Creating Basic Macros in OpenOffice

MASS PRODUCTION

www.photocase.de

The OpenOffice productivity suite can use a variety of scripts and macros to automate recurring tasks. The easiest approach is to use the integrated Basic dialect. This article helps you get started with this surprisingly sophisticated programming language. **BY OLIVER FROMMEL**

If you find yourself repeating complex, multi-step tasks in OpenOffice [1], it may be time to create a macro. OpenOffice supports a variety of programming options. Version 1.1 introduced the concept of bridges, which allow users to add their own C, C++, Java, or Python programs. In version 2.0, which is due for release any time now, the basic OpenOffice package will also support the Common Language Interface (CLI), which allows users to add their own Javascript and C# programs. Of all the languages used with OpenOffice, Basic is perhaps the easiest option. This article describes how to get started with creating macros in Basic.

Getting Started

Basic programming starts in the *Tools* | *Macros* menu, which has two entries: *Record macro*, a macro recorder for interactive use, and *Macro...*, which opens a new window (Figure 1).

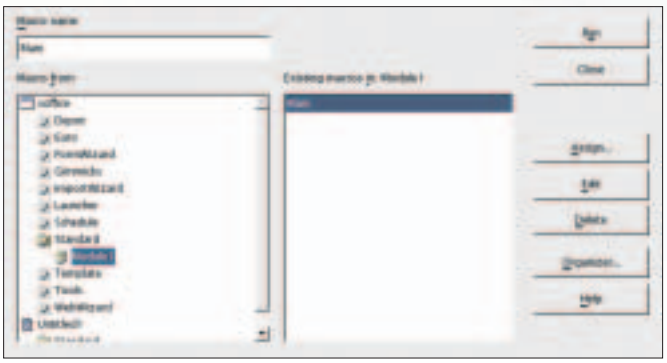

Figure 1: The OpenOffice Basic macro management window.

The window helps you organize the macros that accompany the OpenOffice distribution, and you can also manage your own macros.

Basic programs are assigned to a module and a library, which default to *Module1* and *Standard*. A new module will always contain the *Main* method, which is basically an empty framework without any code (containing only some *REM* remarks). Clicking on *Edit* takes you to the editor and displays the empty function framework (Figure 2).

You can use the full range of Basic features now. Selecting *Help* | *Contents* displays the OpenOffice Basic Help with a list of functions (*Macros and Programming* | *Commands* | *Alphabetical list..*).

Now add the following at the start of the program framework:

```
Sub Main
    Output = "It is " & Time()
    MsgBox Output, 0
End Sub
```

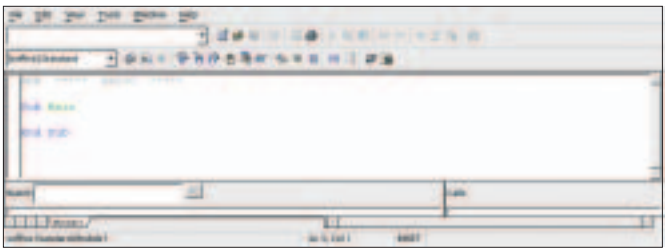

Figure 2: When you create a new macro, OpenOffice generates an empty Main function, as shown in the integrated editor.

The *Sub* keyword designates a function, which is called *Main* in this case. Actually, OpenOffice isn't overly interested in names and starts parsing at the first function it finds in the macro module. *Output* is a string that includes a constant part, "It is ", and the output of the *Time()* function, which gives us the time of day. The & operator concatenates these parts to form a single string. You can leave out the brackets for the *Time* function; they don't effect the output.

Finally, the *MsgBox* function displays a dialog box. The first parameter passes the text to be displayed, and the second parameter defines the dialog type. The *0* in this example means that the dialog box will only display an OK button. *1* would additionally give us a Cancel button, and there are other variants with Yes/No buttons and other combinations. *MsgBox* also returns a value that indicates which button the user has clicked. Our example does not actually do anything with the return value.

This Document

We need more than simple Basic functions if we want to access OpenOffice documents, as this involves accessing the UNO objects and interfaces (see "Box 1: Complex UNO, Simple Basic"). Our access point to this object hierarchy is *ThisDocument*, a keyword that references the document in which the script is running – this can be a text document, a spreadsheet, or even a drawing. *ThisDocument* is the parent object that gives you basic methods for navigating the document tree.

But before we do this, we will declare some variables; again this is optional and not too difficult. The *Dim* keyword will help us handle the job. The number of elements in a list (array) needs to be put in round brackets *(10)*. Basic handles this convention differently from most other programming languages: the parameter does not specify the number of elements but designates the highest index. Thus, *List(10)* gives you a list of 11 elements, *List(0)* through *list(10)* – Basic variables are not case sensitive.

The script needs to check if it's running inside an OpenOffice text document. To do so, we use the *supportsService()* method of the document reference:

```
oDoc = ThisComponent ➥
If oDoc.supportsService➥
("com.sun.star.➥
text.TextDocument") Then
```

Besides this method, the document object also has a promising looking function called *getText()*. The function does not give us the raw text from the document, but instead a reference to the text service which in turn has several methods, including methods for moving a cursor through the text (for instance *createTextCursor*).

The text service has another function called *createEnumeration()*, which enumerates the paragraphs in a document, but this still doesn't give us the raw text. On the contrary, a paragraph has about 150 paragraph properties that precisely describe its style, for example.

We can access the normal text elements in a paragraph by calling *createEnumeration()*. If the document contains a table, OpenOffice will output an error message at this point because *createEnumeration* does not recognize table elements. We would need to add some exception handling for this case.

The *String()* method gives us the raw text for an element. Basic uses a loop construction that starts with *Do* and

Figure 3: The Xray macro is useful for troubleshooting. It displays UNO object properties in Basic.

Box 1: Complex UNO, Simple Basic

OpenOffice has direct support for Basic, which removes the need for an external interface. The Basic language itself isn't exactly rocket science, but you will need some of the advanced features of Basic in order to work with the complex OpenOffice architecture.

OpenOffice has a language independent programming interface called UNO (Universal Network Objects). UNO follows today's software design paradigms (with support for so-called design patterns [2], services, and interfaces). The sheer bulk of the developer and [3] and API documentation [4] attests to the importance of UNO in OpenOffice programming.

OpenOffice Basic does not support the full range of UNO features, as the Basic language itself is quite simple. For example, Basic does not support complex data types, such as hashes, which assign values to keywords. This lack of hash support made it difficult to program what I originally thought might be a simple example: a script that would count the number of occurrences of a specific word in a text. A hash (or associative array) with the word to look for as a key would be an ideal solution for this example. But this would lead to a lot of extra programming in OpenOffice Basic; in fact, you would have to implement your own hash table, which is far beyond the scope of this article.

In contrast to OpenOffice Basic, the general UNO interface is object-oriented. These contrasting approaches lead to a few peculiarities: for example, some methods map directly to properties. In other words, a programmer does not need to call a function such as *circle.radius()* but can use the *circle.radius* attribute directly. This causes some confusion in practical applications that use both notations.

Listing 1: Simple Text Exporter

```
01 Sub Main
02    Dim oDoc As Object
03
04    Filename = "/home/oliver/
   output.txt"
05
06    oDoc = ThisComponent
07    title$ = oDoc.
   DocumentInfo.Title
08
09    If oDoc.supportsService
   ("com.sun.star.text.
   TextDocument") Then
10       FileNo = Freefile()
11       Open Filename For Output
   As #FileNo
12
13       oText = oDoc.getText()
14       oParagraphs = oText.
   createEnumeration()
15
16       Do While oParagraphs.
   hasMoreElements()
17          oPar = oParagraphs.
   nextElement()
18          oTexts = oPar.
   createEnumeration()
19          Do While oTexts.
   hasMoreElements()
20             oText = oTexts.
   nextElement()
21             Print #FileNo
   oText.string
22             If oText.string =
   "" Then
23                Print #FileNo
24             Endif
25          Loop
26       Loop
27    Endif ' If oDoc.
   supportsService(..)
28    Close #FileNo
29 End Sub
```

ends with *Loop* to parse enumerations. The break condition can follow either of these keywords. If the break occurs at the end of the loop, the script will iterate through the loop at least once.

Our sample macro writes the text data that we parsed using this method to a file. The filename is specified by the *Filename* variable. The unusual thing is that you need a file number to open a file; a call to *Freefile()* gives us the number. We can now pass the file number and name to *Open()* to open the file for writing:

```
Open Filename For Output ➥
  As #FileNo
```

The *Print* command with the file number as its first parameter allows us to add lines to the file: *Print #FileNo String*. Without the file number, OpenOffice opens a dialog box when you call *Print*. After adding the lines, we need to close the file. A call to *Close* with the file number as the argument takes care of this. Listing 1 shows the complete macro.

To launch the script, click on the second button from the left in the second row (see Figure 2). This button runs the script for the current document. If OpenOffice discovers a syntax error, it immediately reports an error executing the script and displays the error in a dialog box. Unfortunately, the error messages are typically fairly generic and not much use for troubleshooting (e.g., "Object variable not assigned"). A "generic error" occurs if you try to run the script while the Help browser is your current document.

The buttons with the curly brackets allow you to step through the code. You can select a variable name and then click on the button with the spectacles to view the value of the variable in the Watch field in the lower left of the window – again the restrictions mentioned previously apply. Check "Box 2: Advanced Troubleshooting" for more debugging tips.

Conclusion

The UNO interface for OpenOffice gives script authors a powerful programming toolbox for office applications. But OpenOffice programming is not the intuitive experience one might hope for. The system is just as complex as CORBA [8] or J2EE [9] and assumes knowledge of modern concepts such as component architecture and design patterns.

The prerequisite skills necessary to program OpenOffice macros puts the task beyond the reach of the casual user, but amateur and professional programmers with the required skills will find worlds to explore. ■

Box 2: Advanced Troubleshooting

The integrated debugger is only useful for simple Basic types. As the OpenOffice UNO types do not map directly to Basic types, the debugger will simply display question marks for UNO objects. Other methods give you more information but require some programming effort. For example, *Dbg_supportedInterfaces()* and *Dbg_methods()*, called as methods of an object, are quite useful.

A Basic macro can help simplify troubleshooting. The macro gives you a window with the methods and properties of the UNO object you need to investigate.

The macro is aptly named Xray [6] because it really lets you look inside an object (Figure 3).

After unpacking the Zip file, load the document in OpenOffice and follow the instructions. Basically, all you need to do is assign the Xray macro to the document you are working on (*Macro / Organizer*, then *Libraries*). To x-ray an object, you need to add a line such as *Xray.Xray oDoc* to your script.

Incidentally, the XRay website has another very useful document: a collection of script fragments [7] with short explanations by the macro book author, Andrew Pytonyak.

INFO

[1] OpenOffice: *http://www.OpenOffice.org*

[2] Design patterns: *http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29*

[3] Developer's Guide: *http://api.OpenOffice.org/docs/DevelopersGuide/DevelopersGuide.htm*

[4] UNO Reference: *http://api.OpenOffice.org/docs/common/ref/com/sun/star/module-ix.html*

[5] Starbasic Tutorial by Sun: *ftp://docs-pdf.sun.com/817-3924/817-3924.pdf*

[6] XRay: *http://www.ooomacros.org/dev.php#101416*

[7] Macros explained: *http://www.ooomacros.org/dev.php#91896*

[8] CORBA: *http://www.corba.org*

[9] J2EE: *http://java.sun.com/j2ee*