

# Appendix D. Using ActiveX Controls on the Web

## ***D.1. Introduction***

The most intriguing feature of ActiveX components is that they can be incorporated into a web page, and used as if the web-page itself was a program. (This only works with the Internet Explorer browser however.) Before you can begin using ActiveX components on the web, there are a number of things you must do. This appendix goes through the steps that you must complete to use your ActiveX components on your web site.

## ***D.2. Marking Controls as Safe***

First, you must mark your components as safe. There are two kinds of safe, “safe for scripting” and “safe for initialization.” The first means that its OK for scripts written in either VBScript or JScript to assign values to the properties of your component, and to call its methods. What you’re saying when you mark a component as safe is that there is no way that a pirate web-site could use your component to damage a user’s system. (In other words, if you have a method that reformats the user’s hard-drive, *do not* mark the component as safe.) Safe for Initialization means that its OK for the browser to assign the initial values to your persistent properties. You will normally use both markings or neither. In both cases, think about what could happen if a malicious person included your component in his web page. If your component could be used to damage the user’s system, it is *not safe*.

If you plan to use your components on your web site, you definitely want to make your components safe and mark them as safe. (Putting an unsafe component on the web can cost you a lot of money in law suits.) To mark a component as safe, you must replace the default *CMineCtrl::CSGEditorCtrlFactory::UpdateRegistry(BOOL)* function of your support class. (Replace “Mine” with the name of your component.) You will find this function in the *MineCtl.cpp* file. (If you are using ATL instead of MFC, there is an entirely different procedure that must be followed. Skip ahead to “Safe Marking for ATL Components.”)

Before you do this, you will need to add the helper files to your project. These helper files are included in every project on the CD ROM, you can obtain them from there, or from the Microsoft Developer Network. The files are named “helpers.h” and “helpers.cpp”. Copy these files into your project directory, and make them part of your project. Now add the following two lines to your *MineCtl.cpp* file. Make sure these lines *follow* the other `#include` lines.

```
#include "helpers.h"  
#include <objsafe.h>
```

Finally, replace the body of your *UpdateRegistry* function with the code of Figure D.1. BE CAREFUL, because there are three lines of the existing function body that you need to copy into this new function body. These three lines refer to `#define` statements and variables that contain the name of your component. They are conspicuously marked in Figure D.1.

```

if (bRegister)
{
    BOOL retval = AfxOleRegisterControlClass(
        AfxGetInstanceHandle(),
        m_clsid,
        m_lpszProgID,
        IDS_SGEDITOR,           // COPY FROM OLD FUNCTION BODY
        IDB_SGEDITOR,          // COPY FROM OLD FUNCTION BODY
        afxRegApartmentThreading,
        _dwSGEditorOleMisc,    // COPY FROM OLD FUNCTION BODY
        _tlid,
        _wVerMajor,
        _wVerMinor);
    // mark as safe for scripting--failure OK
    HRESULT hr = CreateComponentCategory(CATID_SafeForScripting,
        L"Controls that are safely scriptable");
    if (SUCCEEDED(hr))
    {
        RegisterCLSIDInCategory(m_clsid, CATID_SafeForScripting);
    }
    hr = CreateComponentCategory(CATID_SafeForInitializing,
        L"Controls safely initializable from persistent data");
    if (SUCCEEDED(hr))
    {
        RegisterCLSIDInCategory(m_clsid, CATID_SafeForInitializing);
    }
    return retval;
}
else
{
    return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
}

```

**Figure D.1 Safe-Marking Code.**

If you neglect to mark your component as safe, anyone accessing your web page will receive a warning about unsafe controls every time your web page is accessed. This will make your page extremely unpopular.

### **D.3. Safe Marking for ATL Components**

The procedure for safe marking of ATL components is much simpler than that for MFC components. To mark an ATL component as safe, first find the line “END\_PROP\_MAP( )” in your MineCtl.h file, where MineCtl is the name you used

when you added the control to your ATL project. Add the four lines of following this line.

```
BEGIN_CATEGORY_MAP(CAPentCtl)
    IMPLEMENTED_CATEGORY(CATID_SafeForScripting)
    IMPLEMENTED_CATEGORY(CATID_SafeForInitializing)
END_CATEGORY_MAP()
```

**Figure D.2. ATL Safe Marking Code.**

#### ***D.4. Supporting Property Bags in ATL***

By default, ATL controls do not implement enough of the ActiveX specification to be usable in a web page. Fortunately, the only deficiency is in the support for persistent properties. (If you don't have any persistent properties, you can ignore this section.) In web pages, the initial values for persistent properties are stored in something called a *Property Bag*. (I don't make up these names!) You need to add property bag support to your ATL component before the persistent properties will be initialized properly. Before you do this, make sure you have downloaded and installed the latest updates for your version of Visual C++, because there is a flagrant bug in ATL property bag support in some of the released versions of Visual C++.

The first step is to add the following to the list of derived classes for your class **CMineCtl**, where *Mine* is the name of your control.

```
public IPersistPropertyBagImpl<CMineCtl>
```

Next, find the COM map for your control. (It should be the next thing in your .h file.)

The COM map starts with the macro `COM_MAP(...)` and ends with the macro `END_COM_MAP( )`. Add the following line to the end of your COM map.

```
COM_INTERFACE_ENTRY(IPersistPropertyBag)
```

Adding these two lines to your project will make persistent properties work on a web page.

### ***D.5. Creating an INF file***

A big part of putting your component on a web page is preparing your component for distribution. Before your ActiveX component can run on someone else's system, it must be downloaded and installed. The INF file contains instructions for installing your component on a new user's system. The INF file will then be packaged with your object file or files, and be downloaded as a unit. The INF file for an ATL component is much simpler than that for an MFC component. The problem is that MFC components require three additional files in addition to your object file. If you are using MFC, your object file will be named `Mine.ocx`, where "Mine" is the name of your project. (More on this in the next section.) If you are using ATL, your object file will be named `Mine.dll`.

The INF file for an MFC component (the Simple Graphical Editor) is given in Figure D.3. The file of Figure D.3 shows the correct way to include the three additional files with your component. I have seen other recommendations about how to do this, but as far as I can tell, none of them work. This version has been tested, and does indeed work correctly.

```
; Install SGEitor.ocx

[Version]
Signature="$CHICAGO$"
AdvancedINF=2.0

[Add.Code]
SGEitor.ocx=SGEitor.ocx
mfc42.dll=mfc42.dll
msvcrt.dll=msvcrt.dll
olepro32.dll=olepro32.dll

[SGEitor.ocx]
file-win32-x86=thiscab
clsid={DCAFB66A-1D94-4D8B-8D4F-1DDF2E5AF7C4}
DestDir=11
FileVersion=1,0,0,1
RegisterServer=yes

; dependent DLLs
[msvcrt.dll]
; This is an example of conditional hook. The hook only gets processed
; if msvcrt.dll of the specified version is absent on client machine.
FileVersion=6,0,8168,0
hook=mfc42installer

[mfc42.dll]
FileVersion=6,0,8168,0
hook=mfc42installer

[olepro32.dll]
FileVersion=5,0,4261,0
hook=mfc42installer

[mfc42installer]
file-win32-x86=http://activex.microsoft.com/controls/vc/mfc42.cab
; If dependent DLLs are packaged directly into the above cabinet file
; along with an .inf file, specify that .inf file to run as follows:
;InfFile=mfc42.inf
; The mfc42.cab file actually contains a self extracting executable.
; In this case we specify a run= command.
run=%EXTRACT_DIR%\mfc42.exe
```

**Figure D.3. An MFC INF File.**

I would recommend using this file and modifying it to install your component. Every line starting with a semicolon is a comment, so you can safely ignore those lines. Make the following changes.

1. Modify the initial comment to document your file. Describe your component as accurately as possible.
2. Change the line that reads “SGEditor.ocx=SGEditor.ocx”. Put your own ocx name in both places.
3. Change the line that reads “[SGEditor.ocx]”. Put your own ocx name here.
4. In the section under [SGEditor.ocx] change the line that reads:  
`clsid={DCAFB66A-1D94-4D8B-8D4F-1DDF2E5AF7C4}`. You need to replace the stuff between the braces with the GUID of your own component. To find out the GUID of your component, open your project and then open the source file that has the suffix “.odl”. You will find this in your list of source files. Scroll down to the end of this file. You will find several strings resembling the above in this file. The one you want is the *last* one in the file. Copy the string from your ODL file into your INF file.
5. In the section under [SGEditor.ocx] find the line that reads  
`FileVersion=1,0,0,1`. Change this to match the version number of your ocx file. **DO NOT GUESS AT THE VERSION NUMBER!** There are two ways to find out the version number. The first is to open your project and look at the Version resource. This is the best way. The second way is to right-click on your .ocx file and call up the properties window. Record the version number correctly.
6. Once you have made these modifications save the file under the name *Mine.inf*, where *Mine* is the name of your ocx file.

Figure D.4 gives the format of an ATL INF file. You modify it in the same way you modify the MFC INF file. The only difference is locating the GUID. For ATL projects,

this can be found in the .idl source file. Suppose the support class for your ATL component is named CMine. Find the line in this file that reads “coclass Mine”. The GUID you want is the one *just before* this line. Don’t forget to modify the version number, and make sure you get it correct.

```
; Apent.ocx
; Advanced Pentominos

[Add.Code]
Apent.ocx=Apent.ocx

[Apent.ocx]
file-win32-x86=thiscab
clsid={6764E70D-6016-11D4-A057-00104B5F5B5B}
DestDir=11
FileVersion=1,0,0,1
RegisterServer=yes
; end of INF file
```

**Figure D.4. An ATL INF File.**

## ***D.6. Creating a CAB file***

Before creating your CAB file, you must have two things: your INF file, and the *release version* of your OCX (or DLL) file. You cannot distribute the debug version of an MFC control. It won’t work on a user’s system unless the user just happens to have C++ installed. Even then it still might not work right. So first, create the release version of your OCX. Go to the Build menu, and select the “Set Active Configuration” command. This command will bring up a window that will allow you to select a configuration. The default for both MFC and ATL is Debug. For MFC change to Release. For ATL change to Release MinDependency. Rebuild the project (it will most likely recompile from scratch). You should now have a Release directory (ReleaseMinDependency for ATL.) *You must have an ocx file in this directory to proceed!*

Create a new sub-directory in your project directory, and place both the INF file and the release-version OCX file in it. Create a new txt file in this directory and add the following single line to it.

```
cabarc -s 6144 n sgeditor.cab SGEdition.ocx SGEdition.inf
```

Change the string “SGEditor” to match the name of your OCX. I recommend making the name of the .CAB file all lower case. If your HTTP server is a UNIX server, case will make a difference, and all lower case causes fewer problems. Save the txt file, and change the suffix from txt to bat. Double-click on the file to create your CAB file. In most cases this will fail the first time you try it because Windows won’t be able to find the cabarc program. Search your system for cabarc.exe. If you find it, copy it (not move!) into your Windows directory. If you can’t find it go to the Microsoft Website and search for cabarc.exe. You may need to search around a bit, but it is part of several different SDKs, and eventually you will find one containing it. (If I give you the URL, it will be out of date by the time you read this.)

### ***D.7. Signing***

Once you have CAB files, you are almost ready to begin distributing your ActiveX components. The last thing you need is to sign your ActiveX component. Unfortunately, there is no cheap way to do this. Before you can sign ActiveX components, you must have something called a “Cert.” You can get these from several different places, Verisign is one of them. Unfortunately, they are rather expensive. If you are a University student,

your University may be able to provide you with one free of charge, but as of this writing only a few Universities provide this service.

If you manage to obtain a cert, this is how you use it. Your cert will actually consist of two files, a file with the suffix `spc` containing the cert itself, and a file with the suffix `pvk` containing your private key. (The C++ SDK has programs that allow you to create these two files, *but they are fakes!* You can practice signing files with these fake files, but the user's web browser won't be able to recognize the signature.) If you have a real cert from a recognized certificate issuing authority, the two files are used by the `SignCode` function to insert a digital signature into your CAB file. You should also sign your OCX file before creating your CAB file. First create an empty file named `signocx.txt`, and add the following line to it.

```
signcode -spc mycert.spc -v mykey.pvk -n "My OCX" -i "http://..." mine.ocx
```

Next create a file named `signcab.txt` and add the following line to it.

```
signcode -spc mycert.spc -v mykey.pvk -n "My OCX" -i "http://..." mine.cab
```

You may need to specify a full path name for the files `mycert.spc` and `mykey.pvk`. You may also need to specify a full path name for the `signcode` exe file. Replace the “`http://...`” string with the web address providing documentation for your component. Replace the “`My OCX`” string with an English description of your component. After creating these two txt files, change the suffix of both to `bat`. You may now double-click on the files to sign your code.

You can distribute your components without signing them, but if you do this, your users will have to change the default security settings of their browsers to download them. You should warn your users about this. If at all possible, sign your CAB files.

### **D.8. Versioning**

In the section on INF files, we warned you several times to “get the version number right.” Once you have placed your component on your web site, the version number becomes *very important*. (Version numbers are important any time your component is being used by someone other than yourself.) Once the component is on your web site, *you cannot change it without changing the version number*. This is not just a teacher’s warning about something nice that you *ought* to do. It is an absolute and inescapable requirement for your web page to function correctly. The software at the user’s site will key on the version number to insure that things are functioning correctly, and if you mess up the version numbers, you mess up your users.

Before distributing your component, edit the version resource, and *fill in all the fields*. The version number appears in two places. *Make sure they are the same*. You *should* change the fourth part of the version number every time you build it for testing. You *must* change the version number every time you distribute it. *Make sure the new version number is **larger** than the previous*.

ActiveX components are downloaded and installed once. After the first access, all accesses to the component are made to the downloaded component, not the component on your web site. The component will be downloaded and installed again when the version number changes. The version number change is the *only way* to get your changes to your users.

## D.9. HTML Tags and HTML Editors

To include an ActiveX component in a web page, place the following tags in your HTML file. (These tags are for the SGEEditor component.)

```
<object classid="clsid:DCAFB66A-1D94-4D8B-8D4F-1DDF2E5AF7C4"  
id="SGEditor1" width="100" height="50"  
codebase="http://www.mydn.com/mine.cab#version=1,0,0,1">  
  <param name="_Version" value="65536">  
  <param name="_ExtentX" value="2646">  
  <param name="_ExtentY" value="1323">  
  <param name="_StockProps" value="0">  
  <param name="DrawMode" value="0">  
  <param name="FillColor" value="0">  
  <param name="LineColor" value="0">  
  <param name="HandleRadius" value="4">  
</object>
```

**Figure D.5. HTML Tags for ActiveX Components.**

The classid parameter of the object tag must contain the GUID of your component. This must be the same GUID specified in your INF file. The codebase parameter of the object tag contains the URL of the CAB file containing your component, and the current version of your component. *The version number must be correct.* There are four places the version number appears. It appears twice inside your ActiveX component, once in your INF file, and once in your codebase parameter. *All four numbers must match.*

If you have several instances of the same component on a single web page, the codebase parameter is required only on the first, but to be safe, you should put it on all of them.

The param tags are used to supply the initialization values for your persistent properties. You must have one param tag for each persistent property. The first four lines in Figure D.5 are properties provided by the browser, not properties defined by the

component. The easiest way to get all of this right is to use an ActiveX aware HTML editor. (One is supplied with Visual Studio.)

### ***D.10. Scripts***

Scripting is beyond the scope of this appendix, however with a few tips you can probably accomplish everything you want to accomplish in VBScript. First, create a Visual Basic program that does the same thing you want your web page do. Take the code of the Visual Basic program, and embed it in a VBScript in your web page. To do this, place the following two lines between the `</head>` and `<body>` tags in your HTML file.

```
<script language="VBScript"><!--  
--></script>
```

Put your visual basic code between these two lines. You will need to make a few changes. First, remove the `Private` keyword from all function definitions. Local variable definitions are not used, so remove them. (All variables are untyped variants.) Global variables and function parameters may not have a type so remove the “`AS INTEGER`” (or other type) from all declarations. Save the file and then run the script. Remove whatever else the browser complains about. Almost anything that will work in Visual Basic will also work in VBScript.

### ***D.11. Licensing***

The full details of licensing are beyond the scope of this appendix, nevertheless we want to at least discuss the basic issues. When using an ActiveX component in Visual

Basic, many commercial components will refuse to run in design mode unless you have paid the licensing fee. The MSDN library that comes with Visual Studio will explain how to incorporate such a feature into your own components. What licensing does is allow programmers to distribute commercial components with their applications. Users of the application cannot use the distributed components to create their own applications without paying the license fee.

This scheme does not work for components distributed with web pages, because there is no design mode. HTML files can be created with virtually any text editor, including the Windows Notepad. To protect your component from unauthorized use, you must also include some sort of run-time license. The way this normally works, is to provide license-checking features that will permit a component to execute in a web page only if the web page, the CAB file and a special licensing file have been downloaded from a common server. The license file is a special file that is tied to a particular system and cannot be easily faked. For more information, see the Microsoft Developer Network (MSDN) index under "LPK files." If you purchase third party components for use on your web pages, you will need to learn how to create and use LPK files. The MSDN documentation provides detailed information on creating and using LPK files.

## ***D.12. Conclusion***

The best way to learn about using ActiveX components in web pages is to practice. For some examples of web pages containing ActiveX components, see the URL <http://www.csee.usf.edu/~maurer/vdal>.