3.6.7. What type is this object?

It is useful to know the object type so that you know what you can do with it. Here is a brief list of inspection methods:

Table 3.2: *Methods used to inspect variables.*

Method	Description
IsArray	Is the parameter an array?
IsEmpty	Is the parameter an uninitialized variant type?
IsNull	Does the parameter contain “no” data?
IsObject	Is the parameter an OLE object?
IsUnoStruct	Is the parameter an UNO structure?
TypeName	What is the type name of the parameter?

A variables type name can also provide information about a variables properties.

Table 3.3: *Values returned by the TypeName() statement.*

Type	TypeName()
Variant	“Empty” or name of contained object
Object	“Object” even if it is null. Same for structures.
regular type	regular type name such as “String”
array	name followed by “()”

Listing 3.9 demonstrates the statements in Table 3.2 for different variable types.

Listing 3.9: *Inspect variables.*

```
Sub TypeTest
  Dim oSFA
  Dim aProperty As New com.sun.star.beans.Property
  oSFA = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
  Dim v, o As Object, s As String, ss$, a(4) As String
  ss = "Empty Variant: " & GetSomeObjInfo(v) & chr(10) &_
    "Empty Object: " & GetSomeObjInfo(o) & chr(10) &_
    "Empty String: " & GetSomeObjInfo(s) & chr(10)
  v = 4
  ss = ss & "int Variant: " & GetSomeObjInfo(v) & chr(10)
  v = o
  ss = ss & "null obj Variant: " & GetSomeObjInfo(v) & chr(10) &_
```

```
    "struct: " & GetSomeObjInfo(aProperty) & chr(10) &_
    "service: " & GetSomeObjInfo(oSFA) & chr(10) &_
    "array: " & GetSomeObjInfo(a())
  MsgBox ss, 64, "Type Info"
End Sub

REM Returns basic type information for the parameter.
REM This also returns the dimensions of an array.
Function GetSomeObjInfo(vObj) As String
  Dim s As String
  s = "TypeName = " & TypeName(vObj) & CHR$(10) &_
    "VarType = " & VarType(vObj) & CHR$(10)
  If IsNull(vObj) Then
    s = s & "IsNull = True"
  ElseIf IsEmpty(vObj) Then
    s = s & "IsEmpty = True"
  Else
    If IsObject(vObj) Then
      On Local Error GoTo DebugNoSet
      s = s & "Implementation = " & _
               NotSafeGetImplementationName(vObj) & CHR$(10)
      DebugNoSet:
      On Local Error Goto 0
      s = s & "IsObject = True" & CHR$(10)
    End If
    If IsUnoStruct(vObj) Then s = s & "IsUnoStruct = True" & CHR$(10)
    If IsDate(vObj) Then s = s & "IsDate = True" & CHR$(10)
    If IsNumeric(vObj) Then s = s & "IsNumeric = True" & CHR$(10)
    If IsArray(vObj) Then
      On Local Error Goto DebugBoundsError:
      Dim i%, sTemp$
      s = s & "IsArray = True" & CHR$(10) & "range = ("
      Do While (i% >= 0)
        i% = i% + 1
        sTemp$ = LBound(vObj, i%) & " To " & UBound(vObj, i%)
        If i% > 1 Then s = s & ", "
        s = s & sTemp$
      Loop
      DebugBoundsError:
      On Local Error Goto 0
```

```
        s = s & ")" & CHR$(10)
      End If
    End If
    GetSomeObjInfo = s
End Function

REM This places an error handler where it will catch the problem
REM and return something anyway!
Function SafeGetImplementationName(vObj) As String
    On Local Error GoTo ThisErrorHere:
    SafeGetImplementationName = NotSafeGetImplementationName(vObj)
    Exit Function
ThisErrorHere:
    On Local Error GoTo 0
    SafeGetImplementationName = "*** Unknown ***"
End Function

REM The problem is that if this Function is called and the vObj
REM type does NOT support the getImplementationName() call,
REM then I receive an "Object variable not set" error at
REM the Function definition.
Function NotSafeGetImplementationName(vObj) As String
    NotSafeGetImplementationName = vObj.getImplementationName()
End Function
```

3.6.8. What methods, properties, interfaces, and services are supported?

UNO objects usually support ServiceInfo, which provides information about the service. Use getImplementationName() to obtain the fully qualified object name. Use the fully qualified name to search Google or the Developer's Guide for more information. Listing 3.10 demonstrates how to print the methods, interfaces, and properties for an UNO object.

Listing 3.10: *What can this object do?*

```
MsgBox  vObj.dbg_methods              'Methods for this object.
MsgBox  vObj.dbg_supportedInterfaces  'Interfaces for by this object.
MsgBox  vObj.dbg_properties           'Properties for this object.
```

OOo includes macros to display debug information. The most commonly used included macros are PrintdbgInfo(object) and ShowArray(object). The WritedbgInfo(object) macro inserts object debug information into an open writer document.

3.6.9. Languages other than Basic

StarBasic provides numerous conveniences not provided by other programming languages. This section touches on only a few of these.

3.6.9.1. CreateUnoService

The CreateUnoService() method is a short cut to obtaining the global service manager and then calling createInstance() on the service manager.

Listing 3.11: *Get the global process service manager.*

```
oManager = GetProcessServiceManager()
oDesk = oManager.createInstance("com.sun.star.frame.Desktop")
```

In StarBasic, this process may be done in a single step – unless you need to use createInstanceWithArguments() that is.

Listing 3.12: *CreateUnoService is less code than using the process service manager.*

```
oDesk = CreateUnoService("com.sun.star.frame.Desktop")
```

Other languages, such as Visual Basic, do not support the CreateUnoService() method.

Listing 3.13: *Create a UNO service using Visual Basic.*

```
Rem Visual Basic does not support CreateUnoService().
Rem The service manager is always the first thing to create
REM In Visual Basic.
Rem If OOo is not running, it is started.
Set oManager = CreateObject("com.sun.star.ServiceManager")
Rem Create a desktop object.
Set oDesk = oManager.createInstance("com.sun.star.frame.Desktop")
```

3.6.9.2. ThisComponent

In StarBasic, ThisComponent references the current the document, or the document that caused a macro to be called. ThisComponent is set when the macro is started, and does not change, even if the macro causes a new document to become current – this includes closing ThisComponent. Even with the pitfalls, ThisComponent is a nice convenience. In languages other than Basic, the common solution is to use the getCurrentComponent() on the desktop object. If the Basic IDE or help window is current, getCurrentComponent() returns these objects, which are not OOo Documents.

3.6.9.3. StarDesktop

StarDesktop references the desktop object which is essentially the primary OOo application. The name originated when the product was named StarOffice and it displayed a main desktop object that contained all of the open components. Examples of components that the desktop object may contain include all supported documents, the BASIC Integrated Development Environment (IDE), and the included help pages (see *Figure 3.4*).

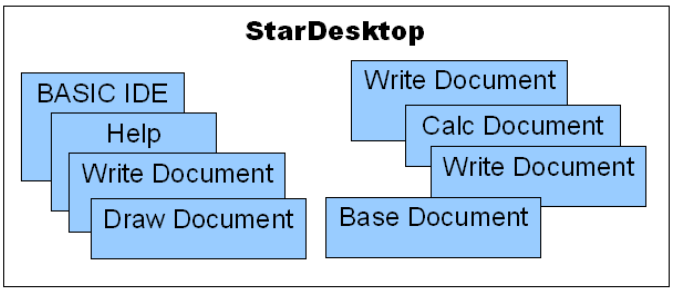

Figure 3.4: The desktop contains components.

Getting back to the macro in *Listing 3.19*, StarDesktop provides access to the currently open components. The method `getCurrentComponent()` returns the currently active component. If the macro is run from the BASIC IDE, then a reference to the BASIC IDE is returned. If the macro is run while a document is displayed, probably by using **Tools > Macros > Run Macro** then `oComp` will reference the current document.

TIP The global variable `ThisComponent` refers to the currently active document. If a non-document type component has the focus, then `ThisComponent` refers to the last active document. As of OOo version 2.01, the Basic IDE, help pages, and Base documents do not cause `ThisComponent` to be set to the current component.

3.6.10. Accessing methods and properties

StarBasic automatically makes the methods and properties supported by an object available – StarBasic sometimes makes properties available that are not available using other methods. In other languages, the interface that defines the method that you want to call must be extracted before it can be used (see Listing 3.14).

Listing 3.14: *In Java, you must obtain an interface before you can use it.*

```
XDesktop xDesk;
xDesk = (XDesktop) UnoRuntime.queryInterface(XDesktop.class, desktop);
XFrame xFrame = (XFrame) xDesk.getCurrentFrame();
XDispatchProvider oProvider = (XDispatchProvider)
UnoRuntime.queryInterface(XDispatchProvider.class, xFrame);
```

If the view cursor is in a text section, the TextSection property contains a reference to the text section. If not, the TextSection property is null. In StarBasic, I can obtain the text section as follows:

Listing 3.15: *OOo Basic allows you to access properties directly.*

```
If IsNull(oDoc.CurrentController.getViewCursor().TextSection) Then
```

In a language other than StarBasic, the CurrentController property and the TextSection property are not directly available. The current controller is available using a “get” method, and the text section is available as a property value. Code that uses get and set methods is easier to translate into other languages than code that uses properties.

Listing 3.16: *Access some properties using get methods.*

```
oVCurs = oDoc.getCurrentController().getViewCursor()
If IsNull(oVCurs.getPropertyValue("TextSection")) Then
```

StarBasic allows some properties to act as an array, even if the property is not an array; specifically, properties that implement an interface for indexed access. Consider the Sheets property in a Calc document. In Calc, both of the following accomplish the same task, but only the second example works outside of StarBasic.

Listing 3.17: *OOo Basic allows you to access some properties as an array.*

```
oDoc.sheets(1)          ' Access as an array.
oDoc.getSheets().getByIndex(1)       ' Use a method.
```

3.7. Summary

The StarBasic language refers to the syntax and the commands for general programming. StarBasic is easy to use.

Listing 3.18: Simple macro that does not access the OOo API.

```
Sub SimpleExample()
  Dim i As Integer
  i = 4
  Print "The value of i = " & i
End Sub
```

Most macros are written to interact with the OOo components. You must, therefore, learn about the methods and properties for each object that you want to use; this is difficult.

Listing 3.19: Simple macro that uses the OOo API to inspect the current component.

```
Sub ExamineCurrentComponent
  Dim oComp
  oComp = StarDesktop.getCurrentComponent()
  If HasUnoInterfaces(oComp, "com.sun.star.frame.XStorable") Then
    If oComp.hasLocation() Then
      Print "The current component has URL: " & oComp.getLocation()
    Else
      Print "The current component does not have a location."
    End If
  Else
```

```
      Print "The current component is not storable"
    End If
  End Sub
```

Although the macro in *Listing 3.19* is simple, it requires a lot knowledge to write. The macro starts by declaring the variable `oComp`, which defaults to type Variant because the type is not explicitly given.

4. Examples

4.1. Debugging And Inspecting Macros

It can be difficult to determine what methods and properties are available for an object. The methods in this section should help.

4.1.1. Determine Document Type

In OOo, most of the functionality is defined by services. To determine the document type, look at the services it supports. The macro shown below uses this method. I assume that this is safer than using `getImplementationName()`.

Listing 4.1: *Identify most OpenOffice.org document types.*

```
'Author: Included with OpenOffice
'Modified by Andrew Pitonyak
Function GetDocumentType(oDoc)
  Dim sImpress$
  Dim sCalc$
  Dim sDraw$
  Dim sBase$
  Dim sMath$
  Dim sWrite$

  sCalc    = "com.sun.star.sheet.SpreadsheetDocument"
  sImpress = "com.sun.star.presentation.PresentationDocument"
  sDraw    = "com.sun.star.drawing.DrawingDocument"
  sBase    = "com.sun.star.sdb.DatabaseDocument"
  sMath    = "com.sun.star.formula.FormulaProperties"
  sWrite   = "com.sun.star.text.TextDocument"

  On Local Error GoTo NODOCUMENTTYPE
  If oDoc.SupportsService(sCalc) Then
    GetDocumentType() = "scalc"
  ElseIf  oDoc.SupportsService(sWrite) Then
    GetDocumentType() = "swriter"
  ElseIf  oDoc.SupportsService(sDraw) Then
    GetDocumentType() = "sdraw"
  ElseIf  oDoc.SupportsService(sMath) Then
    GetDocumentType() = "smath"
  ElseIf  oDoc.SupportsService(sImpress) Then
   GetDocumentType() = "simpress"
  ElseIf  oDoc.SupportsService(sBase) Then
    GetDocumentType() = "sbase"
```

```
  End If
  NODOCUMENTTYPE:
  If Err <> 0 Then
    GetDocumentType = ""
    Resume GOON
    GOON:
  End If
End Function
```

Listing 4.2 returns the name of the PDF export filter based on the document type.

Listing 4.2: *Use the document type to determine the PDF export filter.*

```
Function GetPDFFilter(oDoc)
  REM Author: Alain Viret [Alain.Viret@bger.admin.ch]
  REM Modified by Andrew Pitonyak
  On Local Error GoTo NODOCUMENTTYPE
  Dim sImpress$
  Dim sCalc$
  Dim sDraw$
  Dim sBase$
  Dim sMath$
  Dim sWrite$

  sCalc    = "com.sun.star.sheet.SpreadsheetDocument"
  sImpress = "com.sun.star.presentation.PresentationDocument"
  sDraw    = "com.sun.star.drawing.DrawingDocument"
  sBase    = "com.sun.star.sdb.DatabaseDocument"
  sMath    = "com.sun.star.formula.FormulaProperties"
  sWrite   = "com.sun.star.text.TextDocument"

  On Local Error GoTo NODOCUMENTTYPE
  If oDoc.SupportsService(sCalc) Then
    GetPDFFilter() = "calc_pdf_Export"
  ElseIf  oDoc.SupportsService(sWrite) Then
    GetPDFFilter() = "writer_pdf_Export"
  ElseIf  oDoc.SupportsService(sDraw) Then
    GetPDFFilter() = "draw_pdf_Export"
  ElseIf  oDoc.SupportsService(sMath) Then
    GetPDFFilter() = "math_pdf_Export"
  ElseIf  oDoc.SupportsService(sImpress) Then
    GetPDFFilter() = "impress_pdf_Export"
```

```
    End If
    NODOCUMENTTYPE:
    If Err <> 0 Then
      GetPDFFilter() = ""
      Resume GOON
      GOON:
    End If
End Function
```

4.2. X-Ray

Bernard Marcelly wrote a tool called X-Ray that displays object information in a dialog. X-Ray tool is available for download from http://www.ooomacros.org/dev.php, and is highly recommended by many people!

4.3. Dispatch: Using Universal Network Objects (UNO)

http://udk.openoffice.org and the Developer's Guide are good references in your quest to understand UNO. UNO is a component model offering interoperability between different programming languages, object models, machine architectures, and processes.

On the Windows platforms, many software packages use the existing COM. OOo, however, has its own multi-platform component object model. By using its own object model, OOo's functionality is not limited to Windows. Secondly, OOo can provide a better error handling system than is provided by COM. Nevertheless, COM can still be used to control OpenOffice (Windows). For more information, see http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.htm#1+4+4+Automation+Bridge.

This example dispatches an UNO command to perform the “undo” command.

Listing 4.3: *Use the new dispatch method to perform the undo operation.*

```
Sub Undo
  Dim oDisp
  Dim oFrame
  oFrame = ThisComponent.CurrentController.Frame
  oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
  oDisp.executeDispatch(oFrame,".uno:Undo", "", 0, Array())
End Sub
```

The difficult part is knowing the UNO interface and the parameters for each. Consider the following which should work with newer versions of OOo.

Listing 4.4: *Use the new dispatch method to export a Write document to PDF.*

```
Dim a(2) As New com.sun.star.beans.PropertyValue
a(0).Name = "URL"        : a(0).Value = "my_file_name_.pdf"
```

```
a(1).Name = "FilterName" : a(1).Value = "writer_pdf_Export"
oDisp.executeDispatch(oFrame, ".uno:ExportDirectToPDF", "", 0, a())
```

Listing 4.5: *Use the new dispatch method to go to a cell, copy, and paste.*

```
Dim a(1) As New com.sun.star.beans.PropertyValue
a(0).Name = "ToPoint" : a(0).Value = "$B$3"
oDisp.executeDispatch(oFrame, ".uno:GoToCell", "", 0, a())
oDisp.executeDispatch(oFrame, ".uno:Copy",     "", 0, Array())
oDisp.executeDispatch(oFrame, ".uno:Paste",    "", 0, Array())
```

4.3.1. The dispatcher requires a user interface

Hal Vaughan asked questions, Mathias Bauer answered them.

Q: Will Dispatch Names Change?

A: Macros using dispatch names rather than numbers will not change between OOo versions.

Q: Is there any reason to use the regular API calls instead of calling the dispatcher with the function name?

A: Dispatch calls do not work on a document without a UI. If OOo is run in a real "server" mode where documents can be loaded and scripted without any GUI, only macros using the “real” API will work. The real API is also much more powerful and gives you a better insight in the real objects. IMHO you should use the dispatch API only for two reasons:

1. Recording macros
2. As a workaround when a certain task can not be done by any “real” API (because it does not exist or it is broken).

4.3.1.1. Modifying the menu – a dispatcher example

In OOo 2.0, you should be able to assign a macro to a menu item using the API:

http://specs.openoffice.org/ui_in_general/api/ProgrammaticControlOfMenuAndToolbarItems.sxw

The API IDL states that there is an XMenu Interface and also an XMenuListener. Retrieving the Menu ID from the XML file that defines the menu is possible, should be able to assign a macro to that ID ??? test this! You can also use a DispatchProviderInterceptor, which can be compared to a Listener or Handler.

Use a DispatchProviderInterceptor to listen for Dispatch commands. It is possible to intercept almost every command. Using slot ids, we can assign macros to special DispatchCommands that represent menu items. Although we can not prevent a user from selecting a menu item, we can intercept the command when it is dispatched. A DispatchInterceptor must be implemented and registered.

The ToggleToolbarVisibility macro, written by Peter Biela, toggles the tool bar visibility.