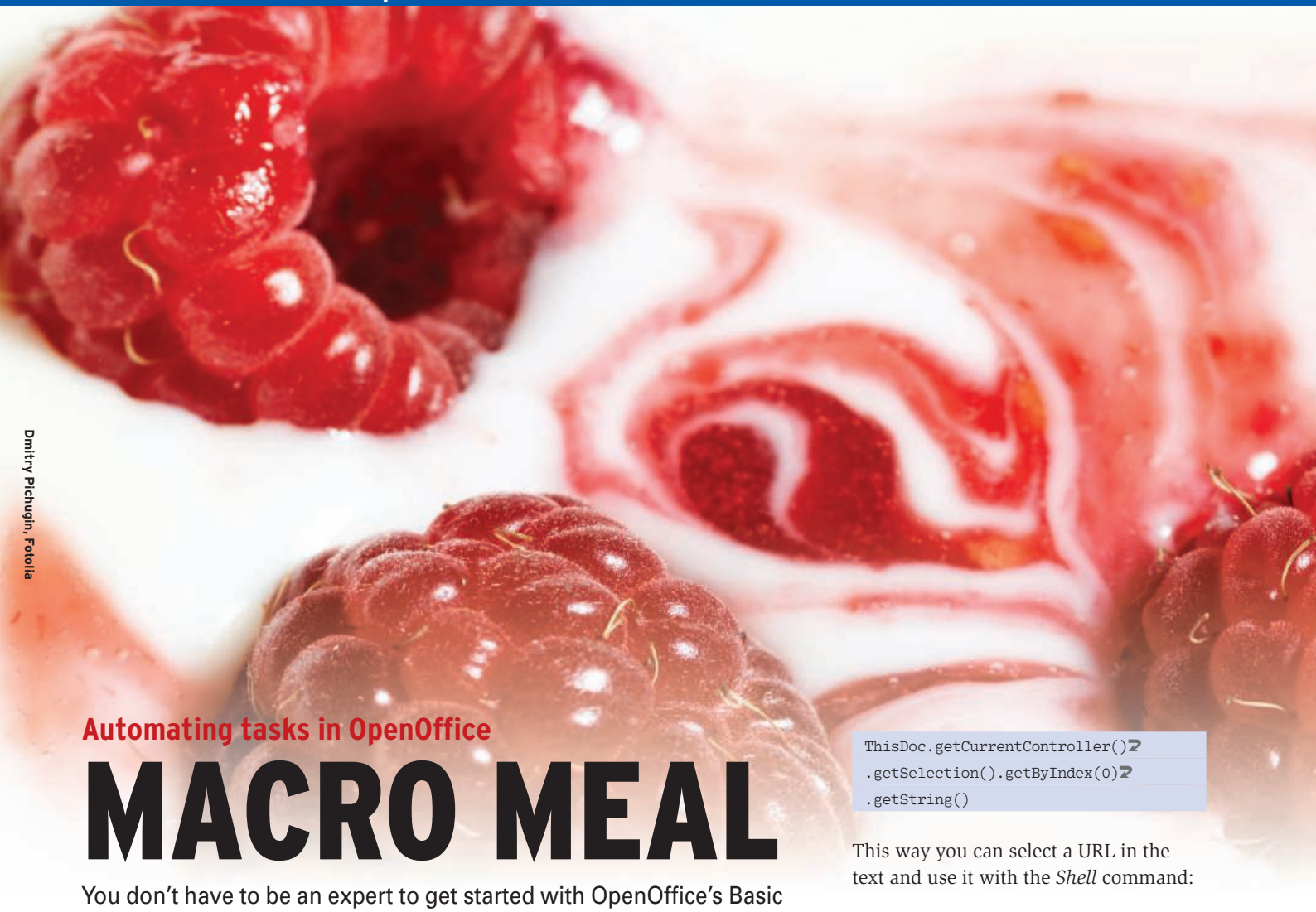

Dmitry Pichugin, Fotolia

Automating tasks in OpenOffice

MACRO MEAL

You don't have to be an expert to get started with OpenOffice's Basic programming interface. **BY DMITRI POPOV**

OpenOffice.org comes with its own Basic-based programming language. Although OOo Basic is not the most difficult programming language, it still requires some time and effort, especially if you are not a programmer. However, reading documentation and fiddling with code might not be necessary if you only need to automate a specific task now and then. In this article, I will provide a few pointers and code snippets, so you can put OOo Basic to some practical use without learning the language from scratch.

Launching External Apps

The ability to launch external applications and pass data to them is one of the most useful features of OpenOffice.org. Using the *Shell* command, you can launch virtually any application installed on your machine. The command has the following format: *Shell (Path, Windowstyle, Parameter)*. *Path* defines the path of the program. *Windowstyle* defines the window in which the program is started, and *Parameter* specifies the command-line parameter. For example, the *Shell* statement below opens the *http://wordnet.princeton.edu/perl/webwn* URL in the Firefox browser and brings it to the foreground:

```
Shell ("firefox", 1, "http://wordnet.princeton.edu/perl/webwn")
```

If you want to open another URL in Firefox, you must change it manually, which is not practical. Fortunately, OOo Basic provides a way to grab a text selection from the currently opened document:

```
ThisDoc=ThisComponent
TextSelection=ThisDoc.getCurrentController()
.getSelection().getByIndex(0)
.getString()
```

This way you can select a URL in the text and use it with the *Shell* command:

```
Shell ("firefox", 1, TextSelection)
```

Another way to acquire a URL is to display an input box in which the user can enter the URL. In this case, the code that opens the URL in Firefox looks like this:

```
InputText=InputBox
("Input field:", "Window title")
Shell ("firefox", 1, InputText)
```

Even with this very simple command, you can create some useful macros, like the one in Listing 1, which allows you to post messages to Twitter directly from within OpenOffice.org.

To post messages, the macro uses the *curl* utility, which must be installed on your machine. The utility uses the following command to send messages to Twitter:

Listing 1: PostToTwitter Macro

```
01 Sub PostToTwitter()
02 InputText=InputBox("Your message:",
     "Post to Twitter")
03 SplitStr = Split(InputText, " ")
04 Tweet = Join(SplitStr, "%20")
05 TwMessage=" -u username:password -d
     status=" & "" & Tweet & "" &
     " http://twitter.com/statuses/update.
     xml"
06 Shell("curl", 1, TwMessage)
07 End Sub
```

```
curl -u username:password -d ➥
status="Your tweet goes here." ➥
http://twitter.com/statuses/➥
update.xml
```

So the macro has to prompt the user to enter a message (called a *tweet* in Twitter parlance), which is then passed as an argument to the *curl* command with the use of the statement *InputText = InputBox("Your message:", "Post to Twitter")*.

The next two lines in the macro deserve a closer look. The problem is that the spaces in the message must be converted into the URL format; otherwise, each word in the message is posted as a separate tweet.

To convert the message into a URL, all spaces must be replaced with the *%20* string, so "*Your tweet goes here.*" becomes "*Your%20tweet%20goes%20here.*" *Split* and *Join* string routines do exactly that. The *Split* routing "chops" the string into text segments using the space and a separator, whereas the *Join* routine "glues" the text segments together using the *%20* string. Next, the macro has to construct the *curl* command argument, which is done by concatenating the required command-line parameters and the tweet (replace *username:password* with your actual Twitter user name and password):

```
TwMessage=" -u username:password -d ➥
status=" & "" & Tweet & "" & ➥
"http://twitter.com/statuses/➥
update.xml"
```

Finally, the macro uses the Shell command to launch the *curl* utility and post the tweet. As you can see, even these few simple commands are enough to create macros and put them to good use.

Working with Documents

The macro in Listing 2 might look a bit complicated, but it introduces a few useful techniques that let you obtain the name of the currently opened document and its path, check the document's status, and save it in a specified location. The macro can be divided into several steps. First, the macro defines the *FileProperties(0)* variable, which is later used to specify file properties for a backup copy of the current document. The *ThisDoc = ThisComponent* statement instructs the macro to use the current document, and the following code loads the *Tools* library:

```
If (Not GlobalScope.➥
BasicLibraries.isLibraryLoaded➥
("Tools")) Then
GlobalScope.BasicLibraries.➥
LoadLibrary("Tools")
End If
```

The *Tools* library contains routines that allow the macro to obtain the directory of the current document. But before the macro does that, it has to check to see whether the document is saved, (i.e., that it actually has a location). If it doesn't, the macro prompts the user to save the document and then quits:

```
If ThisDoc.hasLocation=➥
False Then
MsgBox ("You have to save ➥
document first!", , ➥
"Attention!") :End
End If
```

When saving a copy of the document, the macro appends the current date and time to the file name to make it easier for the user to find the desired version of the document. The macro uses the *CDateToISO* and *Format* routines to obtain the current date in ISO format (i.e., YYYY-MM-DD) and format the current time as HH-MM-SS:

```
DateToday=CDateToISO(Date) & "_" ➥
& Format(Hour(Now), "00") & ➥
"-" & Format(Minute(Now), ➥
"00") & "-"  & Format➥
(Second(Now), "00")
```

The macro obtains the document's URL and uses the *DirectoryNameoutofPath* routine to get the document's directory:

```
DocURL=ThisDoc.getURL()
DocDir=DirectoryNameoutofPath➥
(DocURL, GetPathSeparator())
```

The *Dir* routine then uses the URL to extract the name of the document:

```
FileName=Dir(DocURL, 0)
```

The macro constructs the path for the backup copy of the file, consisting of the path to the directory in which the original document is stored and the filename that includes the created date and time stamp:

```
SaveFile=DocDir & GetPathSeparator() & ➥
FileName & "_" & DateToday
```

The macro saves a backup copy of the document at a specified location using the *Overwrite* file property, which overwrites any file with the same name:

```
FileProperties(0).Name="Overwrite"
FileProperties(0).Value=True
ThisDoc.storeToURL➥
(SaveFile, FileProperties())
```

You can modify the macro to save a backup copy of the document on an FTP server. Add a statement that prompts the user to enter an FTP address (e.g., *ftp://user:password@192.168.1.7/backup/*):

```
FTPServerPath=InputBox➥
("Enter FTP path", "FTP address")
```

Listing 2: BackupDocument Macro

```
01 Sub BackupDocument()
02 Dim FileProperties(0) As New com.sun.
   star.beans.PropertyValue
03 ThisDoc=ThisComponent
04
05 If (Not GlobalScope.BasicLibraries.
   isLibraryLoaded("Tools")) Then
06 GlobalScope.BasicLibraries.
   LoadLibrary("Tools")
07 End If
08
09 If ThisDoc.hasLocation=False Then
10 MsgBox ("You have to save document
   first!", , "Attention!") :End
11 End If
12
13 DateToday=CDateToISO(Date) & "_" &
   Format(Hour(Now), "00") & "-" &
   Format(Minute(Now), "00") & "-"  &
   Format(Second(Now), "00")
14
15 DocURL=ThisDoc.getURL()
16 DocDir=DirectoryNameoutofPath(DocUR
   L, GetPathSeparator())
17 FileName=Dir(DocURL, 0)
18 SaveFile=DocDir & GetPathSeparator()
   & FileName & "_" & DateToday
19 FileProperties(0).Name="Overwrite"
20 FileProperties(0).Value=True
21 ThisDoc.storeToURL(SaveFile,
   FileProperties())
22 End Sub
```

Then modify the *SaveFile = DocDir & GetPathSeparator() & FileName & "_" & DateToday* statement so it looks like this:

```
SaveFile=FTPServerPath & ➥
FileName & "_" & DateToday
```

Then you can customize this macro as necessary for your own environment.

Dialogs

The *InputBox* routine allows you to display simple input boxes, but OOo Basic also lets you create proper dialog boxes containing multiple input fields, drop-down lists, buttons, and other GUI goodies. As an example of how to create a dialog box, consider the macro in Listing 3, which builds a tool that converts temperature from Fahrenheit and Celsius.

The macro displays a dialog box consisting of four elements: an input box, in which the user enters the desired value; a listbox containing conversion directions (i.e., "Celsius -> Fahrenheit" and "Fahrenheit -> Celsius"); another input field that displays the result of the conversion; and a button that performs the conversion and closes the dialog.

Before you start coding the macro, you must create the dialog. To do so, choose *Tools | Macros | Organize Dialogs*, and click the *New* button. Give the dialog a descriptive name (e.g., *SimpleConverterDialog*) and press the *OK* button. Use the available tools in the *Toolbox* bar to add input and result fields (must be numeric fields), a listbox, and a button. Use the *Properties* window to define properties for each element. For example, you have to set the button's type to *OK*. To do this, select the button and choose *OK* from the *Button Type* drop-down list in the *Properties* window. Also, you have to add the "Celsius -> Fahrenheit" and "Fahrenheit -> Celsius" entries to the listbox, which you can do by selecting the listbox field and adding the entries in the *List Entries* field of the *Properties* window. Using the *Name* field in the *Properties* window, you can give each element a name to make it easier to identify. For example, you might want to name the input field *InputField* and the result field *ResultField*.

Once the dialog is ready, you can start coding the macro. The macro first initializes the dialog with the following lines:

```
exitOK=com.sun.star.ui.dialogs.➥
ExecutableDialogResults.OK
Library=DialogLibraries.➥
GetByName("GUI")
TheDialog=Library.GetByName➥
("SimpleConverterDialog")
```

This code assumes that the *SimpleconverterDialog* dialog is stored in the *GUI* library. Next, the macro has to initialize the dialog fields:

```
DialogField1=➥
Dialog.getControl("InputField")
DialogField2=➥
Dialog.getControl("ListBox")
DialogField2.SelectItemPos(0, True)
DialogField3=➥
Dialog.getControl("ResultField")
```

Listing 3: TemperatureConverter Macro

```
01 Sub TemperatureConverter()
02
03 exitOK=com.sun.star.ui.dialogs.
   ExecutableDialogResults.OK
04 Library=DialogLibraries.
   GetByName("GUI")
05 TheDialog=Library.GetByName("SimpleCo
   nverterDialog")
06
07 DialogField1=Dialog.
   getControl("InputField")
08 DialogField2=Dialog.
   getControl("ListBox")
09 DialogField2.SelectItemPos(0, True)
10 DialogField3=Dialog.
   getControl("ResultField")
11
12 Dialog.execute()
13
14 InputValue=DialogField1.value
15
16 Select Case DialogField2.SelectedItem
17
18 Case "Fahrenheit -> Celsius"
19 ConvertedValue=(InputValue-32)*5/9
20 DialogField3=Dialog.
   getControl("ResultField").
   setValue(ConvertedValue)
21 Button=Dialog.
   getControl("CommandButton")
22 Button.Label = "Close"
23 Dialog.execute()
24
25 Case "Celsius -> Fahrenheit"
26 ConvertedValue=InputValue*9/5+32
27 DialogField3=Dialog.
   getControl("ResultField").
   setValue(ConvertedValue)
28 Button=Dialog.
   getControl("CommandButton")
29 Button.Label = "Close"
30 Dialog.execute()
31
32 End Select
33 End Sub
01 Sub TemperatureConverter()
02
03 exitOK=com.sun.star.ui.dialogs.
   ExecutableDialogResults.OK
04 Library=DialogLibraries.
   GetByName("GUI")
05 TheDialog=Library.GetByName("SimpleCo
   nverterDialog")
06
07 DialogField1=Dialog.
   getControl("InputField")
08 DialogField2=Dialog.
   getControl("ListBox")
09 DialogField2.SelectItemPos(0, True)
10 DialogField3=Dialog.
   getControl("ResultField")
11
12 Dialog.execute()
13
14 InputValue=DialogField1.value
15
16 Select Case DialogField2.SelectedItem
17
18 Case "Fahrenheit -> Celsius"
19 ConvertedValue=(InputValue-32)*5/9
20 DialogField3=Dialog.
   getControl("ResultField").
   setValue(ConvertedValue)
21 Button=Dialog.
   getControl("CommandButton")
22 Button.Label = "Close"
23 Dialog.execute()
24
25 Case "Celsius -> Fahrenheit"
26 ConvertedValue=InputValue*9/5+32
27 DialogField3=Dialog.
   getControl("ResultField").
   setValue(ConvertedValue)
28 Button=Dialog.
   getControl("CommandButton")
29 Button.Label = "Close"
30 Dialog.execute()
31
32 End Select
33 End Sub
```

The macro then executes the dialog using the *Dialog.execute* command and assigns the value of the *InputField* to the *InputValue* variable:

```
InputValue=DialogField1.value
```

The *Select Case* command redirects the macro to the appropriate conversion formula, depending on what entry the user selected in the listbox. For example, if the user has selected "Fahrenheit -> Celsius", the macro runs the following:

```
ConvertedValue=(InputValue-32)*5/9
```

The result of the conversion is then inserted into the result field:

```
DialogField3=Dialog.getControl➥
("ResultField").setValue➥
(ConvertedValue)
```

Finally, the macro changes the label of the button to *Close*, and when the user presses the button, the dialog closes and the macro stops:

```
Button=Dialog.getControl➥
("CommandButton")
Button.Label = "Close"
Dialog.execute()
```

The converter macro can be modified easily to perform other types of conversion. All you have to do is specify new entries in the listbox and add a new *Case* code block with the appropriate conversion formula to the macro.

OOo Basic contains tools that let you connect to a database and manipulate the data in it. The macro in Listing 4 demonstrates how to establish a connection to a local database and also shows how to create a query and display the results in a dialog window. *ShowWordlist* is a rather simple macro that connects to a database called *TinyDB*. It then finds all records in the *Words* column in the *wordlist* table and populates a listbox in the *Wordlist* dialog with the results. Before you can establish a connection to the database, register it as a data source in OpenOffice. To do this, choose *Tools | Options*. Select *OpenOffice.org Base | Databases* and press the *New* button. Select the *TinyDB.odb* database and give the new connection the name *TinyDB*. Press *OK | OK*, and you are done. Establishing a connection to the database using OOo Basic requires only three lines of code:

```
DBContext=createUnoService➥
("com.sun.star.sdb.➥
DatabaseContext")
DataSource=➥
DBContext.getByName("TinyDB")
Database=➥
DataSource.GetConnection ("","")
```

Because OpenOffice.org Base uses the SQL language to manipulate database data, creating a query that retrieves all the records in the *wordlist* table is a matter of using the *SELECT FROM* SQL command. To retrieve records from the database using the specified SQL query, the macro uses the *RowSet* service, which the macro must first initiate:

```
SQLResult=createUnoService➥
("com.sun.star.sdb.RowSet")
SQLQuery="SELECT ""Words"" ➥
FROM ""wordlist"""
```

The macro then executes the SQL query using the following code:

```
SQLResult.activeConnection=➥
Database
SQLResult.Command=SQLQuery
SQLResult.execute
```

By now, you know how to initiate a dialog box, so the only part that requires a closer look is the following code block:

```
While SQLResult.next
ListBoxItem=SQLResult.getString(1)
DialogField.additem➥
(ListBoxItem, DialogField.➥
ItemCount)
Wend
```

To populate the listbox, the macro uses the *While…Wend* loop, which picks a record (*SQLResult.next*), extracts the string from the first column – (*ListBoxItem = SQLResult.getString(1)*), and inserts it as a list box item – (*DialogField.additem(ListBoxItem, DialogField.ItemCount)*). Once the macro is done, it closes the database connection:

```
Database.close
Database.dispose()
```

The OpenOffice.org Extension Repository contains useful extensions, many of which are released under open source licenses, so you can use the code in your own creations. For example, you will find some of the techniques and code described in this article in the Writer's Tools extension [1], created by yours truly. Happy programming! ■

Listing 4: ShowWordlist Macro

```
01 Sub ShowWordlist()
02
03 DBContext=createUnoService("com.sun.
   star.sdb.DatabaseContext")
04 DataSource=DBContext.
   getByName("TinyDB")
05 Database=DataSource.GetConnection
   ("","")
06
07 SQLResult=createUnoService("com.sun.
   star.sdb.RowSet")
08 SQLQuery="SELECT ""Words"" FROM
   ""wordlist"""
09 SQLResult.activeConnection=Database
10 SQLResult.Command=SQLQuery
11 SQLResult.execute
12
13 exitOK=com.sun.star.ui.dialogs.
   ExecutableDialogResults.OK
14 Library=DialogLibraries.
   GetByName("Wordlist")
15 TheDialog=Library.
   GetByName("WordlistDialog")
16
17 DialogField=Dialog.
   GetControl("ListBox")
18
19 While SQLResult.next
20 ListBoxItem=SQLResult.getString(1)
21 DialogField.additem(ListBoxItem,
   DialogField.ItemCount)
22 Wend
23
24 If Dialog.Execute=exitOK Then
25 CurrentItemName=DialogField.
   SelectedItem
26 End If
27
28 Database.close
29 Database.dispose()
30 End Sub
```

INFO

[1] Writer's Tools: *writertools.googlecode.com*