Visual Basic 4.0 Technical Articles

Creating 16-Bit and 32-Bit Screen Savers with Visual Basic

Ken Lassesen
Microsoft Developer Network Technology Group

June 29, 1995

Abstract

This article describes how to create 16-bit and 32-bit screen savers for Microsoft® Windows® version 3.1, Windows NT™, or Windows 95 using Visual Basic® version 4.0. The ScrSaver sample that accompanies this article uses a moving picture box to create a screen saver. The reader is assumed to have passed Microsoft Certification in Microsoft Visual Basic for Windows version 3.0 or have equivalent knowledge in making Windows API calls from Visual Basic.

Introduction

Creating a screen saver is not a useful activity for most programmers. The technical information can rarely be reused and there seems to be no practical business need. It is an ego trip. There, now that I've said it, we can get on with creating a screen saver that does whatever you—or your boss—want. I have added a practical twist to the screen saver by enabling it to be an OLE server, an interesting enhancement that gives it a practical use.

If you want to create a screen saver using an MFC framework, read Nigel Thompson's "Creating 32-Bit Screen Savers with Visual C++ and MFC." But what if you want to use Visual Basic® to create your masterpiece? At first glance, it looks tough because you cannot link in Visual Basic to libraries such as the SCRNSAVE library in the Win32® Software Development Kit (SDK).

A more in-depth look at the Win32 SDK documentation reveals that we also apparently need to export certain functions from our application in order for it to be installed and run from the Control Panel desktop applet. This all sounds like it may be impossible to do from Visual Basic 4.0.

Well, as it turns out, we do not need to use the SCRNSAVE library at all. Creating a screen saver is simply a matter of creating a regular application that:

- Has some command-line parsing.
- Assigns a special application title and window style.
- Makes some application programming interface (API) calls when started.
- Detects user input.
- Has .SCR as its file extension.
- Has a configuration dialog box for the screen saver.

Given those few simple requirements, creating a screen saver with Visual Basic does not seem to be such a tough job after all, does it?

Creating the BASIC Framework

Yes, that's BASIC as in "Beginners All-purpose Symbolic Instruction Code," not basic as in "fundamental." Instead of writing out the procedures for you, I will outline the steps you need to take to fulfill the requirements listed in the previous section for creating a screen saver application. I will design the application around "developer-protected" objects. (A real object is compiler-protected.)

The first object, MyApp, is inherited from the App object in Visual Basic. The second object employs a user-defined

data type (UDT) called udtScreenSaverConfig and handles all the information regarding the screen-saver configuration. The third object, frmScreenSaver, handles the actual display of the screen saver and queries the udtScreenSaverConfig object for information. The application itself can be viewed as a fourth object, AppScreenSaver, which connects all of these parts together. These relationships are illustrated in Figure 1, using Booch notation (see the Bibliography listing for an explanation of Booch notation).

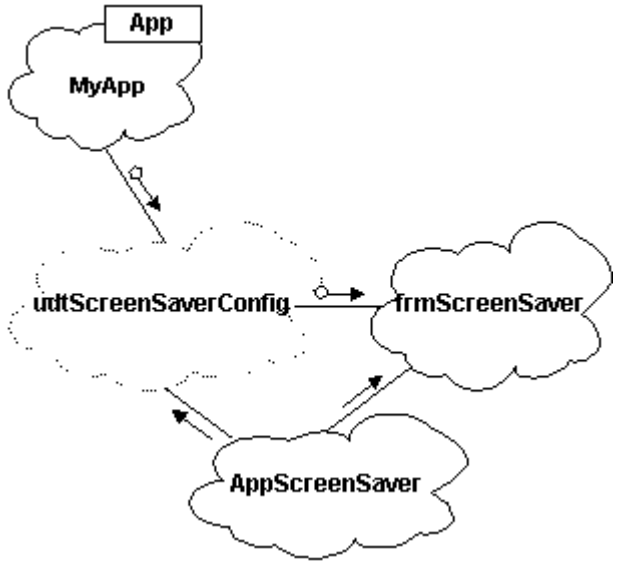

Figure 1. Logical objects of the ScrSaver project

I will describe these objects with one or two sample procedures (either methods or properties) and leave the reader to examine the full implementation in the associated samples. We will start with MyApp because it is used by the other objects.

> Note In the C++ world, objects and their methods and properties are represented by Object::Property or Object::Method. This syntax cannot be implemented with Visual Basic products. Instead, I used the form Object_Property or Object_Method.

The MyApp Object

The MyApp object inherits information from the App object in Visual Basic. The MyApp object is an expansion of the properties of App, which can be used in all projects. Table 1 shows some public properties that MyApp can have.

Table 1. Properties and Methods of the MyApp Object

Property or Method	Description
INIFile	Location of .INI file for application. That is, the value of App.ExeName with ".INI" added at the end of the string.
About	Information about the application in a message box.
Init	Initializes MyApp.
SysInfo	Information about the system in a message box.
GetSectionParm	Returns the value of the specified parameter in a section of INIFile.
PutSectionParm	Writes the value of the specified parameter in a section of INIFile.
GetParm	Returns the value of the specified parameter in the [INIT] section of INIFile.
PutParm	Writes the value of the specified parameter in the [INIT] section of INIFile.

The MyApp object's code is kept in the MYAPP module file and may be copied as a module from project to project. I use a message box instead of additional forms in MyApp_About and MyApp_SysInfo to keep the number of files and amount of code smaller.

The udtScreenSaverConfig Object

The udtScreenSaverConfig object is based on a public UDT of the same name. When a UDT is used, the object may have multiple instances with each instance having its own dimensioned space. The UDT is an argument to all of the procedures of the object because it contains all the data about the object. Table 2 shows some public properties that this object can have.

Table 2. Public Properties of the udtScreenSaverConfig Object

Property	Description
Get	Retrieves values from the associated .INI file (uses MyApp_GetParm).
Write	Writes values to the associated .INI file (uses MyApp_PutParm).
Edit	Displays the values on the frmudtScreenSaverConfig form.
Init	Sets the default values of the UDT.

The udtScreenSaverConfig object's code is kept in the UDTSSC module file and the FRMUSSC form. This object is Visual Basic–specific and not usable in other Visual Basic for Applications products.

The frmScreenSaver Object

The frmScreenSaver object is based on a form of the same name and not on a public UDT. The form may contain a private UDT, as it does in this case. This object is a single-instance object that is created when the form is referenced, and does not expose the UDT. The frmScreenSaver.Form_Load procedure serves as the initialization of the instance and uses udtScreenSaverConfig_Get. The frmScreenSaver.Form_Unload procedure serves as the destroyer of the instance.

The frmScreenSaver object has one public component, frmScreenSaver_Init. The frmScreenSaver_Init procedure starts the screen saver. The controls on a form should never be referenced directly if the form is part of an object, except by procedures that are part of the object (none in this case). This object contains a second form, frmScreenSaverPassword, which adds password protection to the screen saver.

The frmScreenSaver object consists of FRMSS.FRM and FRMSSPW.FRM.

The AppScreenSaver Object

The AppScreenSaver object connects the above objects into an application. Instances of each object are created by AppScreenSaver. The only public procedure of this object is AppScreenSaver_Init, which is called from the startup Main subroutine.

Details of the Objects

Before we dive into the details of the objects, we should glance quickly at Figure 2, which depicts the relationships between the physical files and the logical objects and shows the critical procedure calls between the files.

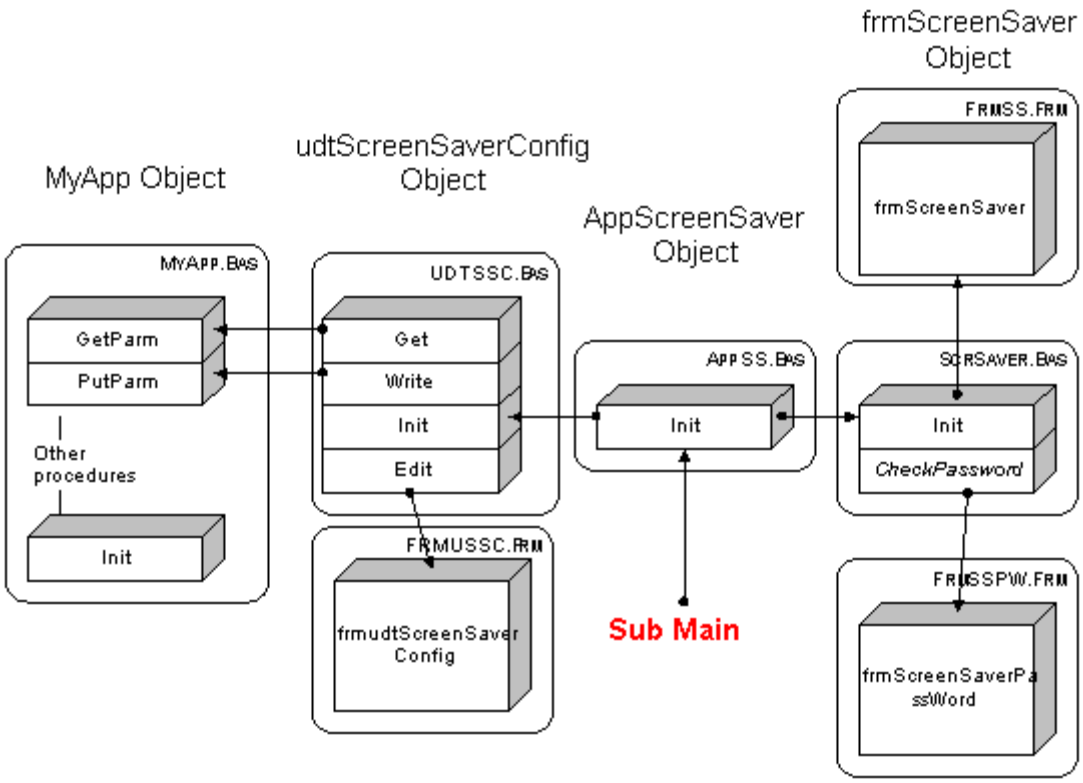

Figure 2. Physical structure of the ScrSaver project

Details of the udtScreenSaverConfig Object

The udtScreenSaverConfig object contains all of the configuration information about the screen saver. For our sample, we will assume that the screen saver will display an image on the screen that will move back and forth. The information needed is stored in a UDT called udtScreenSaverConfig, which I define as follows:

```
Type udtScreenSaverConfig
   BackgroundImageFileName as String  'May be Null
   BackgroundImageColor as Long 'May be 0
   MovingImageFileName as String 'May be Null
   ImageSpeed as Integer 'Min 1
   ImageBehaviour as Integer
   PassWord as String
End Type
```

This information is read and written to MyApp_INIFile by MyApp_GetParm and MyApp_PutParm. The code is trivial, but the following code sample shows a private procedure in the udtScreenSaverConfig object. The Validate procedure verifies that the values are correct. Because this procedure is private, there is no need to prefix it with the object name.

```
Sub udtScreenSaverConfig_Get (MyUDT as udtScreenSaverConfig)
   MyUDT. BackgroundImageFileName = MyApp_GetParm ("BackgroundImageFileName")
   MyUDT. BackgroundImageColor = MyApp_GetParm ("BackgroundImageColor")
   MyUDT. MovingImageFileName = MyApp_GetParm ("MovingImageFileName")
   MyUDT. ImageSpeed = Val(MyApp_GetParm ("ImageSpeed"))
   MyUDT. ImageBehaviour = Val(MyApp_GetParm ("ImageBehaviour"))
   MyUDT. PassWord = MyApp_GetParm ("PassWord")
   Validate MyUDT   'This is a PRIVATE procedure.
End Sub
```

A second procedure, udtScreenSaverConfig_Edit, requires a little explanation. This procedure must display a

form with the information passed in the UDT, then update the UDT if needed (the user could cancel). The trick is to store the UDT data in the module temporarily while the form loads and the UDT data is read to the form. When the user clicks OK, the data is again stored temporarily in the module. This approach makes the module strictly independent of the form, allowing other forms to use the same module without any code changes. The code is shown below.

```
Sub udtScreenSaverConfig_Edit (MyUDT as udtScreenSaverConfig, Aform as Form)
   udtScreenSaverConfig_fCacheUDT UDT
   Aform.Show 1 'Modal: Any form may be used.
   udtScreenSaverConfig_fRecoverUDT UDT
   Validate MyUDT   'This is a PRIVATE procedure
End Sub
```

We have two procedures that I call friends: udtScreenSaverConfig_fRecoverUDT and udtScreenSaverConfig_fCacheUDT. A friend procedure must be declared as public in Visual Basic so that procedures in other modules can call it, but it is not intended to be called by any procedure. The lower case f indicates that it is a friend (the developer must protect friends instead of the compiler protecting them). The code for these two friend procedures is simple and is shown below.

```
Sub udtScreenSaverConfig_fCacheUDT(MyUDT as udtScreenSaverConfig)
localUDT = MyUDT   'localUDT is DIM as udtScreenSaverConfig in Declarations
End Sub
Sub udtScreenSaverConfig_fRecoverUDT(MyUDT as udtScreenSaverConfig)
MyUDT = localUDT 'localUDT is DIM as udtScreenSaverConfig in Declarations
End Sub
```

To tie these procedures together, we must examine the Form_Load event and the pbOK_Click event on the form. Form_Load calls udtScreenSaverConfig_fRecoverUDT to retrieve the values of the UDT and then fill the controls on the form. pbOK_Click retrieves the values of the controls and then calls udtScreenSaverConfig_fCacheUDT, so the data can be retrieved after the form unloads. Figure 3 shows the UDT data movement, again using Booch notation.

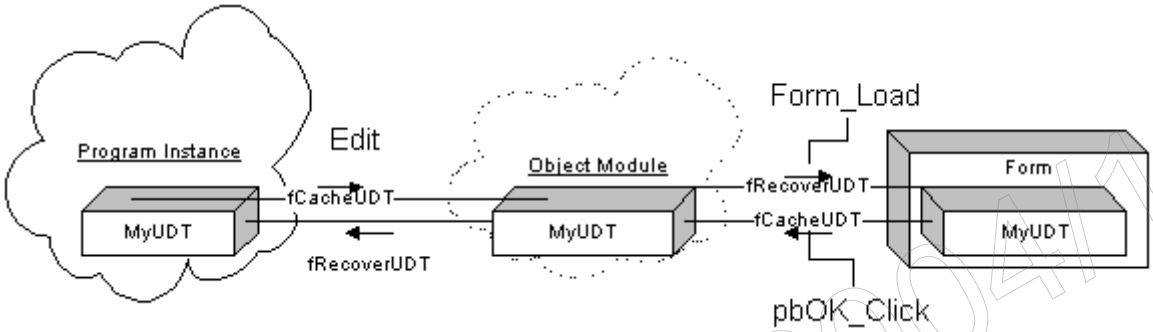

Figure 3. UDT data movement in udtScreenSaverConfig_Edit

Details of the frmScreenSaver Object

The frmScreenSaver object displays a form that is the screen saver. The activity that the screen saver performs is immaterial to the reader. All screen savers use basically the same style of window—a form that is the full size of the screen, has no borders or caption, has no visible mouse, and is on top of all the other windows. This window exists so long as the user does not touch the mouse or hit a key.

The frmScreenSaver object has a single public procedure, frmScreenSaver_Init. This procedure, the properties of the frmScreenSaver form, and API calls to SetWindowPos and SetCursor are needed to create the screen saver's window style.

The frmScreenSaver_Init procedure shows the frmScreenSaver form modal. The frmScreenSaver properties are set as shown below.

```
BorderStyle     =   0  'None
ControlBox      =   0  'False
MaxButton       =   0  'False
MinButton       =   0  'False
WindowState     =   2  'Maximized
```

> Note The Caption property is set to "" and does not appear in the saved form. (If it is there, delete the text.)

The calls to SetWindowPos (on top of other windows) and SetCursor (no cursor) are done in frmScreenSaver.Form_Load. Examine the sample for further details.

Destroying the frmScreenSaver Object

The instance of the frmScreenSaver object must be destroyed upon any user input. This is done by calling pDestroy when any of the following events occur:

- Form_Click
- Form_DblClick
- Form_KeyDown
- Form_KeyPress
- Form_MouseDown
- Form_MouseMove

> Note The Form_MouseMove event requires special handling. Many mice generate small random movements due to vibrations. To prevent accidental termination of the screen saver, some minor or slow movement of the mouse must be permitted. Examine the sample code for further details.

The Destroy procedure does whatever activities are needed before the screen saver terminates. In our sample application, the password is checked, the cursor is restored, and the form is unloaded. Because the events above are not mutually exclusive, we may have multiple calls to Destroy, which can create errors. (For example, if you make three attempts to Unload the same form, the first one will succeed and the subsequent calls will generate errors.) The simplest way to handle this is to ignore all calls until the first call is completed, by using a static variable as shown in the code below.

```
Public Sub frmScreenSaver_Destroy()
Static Once%  ' If multiple calls occur, we want to discard subsequent
              ' calls until the first one completes.
If Once% Then Exit Sub
Once% = True
REM  <Your code here>
.
.
.
DoEvents ' Allow all pending messages to be bounced first.
Once% = False
End Sub
```

Details of the AppScreenSaver Object

The AppScreenSaver object connects the objects shown in Figure 2. AppScreenSaver_Init is called by the Main subroutine, which then creates the needed instances of the objects depending on the command-line arguments. The command-line arguments are returned by the Command function. Table 3 lists the values that should be supported and what they mean, and Figure 1 shows the program logic implementing it.

Table 3. Command-Line Parameters for Screen Savers

Parameter	Meaning
/s, -s or s	Start in screen saver mode.
/c, -c or c	Show configuration dialog box with whatever window is currently active as the parent window.
(none)	Show the configuration dialog box with no parent window.

Note The purpose of the /c option is to disable the application that invoked the screen saver (for example, the Control Panel) while the screen saver's Setup dialog box is active. This is necessary so that the user cannot click the Setup button in the Control Panel and start running another copy of the screen saver. We prevent this in Visual Basic by checking for an instance of App.PrevInstance and keeping the configuration dialog box on top.

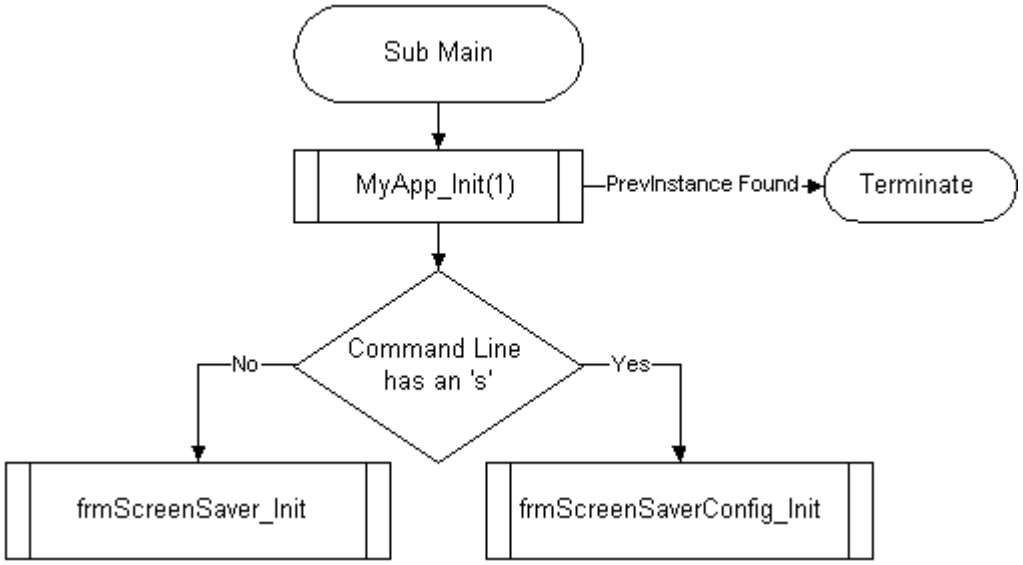

Figure 4. Startup logic of loading the screen saver application

The code for the AppScreenSaver_Init application simply creates the appropriate objects based on the command-line arguments.

```
Public Sub AppScreenSaver_Init()
MyApp_Init 1    ' Single instance only
If InStr(LCase$(Command$), "s") Then ' Contains {'s','S'}?
    Call frmScreenSaver_Init
Else
    Dim MyUDT As udtScreenSaverConfig  ' Create the object
    udtScreenSaverConfig_Get MyUDT
    udtScreenSaverConfig_Edit MyUDT,frmudtScreenSaverConfig
    udtScreenSaverConfig_Put MyUDT
End If
End Sub
```

That's all, folks, for the regular part of coding. There are a few special steps left to do that we will look at next.

The Fine Print

One of the final details in getting your screen saver up and running is to provide the description string in the application title that the Control Panel will insert in the screen saver's list in the desktop applet. This string must start with SCRNSAVE:. We must also change the extension to .SCR; this is done by choosing Make EXE File from the File menu and clicking the Options button to bring up the EXE Options dialog box. In the Title text box under Application, type SCRNSAVE: followed by a title, as shown in Figure 5.

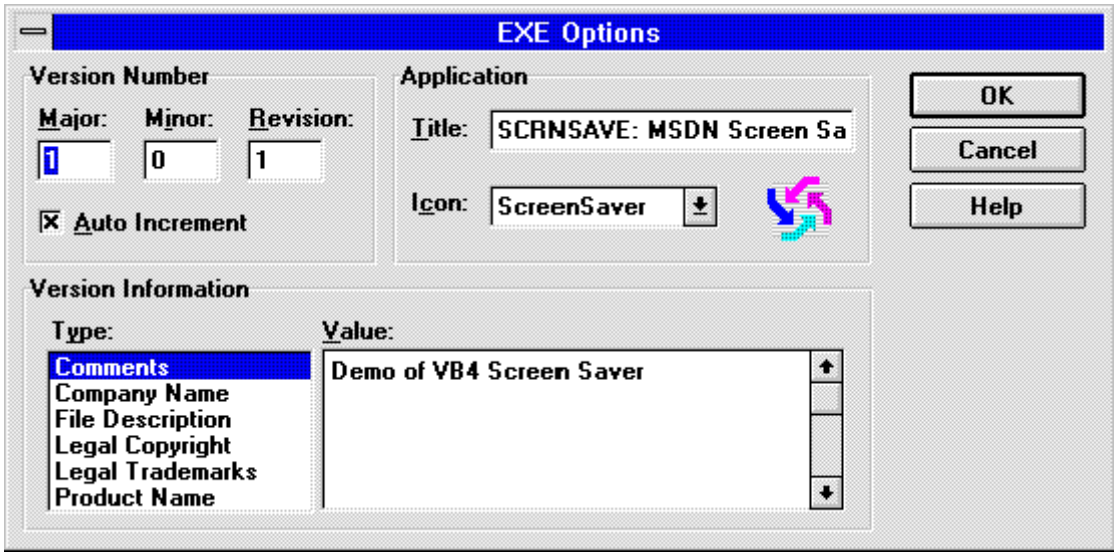

Figure 5. Naming the screen saver in Visual Basic 4.0

When you click OK, the Make EXE File dialog box will reappear. Under File Name, type the EXE name, but change the extension to .SCR. as shown in Figure 6.

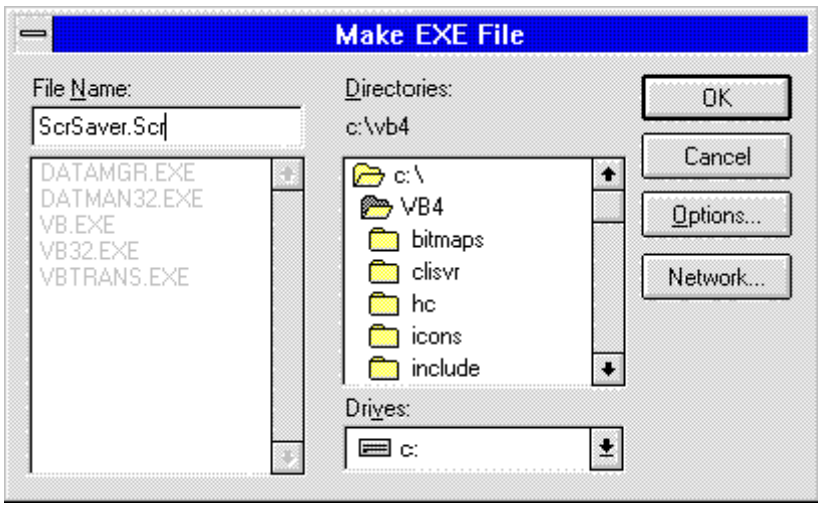

Figure 6. Making the SCR file

Making a 32-Bit Screen Saver

There are two differences between 16-bit screen savers and 32-bit screen savers in Visual Basic. First, the arguments for the 16-bit API calls are different from those for the 32-bit calls. (See my article "Porting Your 16-Bit Office-Based Solutions to 32-Bit Office.") Second, the 32-bit screen saver title is obtained from the first string (string ID 1) in the application string table. Adding this string into the compiled 32-bit executable requires a resource editor that can handle 32-bit executables created by Visual Basic and must be done after creating the 32-bit executables.

Tips and Tricks

An application that places a form that occupies the entire screen and is topmost is a pain to debug. If you put any breakpoint in your application, you will be facing a blank screen and a lot of beeps. Pressing CTRL+B for the debug window will show nothing because the debug window is below your screen-saver form. Instead, blindly type End to escape. If an error does occur, do not expect to be able to see it—the message and code will be hidden beneath

your screen-saver form. If this does not frustrate you enough, imagine not having a cursor visible to indicate where you are clicking. To prevent situations like these, I suggest that during most of development process you:

- Leave the caption bar and control box visible.
- Do not call SetWindowPos.
- Do not call ShowCursor.

A Practical Use

At the start of this article, I stated that screen savers seem to have no practical use. In fact, they have some very practical uses if they are also OLE servers. An OLE server can be used from most Microsoft® Office products to provide functionality not available in the product. An OLE screen saver allows you to blank out the screen and protect your application or solution while it is performing calculations or queries. This protection can be very useful in Microsoft Office applications and in Microsoft Access, which take a long time to process data. If you examine the code for the ScrSaver sample, you will find that the screen saver has been implemented as a simple OLE server with the following verbs:

- Create: Starts the screen saver.
- Destroy: Terminates the screen saver from the client application.
- Status: Allows information to be displayed on the screen saver from the client application.
- SetPassWord: Allows the client to set a password for the current session.

The References and Object Browser dialog boxes are shown below. The server class is LockScr.ScrSaver.

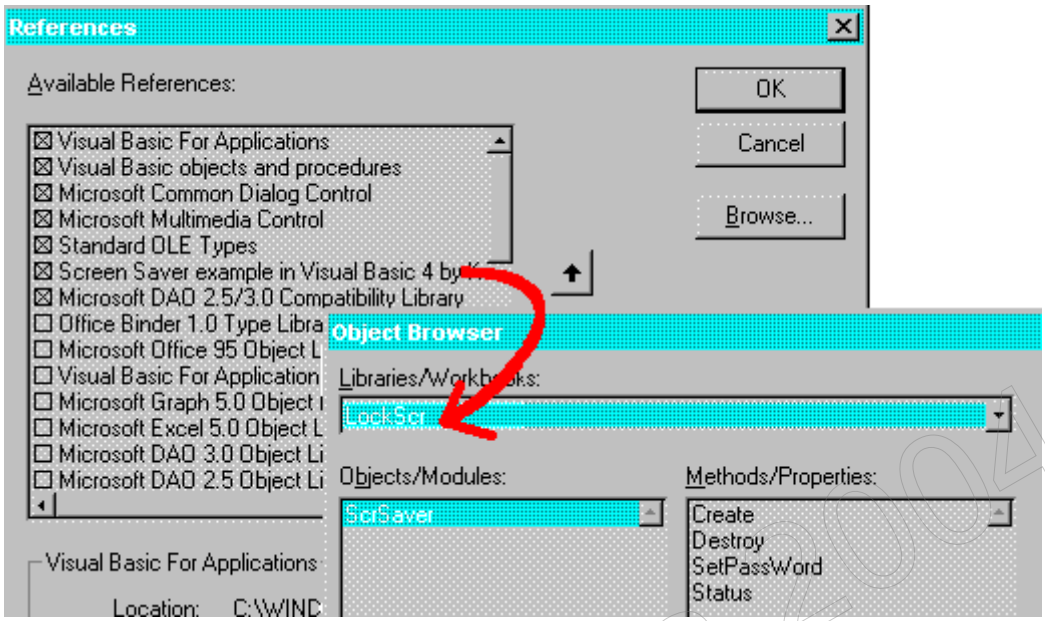

Figure 7. An OLE screen saver selected in the References dialog box and its corresponding appearance in the Object Browser

The use of the screen saver from any product that supports CreateObject is shown below.

```
Sub Demo()
Dim LockScr As Object
Set LockScr = CreateObject("LockScr.ScrSaver")
LockScr.Create
test = Now
While (test - Now) < 1# / (14400#) '6 Seconds
    Debug.Print Now
    LockScr.Status = Time$
    DoEvents
```

```
Wend
LockScr.Destroy
End Sub
```

Summary

This article has described how you can build 16-bit and 32-bit screen savers using Visual Basic, and the code in the sample application provides a framework you can easily enhance. The sample that accompanies this article bounces a picture box off the four sides of the monitor by changing the coordinates of the control—a boring sample that you are sure to improve. You can become very creative in designing your screen savers. If you care to show me your masterpiece, e-mail it to me at KENL@MICROSOFT.COM.

Bibliography

Booch, Grady. Object-Oriented Analysis and Design with Applications. Redwood City, CA: Benjamin/Cummings Pub. Co., 1994.

Knowledge Base Q96780. "Security and Screen Savers."

Knowledge Base Q126239. "PRB: Win32-Based Screen Saver Shows File Name in Control."

Lassesen, Ken. "Porting Your 16-Bit Office-Based Solutions to 32-Bit Office." (Development Library, Technical Articles)

Lassesen, Ken. "Corporate Developer's Guide to Office 95 API Issues." (Development Library, Technical Articles)

Microsoft Win32 Software Development Kit (SDK), "Screen Saver Library." 1995 (MSDN Library, Platform, SDK, and DDK Documentation)

Microsoft Windows version 3.1 Software Development Kit (SDK) Programmer's Reference, Volume 1: Overview. 1987-1992. See Chapter 14, "Screen Saver Library." (MSDN Library Archive, Product Documentation, SDKs, Windows 3.1 SDK)

Ragsdell, Blake. "Creating Your Own Screen Savers." Inside Visual Basic for Windows (April 1994). (MSDN Library, Periodicals)

Thompson, Nigel. "Creating 32-Bit Screen Savers with Visual C++ and MFC." (MSDN Library, Technical Articles)