Issue Date: FoxTalk June 1998

Tame Your ActiveX Controls

Doug Hennig
dhennig@stonefield.com

ActiveX controls are forward, but not backward, compatible. This means that if you install a newer version of an OCX on your system, any applications you ship to people who have an older version installed will break. This article presents a reusable tool that solves this problem in a simple, easy-to-use manner.

How was the period between Christmas and New Year's for you? I hope it was relaxing. For me, it was awful. The reason: I posted an update to Stonefield Database Toolkit (SDT) on our Web site just before Christmas. Anyone who downloaded and installed it immediately got an "OLE class not registered" error when trying to use SDT. I fielded a ton of support calls and e-mails the week after Christmas and had to come up with a fast solution to solve the problem for those folks and prevent even more calls later.

What caused this problem? The main SDT form uses a TreeView control, and I had installed some software on my system that installed a newer version of COMCTL32.OCX, the OCX file containing the TreeView control. Even though I didn't change the TreeView on the form, simply opening the form and saving it caused information about the newer control to be written to the form. Of course, very few others had this newer control on their machines, so even though the form wanted a TreeView control and they had a TreeView control, it wasn't the exact control the form was looking for. Hence, the "OLE class not registered" error.

Is it just me, or do you think this is incredibly lame too? I don't care about different versions of the TreeView. I'd understand if I used new properties or methods of this control that didn't exist in older versions, but as I mentioned, I didn't even touch it in my copy of the form.

I guess I only have myself to blame for being in crisis mode after Christmas. After all, I'd encountered this behavior before when I upgraded from VFP 5.0 to 5.0a; it came with an updated COMCTL32.OCX, and everyone who hadn't upgraded to 5.0a got "OLE class not registered" errors when they tried to use SDT. My workaround at the time was to keep one machine in our office with VFP 5.0 (not 5.0a) installed and do any SDT development involving that one form with the TreeView on it on that one machine. If I ever forgot and modified the form on my machine...

However, knowing I'd experienced the problem before didn't make me feel any better when my Inbox had dozens of e-mails reporting the problem. That was the last straw; I had to come up with a permanent solution to this problem. The tool described in this article is the result.

Background

First, let's look at the cause of the problem. If you open a form containing an ActiveX control as a table (in other words, USE MYFORM.SCX) and browse it, you'll find binary information stored in the OLE field of the record for the ActiveX control. This binary information appears to contain some version-specific information about the control. As a result -- at least in the case of the ActiveX controls shipped by Microsoft -- ActiveX controls are forward but not backward compatible. This means that if I drop version 2.01 of an ActiveX control on a form, people with a newer version of the ActiveX control (such as version 2.02) can open the form, but those with an older version (like version 2.00) can't.

Let's dig a little deeper and see how ActiveX controls are handled on a system. As you're probably aware, simply copying an OCX file to someone's hard drive doesn't allow them to use the ActiveX controls contained in that file. The controls need to be registered on the system either using REGSVR32.EXE (a copy of this program comes with VFP) or letting VFP's Setup wizard handle it for you. Using REGEDIT, we can see that ActiveX controls are registered by a class ID value in HKEY_CLASSES_ROOT\CLSID. For example, the TreeView control is registered in HKEY_CLASSES_ROOT\CLSID\0713E8A2-850A-101B-AFC0-4210102A8DA7. There are several subkeys under this key, the most important ones (as far as we're concerned) being ProgID, VersionIndependentProgID, and InProcServer32. ProgID contains the program ID, which might change from version to version -- for example, on my system, the value of this subkey for the TreeView is COMCTL.TreeCtrl.1. VersionIndependentProgID contains just what the name implies (on my system, the value is COMCTL.TreeCtrl for the TreeView), and InProcServer32 contains the name and path of the OCX containing the control. This immediately suggests several potential problems:

- The ActiveX control was never installed on a machine -- there's no entry in the Registry for the control, and the OCX doesn't exist on the drive. The solution is to properly install it.
- The OCX file was copied to the system but not registered -- the file exists, but there's no entry in the Registry. The solution is to use REGSRV32.EXE to register the ActiveX controls in the OCX. *Hint:* You must include the extension for the file you're registering:

```
"regsvr32 myfile.ocx" instead of "regsvr32 myfile"
```

- The ActiveX control is registered, but the OCX can't be found -- it might have been moved or renamed, or the user might have restored from a backed-up Registry but didn't restore all files. The best solution is to reinstall and register.
- The version installed on the system is different than the one required by an application. While an obvious solution to this problem is to install the version of the control required, this might not always be feasible. For example, if I shipped my copy of COMCTL32.OCX along with SDT, that would get rid of my problems but would probably cause a whole lot of problems for you and any clients you'd created forms with a TreeView for.

Another unforeseen problem can also occur. Some ActiveX controls look for a license (a key stored in the Registry or a file on your drive) in order to use them in a development environment. This makes sense; it allows you to develop and distribute applications using a vendor's controls and have those controls work in a runtime environment while protecting the vendor from someone else being able to develop using those controls without paying for them. The problem this causes, though, is that if the license key gets lost on your system, you'll get a "License not found" error when you try to drop the control on a form. Reinstalling doesn't always solve the problem. Based on messages I've seen on CompuServe and the Universal Thread, this problem has been a common one. For the controls shipped with VFP, the solution is simple: Download a file called FOX.REG from the Universal Thread (there's a great reason to join right there) and run it; it will register the proper licenses on your system. For other third-party controls, reinstall or contact your vendor.

As you can see, most of the problems regarding ActiveX controls require reinstalling them. However, as I mentioned, it might not be possible to solve the version problem this way. The rest of this article discusses an easy-to-implement solution to this annoying problem.

SFActiveX

The solution is to instantiate ActiveX controls at runtime rather than using them at development time. This way, the control is instantiated from the version installed on the user's system rather than looking for the version that was installed on your system. To make this job easier, I created an ActiveX loader class called SFActiveX.

SFActiveX is based on SFCustom, our Custom base class defined in our base class library SFCTRLS.VCX. It has the custom properties shown in **Table 1**.

Table 1. Custom properties of SFActiveX.

Property	Description
cClass	The name of the class defining the ActiveX control or a subclass of it
cClassID	The Class ID for the object
cLibrary	The name of the library containing a subclass of the ActiveX control
cNewObjectName	The name to give the ActiveX control this object will create
cObjectName	The name of the placeholder object to replace with the ActiveX control
cOLEClass	The OLE class for the object

cClass defaults to "OLEControl" because that's the VFP class that ActiveX controls are contained in. However, if you want to use a subclass of an ActiveX control (I'll show at least one reason why you'd want to do that later in the article), enter the name of the class in this property and the name of the library file containing the class definition in cLibrary (specify an extension to identify VCX from PRG libraries).

You can specify the ActiveX control to use in one of two ways: by cClassID or cOLEClass. If you specify one, leave the other blank. cClassID is the class ID for the ActiveX control, and cOLEClass is its ProgID. How do you know what to enter for these properties for a given control? For the cOLEClass, the simplest way is to drop the desired control on a form, then look at its OLEClass property. For example, the OLEClass for the TreeView control is "COMCTL.TreeCtrl" (you might see "COMCTL.TreeCtrl.1" in your instance; however, don't worry about the ".1" for cOLEClass). cClassID is a little harder to determine. You'll need to fire up REGEDIT and search for the OLEClass for the control, then see what class ID it appears under. For example, the TreeView control appears under "{0713E8A2-850A-101B-AFC0-4210102A8DA7}". You don't need to enter the curly braces when you enter this value into cClassID. It's better to specify cClassID than cOLEClass because the name of the control might change when different versions are installed (for example, the TreeView control that shipped with VFP 5.0a has a ProgID of "COMCTL.TreeCtrl.1", which could cause problems under certain circumstances if you specified "COMCTL.TreeCtrl").

Some ActiveX controls have a visual appearance, and others don't. For those that do, you'll need to specify the location and size of the control. You could do it in the Init of the form by setting the Top, Left, Width, and Height properties in code, but I find it more intuitive to put a Shape object on the form and size and position it where I want the ActiveX control to go. This object becomes a placeholder for the ActiveX control, making it easier to size, position, and place other objects in relation to it. To tell SFActiveX that this object is the placeholder, enter its name in the cObjectName property. SFActiveX will remove this object and put the ActiveX control in its place.

SFActiveX doesn't have a lot of code. Its Init method calls the custom method LoadActiveX (which does all the work) unless told not to do so by passing .T. as a parameter (this is provided so you can instantiate SFActiveX in code, set the properties as appropriate, and then call LoadActiveX manually to load the control). I won't address all the code in the LoadActiveX method here (you can check it out yourself), just the more interesting stuff.

SFActiveX relies heavily on being able to look stuff up in the Windows Registry, because that's where ActiveX controls installed on a user's machine are registered. In order to work with the Registry, LoadActiveX uses the services of another class, SFRegistry, defined in SFREGISTRY.VCX. We won't look at this class here; suffice it to say that I stole most of the code from REGISTRY.PRG, a Registry-handling class that comes with VFP, but made the programmatic interface a bit simpler.

In order to use SFRegistry, you have to instantiate it; that work is done by a call to NEWOBJECT. NEWOBJECT.PRG is an updated version of NEWOBJ.PRG that I presented in this column in the September 1997 issue. It was updated to support the same syntax as the Tahoe NEWOBJECT function. This means that in Tahoe applications, you can omit this program, and any code calling it will work just fine.

The first job for LoadActiveX is to figure out whether the desired ActiveX control is installed on the user's system and, if the class ID was specified rather than the OLE class, which OLE class to use. The following code does this. Note that if you specify the OLE class and it can't be found, LoadActiveX adds a ".1" suffix to the class and tries that before giving up. (The method starts with WITH THIS, so all unspecified object references are the class itself.)

```
loRegistry = newobject('SFRegistry', ;
  'SFRegistry.vcx', cnHKEY_CLASSES_ROOT)
if empty(.cOLEClass)
  lcCLSID  = iif(left(.cClassID, 1) = '{', ;
    .cClassID, '{' + .cClassID + '}')
  lcProgID = loRegistry.GetKey('CLSID\' + lcCLSID + ;
    '\ProgID')
  llOK = not empty(lcProgID)
else
  lcProgID = .cOLEClass
  lcCLSID  = loRegistry.GetKey(lcProgID + '\CLSID')
  if empty(lcCLSID)
    lcProgID = .cOLEClass + '.1'
    lcCLSID  = loRegistry.GetKey(lcProgID + '\CLSID')
  endif empty(lcCLSID)
  llOK = not empty(lcCLSID)
endif empty(.cOLEClass)
if llOK
  lcOCX = loRegistry.GetKey('CLSID\' + lcCLSID + ;
    '\InProcServer32')
  llOK  = not empty(lcOCX) and file(lcOCX)
endif llOK
```

At the end of this code, lcProgID contains the OLE class (ProgID), lcCLSID contains the class ID (which we don't care about after this point), and lcOCX contains the name and path of the OCX file on the user's system (found in the InProcServer32 key in the Registry). llOK is only .T. if we found what we were looking for in the Registry and the OCX file exists. If not, the user will get an error message, and this method will return .F.

If everything is okay, LoadActiveX checks whether a placeholder was specified and, if so, saves its size and position and then removes it.

```
llObject = not empty(.cObjectName)
if llObject
  lcObject = .cObjectName
  with .Parent.&lcObject
    lnTop    = .Top
    lnLeft   = .Left
    lnWidth  = .Width
    lnHeight = .Height
  endwith
  .Parent.RemoveObject(lcObject)
endif llObject
```

If a class and class library were specified, LoadActiveX opens the library (using either SET PROCEDURE or SET CLASSLIB, depending on the library's extension). It then adds the ActiveX control or ActiveX subclass to the container, giving it the name specified in the cNewObjectName property. If a placeholder object was specified, the control is sized and positioned to the saved values. Notice the ProgID (stored in lcProgID) must be specified in the call to AddObject; if this was left out, the user would get a dialog box asking which control to insert. This is why we have to look in the Registry to determine the ProgID for the ActiveX control.

```
lcObject = .cNewObjectName
.Parent.AddObject(lcObject, .cClass, lcProgID)
with .Parent.&lcObject
  if llObject
    .Top    = lnTop
    .Left   = lnLeft
    .Width  = lnWidth
    .Height = lnHeight
  endif llObject
```

```
  .Visible = .T.
endwith
```

Using SFActiveX

It's easy to use SFActiveX: Simply drop it on a form (or other container, such as a page in a PageFrame where the ActiveX control should go), fill in some properties, and put code in the Init method of the form to set any properties of the instantiated ActiveX control as desired. To make it even easier to use, I've created subclasses of SFActiveX for the ActiveX controls I use most frequently: SFCommonDialog, SFImageList, and SFTreeView. These subclasses just have the appropriate class ID entered into the cClassID property so you don't have to look it up. Drop one of these on a form to create the desired ActiveX control.

Ah, but there's one complication: What if you need to put some code in a method of the ActiveX control? This is frequently the case with visual controls -- for example, with the TreeView and ListView controls, you need to take some action when the user clicks on a node or item in the list. In the case of the TreeView control, you'd put code into the NodeClick method. The problem, of course, is that you can't add code to an object. The only way to accomplish this is to create a subclass of the ActiveX control, put the desired code into the subclass, and tell SFActiveX (via the cClass and cLibrary properties) to instantiate that class. One slight "gotcha": You can't create the subclass in a VCX because, like dropping an ActiveX control directly on a form, VFP puts binary information about the control into the VCX, and that's the whole cause of the problem SFActiveX was designed to solve. Instead, you need to create the subclass in a PRG. Here's the code from the sample ACLASSES.PRG that defines a subclass of the TreeView control; it has code in the NodeClick method (albeit simple code) that fires when the user clicks on a node. Note that OLEClass must be specified, or the user will get a dialog box asking which control to insert.

```
define class MyTreeView as OLEControl
  OLEClass = 'COMCTL.TreeCtrl'
  Name     = 'MyTreeView'
  procedure NodeClick
    lparameters toNode
    wait window 'You clicked on node ' + toNode.Text
  endproc
enddefine
```

Believe it or not, with all the problems we've solved using this scheme, we're not quite out of the woods yet! There's one last "gotcha": Although you might think you could set some properties of the control in the subclass definition (such as Style and LineStyle in the case of the TreeView control), don't do it. For a reason that escapes me, setting properties in the subclass seems to cause those properties to be carved in stone. Trying to change them later in a form the control is on fails completely. This means you have to set the properties after the control has been instantiated, usually in the Init of the form (which you can do, since by the time the form's Init fires, the ActiveX controls have been instantiated). I'll show an example of this in a moment.

One final heads-up about the subclass: If you create an Init method in the subclass, it'll need to accept a parameter. The OLE class is passed to the Init method (notice the third parameter in the AddObject call in LoadActiveX); although you don't need it, you'll get an error if you don't have an LPARAMETERS statement.

The sample TREEVIEW.SCX form shows how SFActiveX is used. This form contains three objects: an SFTreeView object (SFActiveX subclass specific for TreeViews) called oTreeViewLoader, an SFImageList object (SFActiveX subclass specific for ImageLists) called oImageListLoader, and a Shape called shpTreeView that acts as the placeholder for the TreeView control. The cNewObjectName properties for oTreeViewLoader and oImageListLoader specify the names of the controls to create (oTree and oImageList, respectively). oTreeViewLoader also has the name of the placeholder object (shpTreeView) in its cObjectName property and the name of the TreeView subclass we want to use (MyTreeView) and its location (ACLASSES.PRG) in the cClass and cLibrary properties. The Init method of the form sets some properties for the ActiveX controls, loading images in the case of the ImageList and nodes in the case of the TreeView.

```
local loPicture
* Load the ImageList images.
with This.oImageList
  loPicture = loadpicture('AUDIO.ICO')
  .ListImages.Add(, 'Audio', loPicture)
  loPicture = loadpicture('DESKTOP.ICO')
  .ListImages.Add(, 'Desktop', loPicture)
endwith
* Set some TreeView properties.
with This.oTree
  .ImageList     = This.oImageList.Object
  .Style         = 7
  .LineStyle     = 1
  .LabelEdit     = 1
  .HideSelection = .F.
  .Indentation   = 25
* Load the TreeView with sample nodes.
  .Nodes.Add(, 1, 'Top1', 'First Top Node',  'Audio')
  .Nodes.Add(, 1, 'Top2', 'Second Top Node', 'Audio')
  .Nodes.Add('Top1', 4, 'Child1', 'First Child Node', ;
    'Desktop')
  .Nodes.Add('Top1', 4, 'Child2', 'Second Child Node', ;
```

```
    'Desktop')
  .Nodes.Add('Top2', 4, 'Child3', 'Third Child Node', ;
    'Desktop')
  .Nodes.Add('Top2', 4, 'Child4', 'Fourth Child Node', ;
    'Desktop')
endwith
```

Summary
ActiveX controls are both wonderful and terrible. They're wonderful because they can give your applications the professional, polished look users expect from modern 32-bit applications (the TreeView control that comes with VFP and the ctListBar control from dbi technologies inc. are good examples), or they can provide advanced capabilities that would either be impossible or very time-consuming to create in VFP code (the DynamiCube control I discussed in my March 1998 column is an example of that). They're terrible because when "OLE class not registered" or other OLE errors occur, you can't just roll up your sleeves and use the VFP Debugger to track down the problems. SFActiveX has been a life saver for me; it handles the main problem I have with ActiveX controls. I'm sure you'll find it as useful as I have.