Track Component Use with Excel VBA

by Chris Barlow

Use Excel 97 to analyze file usage across your Visual Basic projects.

Click & Retrieve Source CODE!

Visual Basic is built on the concept of reusable components. If you plan carefully as you develop your procedures, you can build up a powerful code library of forms, modules, and classes that will make each new project easier to develop than the previous one. At my company, SunOpTech, we have a standard module, SUNOP.BAS, that contains procedures used in most of our projects. The procedures parse strings, extract file names from paths, trim null characters from a string, and so on. Another module, SOTDTS.BAS, contains procedures that convert dates and times between double, long, and hexadecimal strings.

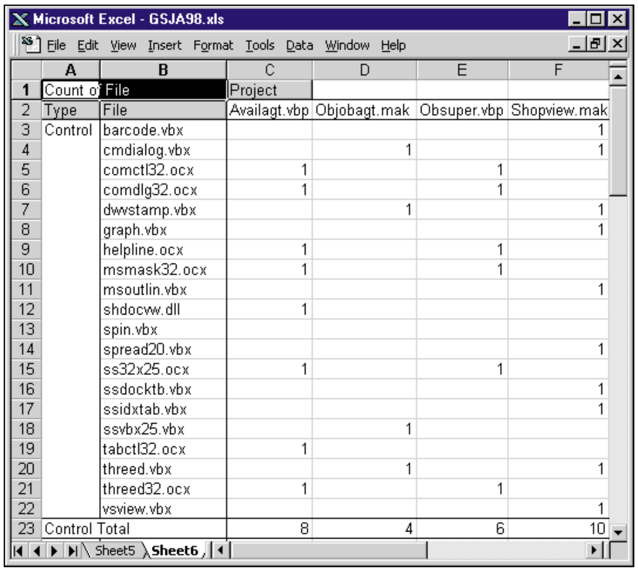
The OCX and VBX files that contain ActiveX controls encapsulate sophisticated functionality you can reuse in your Visual Basic projects. However, you can get many of the same reusability benefits by developing your own class modules. You can use your own standard set of features in future projects—either directly or by compiling them into ActiveX servers.

As you develop multiple projects over time by reusing certain components, you will have a more and more difficult time managing these shared components. Did the OBJobAgent project use the SPREAD.VBX or the newer SSVBX25.VBX? Did the OBSuper project share the OBORD32 module with the AvailAgent project? You need to make a change to the version-checking procedure—which projects use the OBVer module?

As you upgrade shared procedures or controls, you don't want to waste all productivity savings by breaking existing projects. Some source code repositories are designed to help you manage these components, but most smaller organizations don't have an easy way to handle these problems. As a Visual Basic programmer, you can use Visual Basic for Applications (VBA) within Excel 97 to develop some Excel pivot tables that show component usage by project (see Figure 1).

Visual Basic project files are simple ASCII files that contain the paths and file names for the files in the project. Visual Basic versions 1 through 3 used a simple file that showed only the forms, modules, and VBX files contained in the project (see Listing 1). VB4 and VB5 changed the file format and the extension to include more detailed information required for the projects, such as class modules, OCX GUIDs and versions, and referenced ActiveX servers (see Listing 2).

You can open these project files as regular ASCII files, read the file names, identify the file type, and use the built-in capabilities of Excel 97 to create a pivot table from the data. Because of Office 97's programmability and the fact that the same


Microsoft Excel - GSJA98.xls
File Edit View Insert Format Tools Data Window Help
Sheet5
Sheet6

	A	B	C	D	E	F
1	Count of File		Project			
2	Type	File	Availagt.vbp	Objobagt.mak	Obsuper.vbp	Shopview.mak
3	Control	barcode.vbx				1
4		cmdialog.vbx		1		1
5		comctl32.ocx	1		1	
6		comdlg32.ocx	1		1	
7		dwwstamp.vbx		1		1
8		graph.vbx				1
9		helpline.ocx	1		1	
10		msmask32.ocx	1		1	
11		msoutlin.vbx				1
12		shdocvw.dll	1			
13		spin.vbx				
14		spread20.vbx				1
15		ss32x25.ocx	1		1	
16		ssdocktb.vbx				1
17		ssidxtab.vbx				1
18		ssvbx25.vbx		1		
19		tabctl32.ocx	1			
20		threed.vbx		1		1
21		threed32.ocx	1		1	
22		vsview.vbx				1
23	Control Total		8	4	6	10

FIGURE 1 ***Create a Project Pivot Table.*** *This pivot table was created using the CreateVBProjCrossRef procedure. Notice how easy it is to see which projects use each of the controls.*

Chris Barlow is president of SunOpTech, a developer of manufacturing decision-support and supply chain applications, including the ObjectBank, ObjectOrder, and ObjectJob Systems. He holds two patents for distributed asynchronous object-oriented and scheduling systems. Chris, who is a frequent speaker at VBITS, Tech•Ed, and DevDays and has been featured in two Microsoft videos, holds degrees from Harvard Business School and Dartmouth College. Reach Chris at ChrisB@SunOpTech.com.

VB3

```
SHOPVW2.FRM
C:\WINDOWS\SYSTEM\CMDIALOG.VBX
SHOPVIEW.BAS
..\OBORD3.BAS
..\SOTDTS.BAS
..\SUNOP.BAS
C:\WINDOWS\SYSTEM\THREED.VBX
C:\WINDOWS\SYSTEM\SPREAD20.VBX
..\OBJOB.BAS
C:\WINDOWS\SYSTEM\SSIDXTAB.VBX
C:\WINDOWS\SYSTEM\SSDOCKTB.VBX
DBV.BAS
C:\WINDOWS\SYSTEM\DWVSTAMP.VBX
..\OBJSAF.BAS
..\UNC.BAS
..\DEBUG3.BAS
..\OBCONS3.BAS
..\SPREAD.BAS
C:\WINDOWS\SYSTEM\GRAPH.VBX
..\OBCALNDR.BAS
..\OBMASTER.BAS
C:\WINDOWS\SYSTEM\VSVIEW.VBX
..\PRICING.BAS
..\NBOM.BAS
..\VER.BAS
..\TOOLS.BAS
..\OBSTOCK.BAS
C:\WINDOWS\SYSTEM\MSOUTLIN.VBX
..\OBORD16.BAS
SHOPVW3.BAS
..\BARCODE.BAS
C:\WINDOWS\SYSTEM\BARCODE.VBX
SHOPVIEW.FRM
INIFORM.FRM
ABOUTSHV.FRM
DETAIL.FRM
MDICOMPI.FRM
COMPLETE.FRM
..\FOBJBANK.FRM
FSORT.FRM
SHOPGRAP.FRM
FRM_PREV.FRM
JOB_PREV.FRM
..\TOOLSCAN.FRM
FWORKSP.FRM
CHANGE.FRM
ASSIGN.FRM
FRM_HOLD.FRM
FSHOPQUA.FRM
FMULTSCH.FRM
FPRTRAV.FRM
ProjWinSize=26,541,257,481
ProjWinShow=2
Command="\\bmach1\sunop\database"
IconForm="FrmMain"
Title="ShopView"
ExeName="SHOPVIEW.EXE"
```

LISTING 1 ***Typical Visual Basic 3 MAK File Format.*** *The VB3 MAK file contains a list of the paths and file names for the forms, modules, and VBXs contained in the project.*

language you use in standalone Visual Basic is built into all Office applications, you have many options for creating this solution. You could write the procedures using Visual Basic 5 and drive Excel as an ActiveX server. But I'm going to show you how to write the procedures within the Visual Basic environment hosted by Excel so you can learn how easy it is to work with Excel. You'll also see how similar VBA is to the standalone Visual Basic product.

SET UP YOUR PROCEDURES

Launch Excel and press Alt-F11 to open the Visual Basic development environment. This should look familiar to you—the environment is the same in all Visual Basic products. If you're a VB programmer, you're also a VBA programmer. Insert a new module and type this code in the code window:

```
Sub CreateVBProjCrossRef()
SetUpHeadings
Do While LoadProjectFile
Loop
CreatePivotTable
End Sub
```

The procedure sets up the headings that you'll use in the pivot table, then keeps loading Visual Basic project files into a sheet until the LoadProjectFile procedure returns False. The procedure then calls a subroutine to create the pivot table. I always start my highest-level procedures in this simple manner. It's almost like creating pseudocode that shows exactly what the procedure will do. Then I write the subroutines to do the real work. I find that this approach is self-documenting and makes it easy for another programmer to pick up and understand.

The SetUpHeadings procedure is nearly as easy to code. It uses the Selection object of the active sheet to assign the column headings. Notice that you use the Offset method to move to the right to fill in the second and third column headings, and to move the selection back to the first row under the first column heading:

```
Sub SetUpHeadings( )
Selection = "Project"
Selection.Offset(0, 1) = "Type"
Selection.Offset(0, 2) = "File"
Selection.Offset(1, 0).Select
End Sub
```

The LoadProjectFile procedure needs to display a common dialog to locate the MAK or VBP project files. Use the GetOpenFilename method of the Application object to display this dialog. Call the LoadVB procedure to load the information from the Visual Basic project file into a Files collection and put the information from the Files collection into the sheet. Notice that the first For…Next loop steps through the Files collection, while the second For…Next loop steps through the array of detail information for each file:

VB5

```
Type=Exe
Object={00028C01-0000-0000-0000-000000000046}#1.0#0; _
  DBGRID32.OCX
Reference=*\G{00020430-0000-0000-C000-_
  000000000046}#2.0#0#C:\WINDOWS\SYSTEM\STDOLE2.TLB#_
  Standard OLE Types
Reference=*\G{00025E04-0000-0000-C000-_
  000000000046}#3.5#0#C:\PROGRAM FILES\COMMON _
  FILES\MICROSOFT SHARED\DC:\PROGRAM FIL#Microsoft DAO _
  2.5 Object Library
Reference=*\G{1187DF4B-6F23-11D0-97D5-_
  444553540000}#1.0#0#C:\PROGRA~1\COMMON~1\OLESVR\_
  SOTSERVE.EXE#SOTTimeServer
Reference=*\G{7FE727A3-7059-11D0-A2A3-_
  00C04FC58652}#1.0#0#C:\SUNOP\CRCSERVE.EXE#CRCServer
Object={F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.1#0; _
  COMDLG32.OCX
Object={B02F3647-766B-11CE-AF28-C3A2FBE76A13}#2.5#0; _
  SS32X25.OCX
Reference=*\G{B3F42A46-0F3D-11D1-B44F-_
  444553540000}#1.0#0#C:\VB\APPS\VBPJ\1997\VBPJ9711\_
  SOTMAPI.dll#SOTMAPI
Form=FrmHM.FRM
Module=DEBUG3; ..\DEBUG3.BAS
Module=UNC32; ..\UNC32.Bas
Module=HMBLD; HMBld.bas
Module=OBJOB; ..\Objob.bas
Module=SOTDTS32; ..\Sotdts32.bas
Module=Module2; ..\Obord3.bas
Module=Module3; ..\Obcons3.bas
Module=SunOp32; ..\Sunop32.BAS
Module=OBMASTER; ..\Obmaster.bas
Module=PRICING; ..\Pricing.bas
Module=OBOrd32; ..\OBOrd32.bas
Module=OBStock32; ..\Obstock32.bas
Form=..\Aboutbox32.frm
Module=Spread32; ..\Spread32.bas
Module=MAPI32; ..\Mapi32.bas
IconForm="frmMain"
Startup="frmMain"
HelpFile=""
Title="HMBuild"
ExeName32="HMBuild.exe"
Command32=""
Name="HMBuild"
HelpContextID="0"
CompatibleMode="0"
MajorVer=3
MinorVer=3
RevisionVer=9
AutoIncrementVer=1
ServerSupportFiles=0
VersionCompanyName="SunOpTech®"
VersionLegalCopyright="© 1997 SunOpTech, Ltd."
VersionLegalTrademarks="SunOpTech®"
CompilationType=-1
OptimizationType=0
FavorPentiumPro(tm)=-1
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
ThreadPerObject=0
MaxNumberOfThreads=1
```

LISTING 2 ***Typical Visual Basic 5 VBP File Format.*** *The VB5 VBP file contains a list of the GUIDs, paths, and file names for the forms, modules, classes, references, and OCXs contained in the project.*

```
Function LoadProjectFile() As Boolean
Dim File, FileDet
File = Application.GetOpenFilename(_
   FileFilter:="VB Project Files _
   (*.mak;*.vbp), *.mak;*.vbp", _
   Title:="Load VB Project File", _
   ButtonText:="Load")
If File <> False Then
   Project = File
   If LoadVB Then
      'put in spread
      For Each File In Files
         For Each FileDet In File
            Selection = FileDet
            Selection.Offset(0, _
               1).Select
         Next
         Selection.Offset(1, -3).Select
      Next
   End If
   LoadProjectFile = True
End If
End Function
```

The LoadVB procedure opens the selected file and determines whether the file is a VB4 or VB5 project file by checking for an equals sign in the first line (see Listing 1). Then it calls either the LoadVB5 or LoadVB3 procedure. You can see how each of these procedures uses a Case statement to identify the type of file. The LoadVB3 procedure uses the ParseString function to return the string following the period so it can identify the file as FRM, BAS, or VBX. Then the LoadVB3 procedure adds the file to the Files collection using an Array statement to create a variant array of the project name, file type, and file path. The procedure loops until the file is completely read:

```
Private Sub LoadVB3(FileNum%)
Dim txt$
Do While Not EOF(FileNum)
   Input #FileNum, txt
   txt = LCase(txt)
   Select Case ParseString(txt, ".", 2)
   Case "frm"
      Files.Add Array(Project, _
         "Form", txt), txt
   Case "bas"
      Files.Add Array(Project, _
         "Module", txt), txt
   Case "vbx"
      Files.Add Array(Project, _
         "Control", StripPathName_
         (txt)), txt
   End Select
Loop
End Sub
```

The LoadVB5 procedure, on the other hand, parses the string read from the file at the equals sign to determine the file type. Inside the appropriate Case statement, it parses differently depending on the format of that type of line:

```
Private Sub LoadVB5(FileNum%)
Dim txt$, itm$
Do While Not EOF(FileNum)
   Input #FileNum, txt
   txt = LCase(txt)
   itm = ParseString(txt, "=", 1)
   txt = ParseString(txt, "=", 2)
   Select Case itm
   Case "object"
      Files.Add Array(Project, _
         "Control", ParseString(txt, _
         ";", 2)), txt
   Case "reference"
      Files.Add Array(Project, _
         "Reference", StripPathName(_
         ParseString(txt, "#", _
         4))), txt
   Case "module"
      Files.Add Array(Project, _
         "Module", ParseString(txt, _
         ";", 2)), txt
   Case "form"
      Files.Add Array(Project, _
      "Form", txt), txt
   End Select
```

```
Loop
End Sub
```

CREATE A PIVOT TABLE

Now you've finished the first two functions for the main CreateVBProjCrossRef procedure. The user has selected the Visual Basic project files, and the data has been loaded into a worksheet. Next, you need to create a pivot table from this data. Excel pivot tables are extremely powerful, but can be tedious to create from code. I always use the Recorder as I create the pivot table using the Pivot Table Wizard from the Data menu. Then I modify the code to operate in a generic manner. This code was created by the Recorder; you can see that the first line uses a reference to a specific range of 16 rows:

```
Sub RecordedPivotTableMacro()
ActiveSheet.PivotTableWizard SourceType:=xlDatabase, _
   SourceData:="Sheet1!R1C1:R16C3", _
   TableDestination:="", TableName:="PivotTable1"
ActiveSheet.PivotTables("PivotTable1")._
   AddFields RowFields:=Array("Type", _
   "File"), ColumnFields:="Project"
ActiveSheet.PivotTables("PivotTable1")._
   PivotFields("File").Orientation = xlDataField
End Sub
```

To make this procedure useful for any number of rows, you need to add only two lines of code to move the selection up one row. Use the CurrentRegion method of the Selection object to select all the contiguous columns and rows. You can change the SourceData argument of the PivotTableWizard to use the Selection as the range:

```
Sub CreatePivotTable()
Selection.Offset(-1, 0).Select
Selection.CurrentRegion.Select
ActiveSheet.PivotTableWizard _
   SourceType:=xlDatabase, SourceData:=Selection, _
   TableDestination:="", TableName:="PivotTable1"
ActiveSheet.PivotTables("PivotTable1")._
   AddFields RowFields:=Array("Type", _
   "File"), ColumnFields:="Project"
ActiveSheet.PivotTables("PivotTable1")._
   PivotFields("File").Orientation = xlDataField
End Sub
```

Try out your new procedure on some of your Visual Basic projects. You can see how easy it is to use the power of Excel to help manage your Visual Basic projects. ☒

Code Online

You can find all the code published in this issue of VBPJ *on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.*

Track Component Use with Excel VBA

Locator+ Codes

Listings ZIP file plus GSJA98.XLS, which contains the Visual Basic procedures to process your project files (free Registered Level): VBPJ0198
Listings for this article plus GSJA98.XLS (subscriber Premier Level): GS0198P