 1. This is faster than using the Macro dialog.

The code used in this chapter is available in an OpenOffice.org text document named SC01.sxw. Download and open this document. When a document containing macros is opened, OpenOffice.org issues a warning (see Figure 12). This warning is to help you avoid accidentally running a macro containing a virus. Although you can still manually run any macro using the Macro dialog, macro buttons in the document will not function. Click Run to fully enable the macro buttons added to SC01.sxw.

TIP You can configure a document to run a macro automatically when the document loads. This is how a macro virus spreads by using documents. If you don't expect a document to contain a macro, you should always click **Do Not Run**. This prevents any macro from running automatically when the document is loaded.

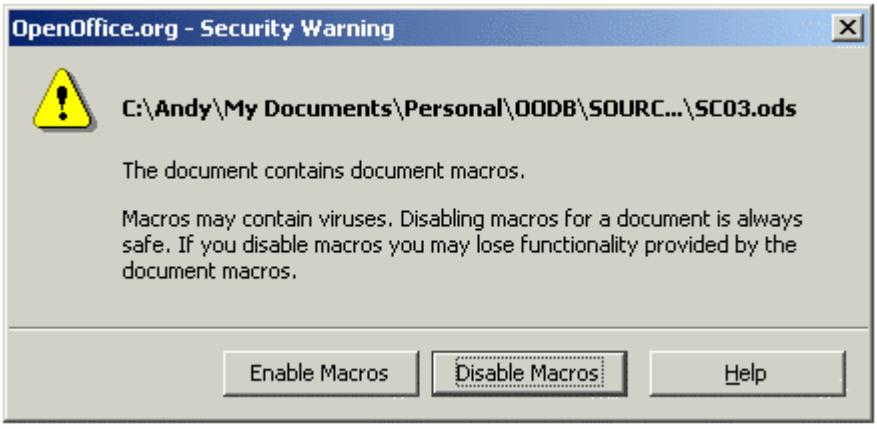


Figure 12. You just opened an OpenOffice.org document that contains a macro.

SC01.sxw contains three buttons: Main, Hello World 1, and Hello World 2. Each button is configured to run the corresponding subroutine when the button is clicked. The buttons do nothing if you click **Do Not Run** when the document is loaded.

It is possible to add a directory to the list of "secure paths." If you are certain that a path contains documents that you can safely trust not to contain macro viruses, you can check the box, and the path will be added to the list of secure paths. In the future, all documents loaded from the same location will be considered safe, and macros will always run.

Storing a macro in the application library

The OpenOffice.org application itself is a library container. This is an excellent place to store code and dialogs common to multiple documents. Version control is easier if macros are stored in one location. If five documents all contain the same macro, not only is storage space wasted, but if the macro changes, you must change the macro in five different documents.

To store a macro in the application libraries, use the same methods used for documents. The application-level container is named "soffice". The OpenOffice.org application includes multiple libraries. The Standard library includes a module named Module1, which contains one empty macro that does nothing. Use the Macro Organizer dialog in Figure 3 to add new libraries.

Caution

Uninstalling OpenOffice.org may delete libraries stored at the application level, so you should always keep a backup of your libraries. Reinstalling or installing a new version of OpenOffice.org may overwrite application-level libraries. Back up these libraries when you back up your documents. In most cases, the libraries that you created are still there, but the configuration files are new and do not reflect the new libraries. Therefore, it's usually possible to restore your libraries from the standard library location. For more information, see the section titled "Library management," later in this chapter.

Each application library is stored in its own directory. To determine where OpenOffice.org stores application libraries, select **Tools > Options**. In the Options dialog, expand the OpenOffice.org branch in the tree menu and select **Paths**. The Basic entry shows the locations of the external libraries.

Before installing a new version of OpenOffice.org, make a copy of all application-level libraries. If you install OOo into the same location, it overwrites the configuration file that tells OOo where your application-level libraries are. The libraries are usually still there but OOo does not know about them. To restore lost libraries, regardless of where they are located, use the Libraries tab on the Macro Organizer (see Figure 3). Verify that "soffice" is selected in the Application/Document list, and then click the **Append** button. Navigate to the directory containing the library that you want to add. Select the file script.xlb and click **Open**. Do this for each library that you want to restore. This method can also be used to add libraries stored in documents.

TIP Do not use the Standard library if you think you'll ever want to append your library to another location. Store all of your modules in libraries with meaningful names that you create. The Standard library is special, and you cannot delete it or overwrite it.

To practice adding a macro to the application-level library, open the Macro Organizer. Verify that the "soffice" library container is the current container. Click the **New Module** button to add new modules to the application-level libraries. To add new libraries, select the Libraries tab. Verify that "soffice" is selected in the Application/Document list, and then click the **New** button.

Libraries stored in documents may be appended to the application library container. When a library is appended, it overwrites an existing library with the same name. It is, therefore, a good idea to create meaningful library names to hold macros. This limits problems moving macros between library containers.

The Integrated Development Environment

An integrated development environment (IDE) is a set of programming tools used to facilitate the creation of software. OpenOffice.org includes a very capable IDE with tools that run, edit, and find errors in your macros. It is worth the time to become familiar with its features. Figure 13 shows the IDE with captions added for many of the icons and display areas. The central display area where macro code is listed is the editor window. Many of the features, such as Stop, Breakpoint, Single Step, and the Watch pane serve as a simple yet effective debugger for macro code.

This section provides a quick overview of the standard functions of the IDE. Do not be surprised if you don't fully understand how to use them all at this point. You will become very familiar with these functions as you work through the examples. The first set of functions are used for debugging, and the ones described near the end of this section support the organization and management of objects in your macro programs, libraries, and documents. Following are the icon descriptions.

The **Compile** icon compiles and performs a syntax check of *only* the current module. The Compile icon is useful if you don't want to run the macro but you want to verify that it's syntactically correct. No message is displayed unless an error is found (see Figure 10). When an error is found, a dialog appears, indicating the error. An arrow in the Breakpoint column marks the line with the error, and the portion of the code that caused the error is highlighted. Click the **OK** button to close the error dialog.

Note The process of compiling translates OOo macros into machine language, which the computer can understand and run.

The **Run** icon compiles all of the modules in the current library and then runs the first subroutine or function in the current module. This is different from the Compile icon, which compiles *only* the current module.

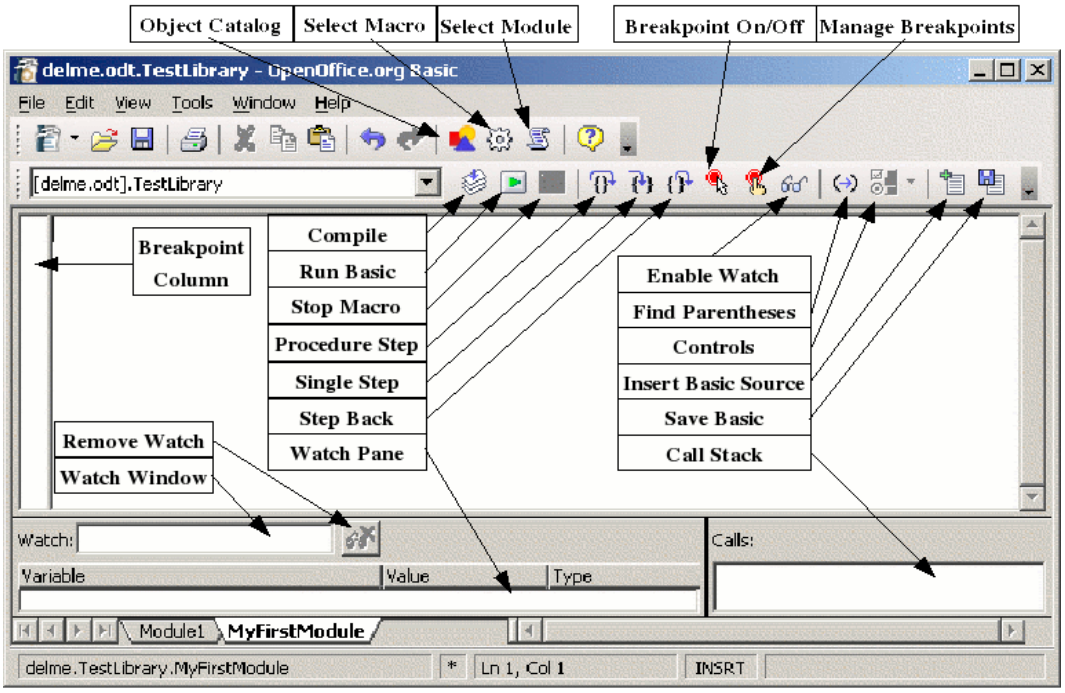


Figure 13. The IDE included with OpenOffice.org is very capable.

The **Stop** icon stops a running macro. When you click this icon, you can't resume the macro; you must start it again, from the beginning. The Stop icon is enabled only while a macro is running. When enabled, the Stop icon resembles a traffic stop sign.

The **Procedure Step** icon runs the current statement. If the macro is not yet running, the first routine in the module is started and marked as the current statement. The current statement has an arrow in the Breakpoint column, and the cursor is moved to that line. If, however, the macro is already running, the current statement runs and the next runnable statement is marked as current. The Procedure Step icon treats calls to other routines as a single statement and does not step into them. Notice that the icon has an arrow that curves *around* the curly brackets that represent a subroutine or function call.

The **Single Step** icon runs the current statement. The behavior is the same as the Procedure Step icon except that subroutines and functions are not treated as a single statement. Each statement in the called routine is considered a statement. Subroutines and functions are stepped into, marking the called subroutine or function definition as the current statement. Notice that the icon contains an arrow that points *into* the curly brackets that represent a subroutine or function call.

The **Step Back** icon runs the macro to the end of the current routine and then steps out of it. The effect is the same as repeatedly clicking the Procedure Step icon until the last statement in the current routine (End Sub or End Function) is current, and then clicking Procedure Step one more time to step out of the routine. The statement following the call to the current routine becomes the current statement. If you accidentally click Single Step rather than Procedure Step, click the Step Back icon once. Notice that the icon contains an arrow that *leaves* the curly brackets that represent a subroutine or function call.

The **Breakpoint** icon sets a breakpoint at the statement containing the cursor. A red stop sign marks the line in the Breakpoint column. Double-click the Breakpoint column to toggle a breakpoint at that statement. Right-click a breakpoint in the Breakpoint column to activate or deactivate it.

The **Manage Breakpoints** icon loads the Manage Breakpoints dialog (see Figure 14).

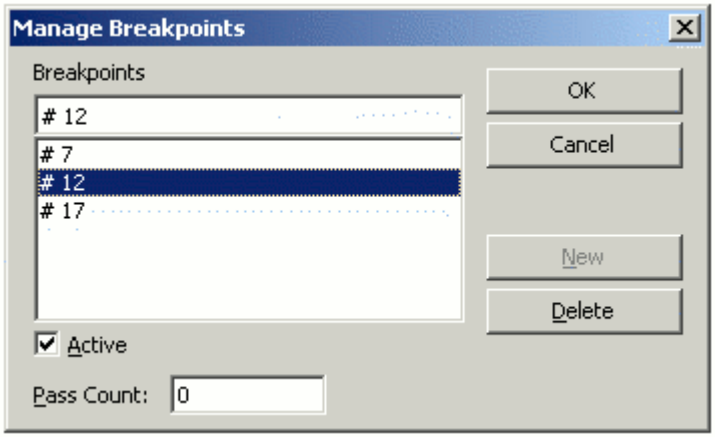


Figure 14. Manually edit and configure breakpoints.

The **Add Watch** icon assumes that the current word (the word that contains the icon) is a variable and adds this variable name to the Watch pane.

The **Object Catalog** icon opens the Objects window (see **Figure 15**), where you can browse all of the currently available library containers. Use this window to see which libraries, modules, and subroutines are available. Double-click a subroutine to load it into the IDE. The functionality is similar to the Navigator in an OOo Writer document. You must save a file before its modules are available in the Object Catalog.

TIP	Leave the Objects window open and use it as a navigator to quickly jump between modules, libraries, or even subroutines in the same module.

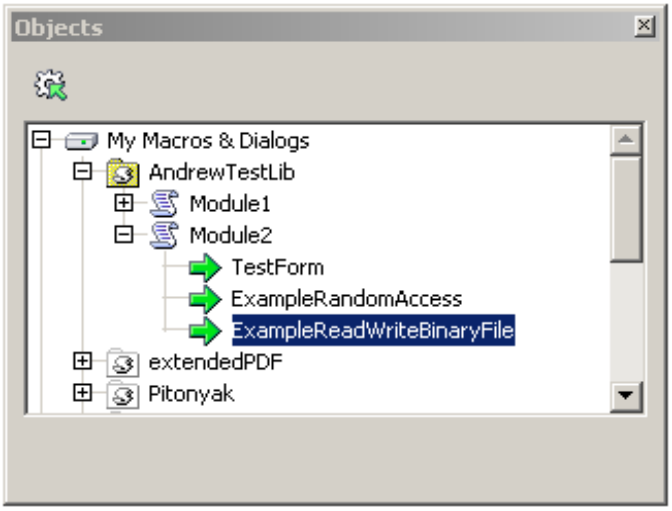


Figure 15. You can browse the available libraries and modules in the Objects window.

The **Macros** icon loads the Macro dialog. Selecting Tools | Macros | Macro also loads the Macro dialog.

The **Modules** icon loads the Macro Organizer dialog. This icon has the same effect as clicking the Organizer button in the Macro dialog (see Figure 2 and Figure 3).

Select or place the cursor directly to the left of a parenthesis, and then click the **Find Parentheses** icon to find the matching parentheses. When the IDE matches parentheses, it selects the matching parentheses and everything that they enclose.

To open the Controls window, click the **Controls** icon while editing a dialog. (For more information about controls, see Chapter 17, “Dialogs and Controls” in *OpenOffice.org Macros Explained*)

To create a dialog for editing, click the **Modules** icon to load the Macro Organizer dialog. Click the New Dialog button to create a new dialog. See Chapter 17, “Dialogs and Controls,” in *OpenOffice.org Macros Explained* for more about using and creating dialogs in macros.

The last two icons, **Insert Source Text** and **Save As Source**, are used to insert text stored in an external source file into the current module, and to save the current module to an external text file. This is an excellent way to create a backup of a macro or to create a text file that can be easily sent to another person. This is different from the **Disk** icon, which is used to save the entire library or document that contains the module.

Using breakpoints

If you set a breakpoint in the code, the macro will stop running at that point. You can then inspect variables, continue running the macro, or single-step the macro. If a macro fails and you don't know why, single-stepping (running one statement at a time) allows you to watch a macro in action. When the macro fails, you'll know how it got there. If a large number of statements run before the problem occurs, it may not be feasible to run one statement at a time, so you can set a breakpoint at or near the line that causes the problem. The program stops running at that point and you can single-step the macro and watch the behavior.

The **Breakpoint** icon sets a breakpoint at the statement containing the cursor. A red stop sign marks the line in the Breakpoint column. Double-click in the Breakpoint column to toggle a breakpoint at that statement. Right-click a breakpoint in the Breakpoint column to activate or deactivate it.

The **Manage Breakpoints** icon loads the Manage Breakpoints dialog (see Figure 14). All of the active breakpoints in the current IDE dialog appear in the lower list. To add a breakpoint, enter a line number in the entry field at the top and then click **New**. To delete a breakpoint, select a breakpoint in the list and click the **Delete** button. Clear the **Active** check box to disable the highlighted breakpoint without deleting it. The *Pass Count* input box indicates the number of times a breakpoint must be reached before it is considered active. If the pass count is four (4), then the fourth time that the statement containing the breakpoint is to be run, it will stop rather than run. This is extremely useful when a portion of the macro does not fail until it has been called multiple times.

There are two things that cause a breakpoint to be ignored: a pass count that is not zero, and explicitly marking the breakpoint as "not active" in the Manage Breakpoints dialog. Every breakpoint has a pass count that is decremented toward zero when it is reached. If the result of decrementing is zero, the breakpoint becomes active and stays active because the pass count stays at zero thereafter. The pass count is not restored to its original value when the macro is finished or restarted.

It is easy to monitor the value of variables from the IDE while a routine is running. Place the cursor next to or in any word in the Edit window and click the Add Watch icon to add the word to the Watch pane. The Watch pane displays the value of variables that are currently in scope. The text "<Out of Scope>" is displayed for variables that are not available. Another way to add variables to the Watch pane is to type the name into the Watch window and press Enter. To delete a name from the Watch pane, select it in the Watch pane or type the name into the Watch window and click the **Remove Watch** icon. Click a name in the Watch pane to place its name in the Watch window.

Note	A variable that is in scope is currently available or visible. For example, if the variable "j" is defined inside HelloWorld1, it is not visible (in scope) inside HelloWorld2. This is discussed later.

Library management

This section deals with creating, transferring, and renaming libraries and modules. When considering library management, it is important to first understand some basics that have already been discussed:

- A library container contains zero or more libraries.
- Each library contains zero or more modules and dialogs.
- Each module contains zero or more macros.
- The application is a library container named “soffice”. Libraries stored in the application are globally available to all macros.
- Every document is a library container.
- The library named Standard is special; it always exists and cannot be overwritten. I recommend against using the Standard library.
- Always give meaningful names to the libraries and modules that you create. For example, Library1 and Module4 are not meaningful names, although AXONInvoiceForm1 might be more descriptive and helpful.

How libraries are stored

OpenOffice.org libraries are stored as XML files that are easily editable using any text editor. In other words, it is easy for you to poke around and damage your files. Although manually editing your external libraries is generally considered foolish, I have had at least one instance where this was required, because OOo was unable to load a module that contained a syntax error.

TIP Manually editing OOo files is best left to advanced users. Beginning users may want to quickly skim through this material or skip to the next section.

Application libraries

Each application library is stored in a single directory, and each module and dialog is contained in a single file. The Options dialog (Tools | Options | OpenOffice.org | Paths) contains an entry that identifies where libraries are located. The global libraries that are included with OpenOffice.org are stored in a shared basic directory under the directory in which OOo is installed. Examples:

```
C:\Program Files\OpenOffice.1.1.1\share\basic  'A Windows installation
/usr/local/OpenOffice.org1.1.1/share/basic     'A Linux installation
```

The libraries that you create are stored in different directories. On my Windows computer, I have a single-user installation, and on my Linux computer I have a multiple-person network installation. The choices that you make while installing OOo affect the location of your personal libraries. Here are two examples:

```
C:\Program Files\OpenOffice.1.1.1\user\basic  'Windows user files
/home/andy/OpenOffice.org1.1.1/user/basic     'Linux user files
```

Listing the shared directory shows one file for each application library that is included with OOo. The user directory, however, is a bit more interesting (see Table 1).

Table 1. Files and some directories in my user/basic directory.

Entry	Description
dialog.xlc	XML file that references every dialog file known to this user in OpenOffice.org.
script.xlc	XML file that references every library file known to this user in OpenOffice.org.
Standard	Directory containing the Standard library.
Pitonyak	Directory containing a library with code that I created.
PitonyakDialogs	Directory containing a library with some code and a dialog.

Note Table 1 references the directories Pitonyak and PitonyakDialogs. The Pitonyak library and the PitonyakDialogs library are not related; their names are similar because I lacked creativity and good sense when I named them. It is not true that the library PitonyakDialogs contains the dialogs for the Pitonyak library.

The files dialog.xlc and script.xlc contain a reference to all of the dialogs and libraries that are known to OOo. The visible libraries—as seen in the Macro dialog and the Macro Organizer dialog (see Figure 16)—are built from the files dialog.xlc and script.xlc. If these two files are overwritten, OOo will not know about your personal libraries even if they exist.

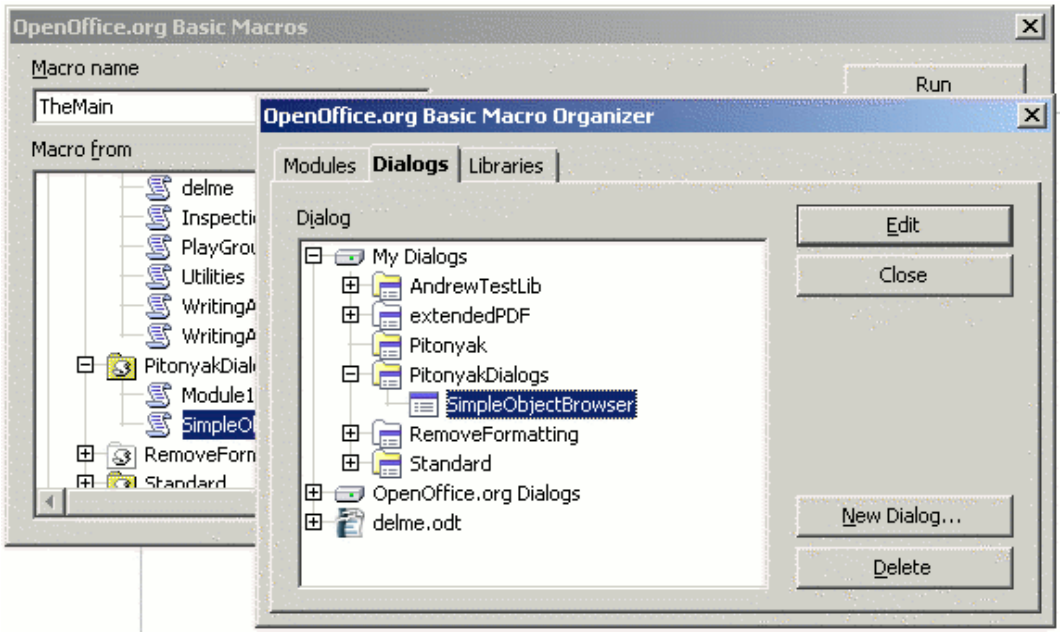


Figure 16. The Macro dialog and the Macro Organizer dialog show available libraries and modules.

The Macro Organizer dialog shows that the PitonyakDialogs library contains two code modules and one dialog. **Table 2** contains a listing of the files in the PitonyakDialogs directory. Notice that each module and dialog in a library has a corresponding file.

Table 2. Files in the PitonyakDialogs library directory.

File	Description
dialog.xlb	References the dialogs contained in this library.
script.xlb	References the modules contained in this library.
Module1.xba	BASIC code in the module named Module1.
SimpleObjectBrowserCode.xba	BASIC code in the module named SimpleObjectBrowserCode.
SimpleObjectBrowser.xdl	A dialog in the module named SimpleObjectBrowser.

The files dialog.xlc and script.xlc in Table 1 reference the files dialog.xlb and script.xlb in Table 2. In general, none of these files should be manually modified, but in an emergency, they may be modified by hand to correct certain types of errors.

Document libraries

An OpenOffice.org document, when saved to disk, is stored in the standard ZIP format. Any program that can view and extract ZIP files can be used to inspect an OOo document—however, some programs will require you to change the file extension to end with ZIP.

After unzipping an OOo document, you will find files that contain the primary content, styles, and settings. The extracted document also contains three directories. The META-INF directory references all of the other files, embedded pictures, code libraries, and dialogs. The Dialogs directory contains all of the embedded dialogs, and the Basic directory contains all of the embedded libraries. Notice that libraries contained in the "soffice" application-level container are stored in a slightly different configuration than the libraries contained in a document.

As an experiment, I took a document that contained numerous controls that called a specific library. I unzipped the document and then used a text-search tool to find all references to a specific library named CH03. After manually changing every occurrence of the text "CH03" to "CH04", I zipped the directory back into a single file, and OOo was able to read and use the file. I successfully changed the name of a contained library and every reference to the controls by editing the XML definitions.

TIP The point of this section is that, in an emergency, you can manually inspect a document's XML and potentially fix problems. This is usually NOT the best way to change the name of a document's libraries.

Using the Macro Organizer

The Macro Organizer (**Tools > Macros > Macro > Organizer**) is able to satisfy most users' needs in regards to organizing modules and libraries. The *Modules* tab of the Macro Organizer dialog (see Figure 16) provides the capability to create and delete modules. The Macro Organizer dialog also has a Libraries tab (see Figure 17) used to create and delete libraries. The *Libraries* tab contains a drop-down box at the top that is used to select the library container. In other words, you can select a specific open document or the application library container named "soffice."

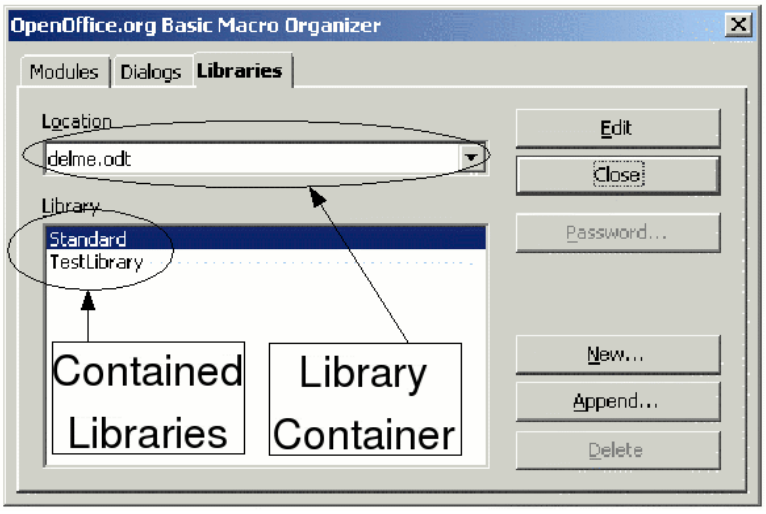


Figure 17. Use the Macro Organizer to create, add, and delete libraries.

Note A document's name is the file name, unless the document title is set in the document's Properties dialog (**File > Properties > Description**). If the Title property is set, it is used as the document name in the window title, the Macro dialog, and the Macro Organizer dialog.

Renaming modules and libraries

You can change the name of a module or library from the Macro Organizer dialog. Module names are changed from the Modules tab, and library names are changed from the Libraries tab. When you change a library or module name, it doesn't change anything that references the contained macros. For example, if I write down your telephone number in my Palm Pilot and you change your telephone number, my Palm Pilot is not automatically updated. So, what might call a macro?

- When controls are embedded in a document or a dialog, they frequently use macros as event handlers.
- Macros call dialogs that are contained in libraries.
- Macros may be called from programs outside of OpenOffice.org.

Caution  When you rename a module or a library, references to the contained macros are not updated.

Renaming a library or module isn't a bad thing to do; just remember that things that reference the modules and libraries won't be updated. If nothing is referencing your code, feel free to change the module and library names. You can rename libraries and modules by using the Macro Organizer; the procedure is the same for both:

1) Find the library or module in the appropriate tab of the Macro Organizer (see Figure 16 and Figure 17).
2) Select the library or module.
3) Wait a moment and click on the library or module. The cursor should appear to edit the library or module name. I have found this to be a bit sensitive, and sometimes I must purposely single-click a few times. Do not accidentally double-click, because this opens the library or module contents for editing.
4) Type a new name for the library or module and press the *Enter* key.

I had a large document that contained numerous buttons. The buttons called macros in a library and I had to change the name of the library. Unfortunately, after I changed the name of the library, the buttons still pointed to the original library, which no longer existed. Feeling particularly daring, I unzipped the document into a temporary directory (remember that an OOo document file is really a ZIP file containing numerous files that, as a whole, are the document). I then used my favorite text editor to load each file and I changed the old library name to the new library name. When I was finished, I zipped all of the files and directories back into a single ZIP file and I had successfully changed all of the references.

Caution 

Manually editing an OOo document file by unzipping all of the contained files and directories and then zipping them back is an error-prone process. If you do it wrong, the document will stop working. In other words, keep a copy of the original file.

Adding libraries

The **Append** button (see Figure 17) in the Macro Organizer dialog opens the Append Libraries dialog, which is really a file-selection dialog. This dialog is used to select the file that contains the library to append. To add a library contained in a document, start by selecting the document. The **Open** button on the file-selection Append Libraries dialog opens the library-selection Append Libraries dialog (see Figure 18). Use the library-selection Append Libraries dialog to view the libraries contained in the selected document and select the libraries you want to append.

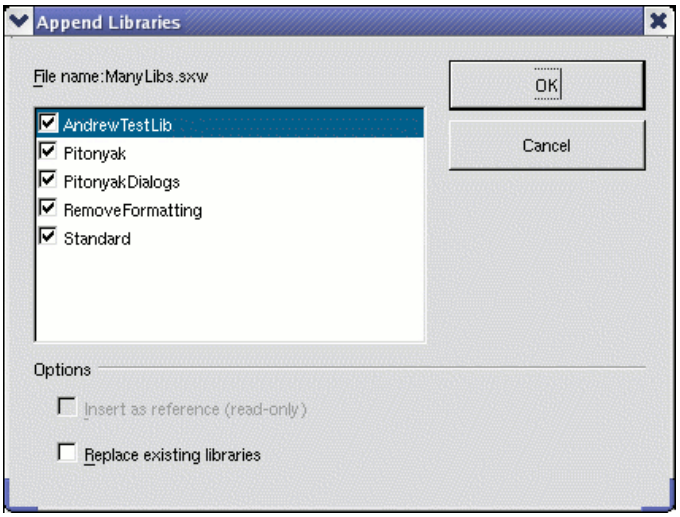


Figure 18. Use the Append Libraries dialog to select the libraries to add.

The library-selection Append Libraries dialog in Figure 18 allows you to append as many libraries as you like. Existing libraries are not overwritten unless the “Replace existing libraries” box is checked. The “Insert as reference” check box is available only while appending libraries that are not contained in a document. Click **OK** to append the selected libraries.

TIP It is not possible to overwrite the Standard library. I recommend against using the Standard library because you can't append it to another document or the application.

Libraries that are not contained in a document are stored in individual directories. To append a library that is not stored in a document, open the file-selection Append Libraries dialog (see Figure 19) and select the directory that contains the library files. It doesn't matter where the library files are stored. The files may be on a floppy disk as a backup, or they may be from the same directory used by OOo for application-level libraries. When I install a new version of OpenOffice.org, I append the libraries from my previous OOo installation.

While appending a library that is not contained in a document, two files are shown: dialog.xlb and script.xlb (see Table 2 and Figure 19). Both files are required and automatically appended regardless of which file you choose. In other words, you can select either dialog.xlb or script.xlb; both will be appended.

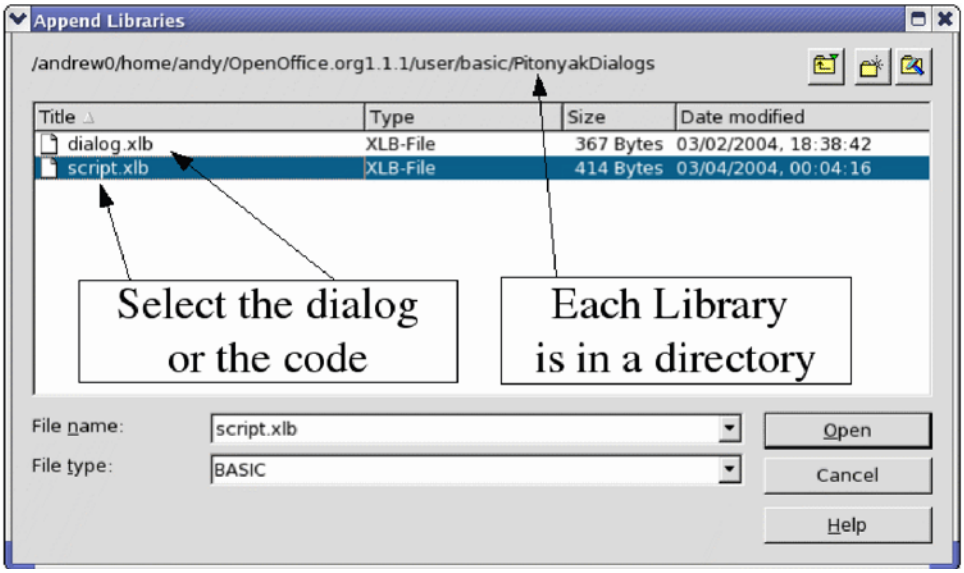


Figure 19. It doesn't matter which file you choose; both are appended.

TIP When I install a new version of OOo, I append my personal libraries from the previously installed version. I also move my libraries to other computers and install them there.

Chapter 16, "Library Management" in *OpenOffice.org Macros Explained* contains information and examples of manipulating and accessing libraries and modules using OOo Basic.

Conclusion

Macros are stored in modules, modules are stored in libraries, and libraries are stored in library containers. The application is a library container, as is every document. The IDE is used to create and debug macros and dialogs.

You have just completed one of the most difficult steps in writing macros for OpenOffice.org: writing your first macro! You are now ready to explore, try other macro examples, and create a few of your own.