Latest ActiveX Technology Elements

Cezar BOTEZATU, Ph.D., Cornelia BOTEZATU, Ph.D.
Romanian-American University, Bucharest, Romania

The paper is meant to be an introduction into the ActiveX technology and some of its most popular applications. The authors have tried to briefly discuss about ActiveX controls, their characteristics, advantages and purpose, about Data Objects and about Microsoft Office Visio 2003 ActiveX Control. The paper also contains an extensive set of examples that make it accessible even to the uninitiated reader.
Keywords: *ActiveX technology, ActiveX control, The <OBJECT> Tag, ADO (ActiveX Data Objects), Microsoft Office Visio 2003 ActiveX Control.*

ActiveX is a set of technologies from Microsoft that enables interactive content for the World Wide Web. Before ActiveX, Web content was static, 2-dimensional text and graphics. With ActiveX, Web sites come alive using multimedia effects, interactive objects, and sophisticated applications that create a user experience comparable to that of high-quality CD-ROM titles. ActiveX provides the glue that ties together a wide assortment of technology building blocks to enable these "active" Web sites.

An ActiveX control is really just another term for "OLE Object" or, more specifically, "Component Object Model (COM) Object." In other words, a control, at the very least, is some COM object that supports the **IUnknown** interface and is also self-registering. Through **QueryInterface** a container can manage the lifetime of the control, as well as dynamically discover the full extent of a control's functionality based on the available interfaces. This allows a control to implement as little functionality as it needs to, instead of supporting a large number of interfaces that actually don't do anything. In short, this minimal requirement for nothing more than **IUnknown** allows any control to be as lightweight as it can.

Other than **IUnknown** and self-registration, there are no other requirements for a control. There are, however, conventions that should be followed about what the support of an interface means in terms of functionality provided to the container by the control. This section describes what it means for a control to actually support an interface, as well as methods, properties, and events that a control should provide as a baseline if it has occasion to support methods, properties, and events.

Self-Registration

ActiveX Controls must support self-registration by implementing the DllRegisterServer and DllUnregisterServer functions. ActiveX Controls must register all the standard registry entries for embeddable objects and automation servers.

ActiveX Controls must use the Component Categories application programming interface (API) to register themselves as a control and register the component categories that they require a host to support, and any categories that the control implements. In addition, an ActiveX control might want to register the "control" keyword in order to allow older containers, such as Microsoft Visual Basic 4, to host them.

ActiveX Controls should also register the ToolBoxBitmap32 registry key, although this is not mandatory.

The Insertable component category should only be registered if the control is suitable for use as a compound document object. It is important to note that a compound document object must support certain interfaces beyond the minimum **IUnknown** required for an ActiveX control. Although an ActiveX control might qualify as a compound document object, the control's documentation should clearly state what behavior to expect under these circumstances.

The VBScript programming language is only

half of Microsoft's current Internet programming strategy. The other half is ActiveX, a way to develop programmable objects that can be added to Web pages alongside images, text, Java applets, and other media.

VBScript provides access to the intrinsic HTML controls-buttons, text fields, radio buttons, and other things that are common to Web-based forms. These controls are encountered any time someone registers to join a Web site or order a product through the Web.

To create sophisticated programs to run on a Web page, somebody might want to extend the possibilities beyond intrinsic controls by using ActiveX controls.

An ActiveX control is developed using a language such as Visual C++, Visual Basic, or Delphi. Like an OCX, an ActiveX component is designed to be used by some other software - a Web browser, in this case.

More than 3,000 ActiveX controls are available for use, according to Microsoft. In addition to being usable on Web pages, these controls can be used with other types of software developed with programming languages such as Java, Borland C++ and Delphi, Visual Basic, and Visual C++.

These controls are developed in other languages, but their operation can be modified and customized with the use of HTML code and VBScript programs. ActiveX controls are placed on a Web page using a special extended HTML tag called <OBJECT> and a supporting tag called <PARAM>.

The <OBJECT> Tag

The <OBJECT> tag, proposed by the World Wide Web Consortium as a standard, is used to place an object on a Web page. The primary type of object discussed in this chapter is an ActiveX control, but the tag considers an object to be any type of media that can be put on a page. <OBJECT> was proposed by the Consortium as a way to replace several current HTML tags and attributes-the <IMG> tag, the Java <APPLET> tag, the DYNSRC attribute used for audio and video by Microsoft, and other proprietary extensions to HTML. <OBJECT> also is flexible enough in design to handle new forms of media not yet invented for the Web.

Attributes are used with the <OBJECT> tag to specify the following information: the object's name; the type of object; the URL address where the object can be found; layout information such as height, width, spacing, border width, alignment, and so on; an ID code to verify the object's identity.

If the object has parameters, they can be set with the <PARAM> tag. This tag has two attributes: NAME and VALUE. The NAME attribute gives the parameter a name, and VALUE sets up a value for that parameter.

Although <OBJECT> is intended to be used with a broad range of media, one example of it in current use is the ActiveX control.

Generating <OBJECT> HTML Code

When the ActiveX Control Pad is used to add an ActiveX object to a Web page, an <OBJECT> tag is added automatically to the page. Here's an example of an ActiveX control's <OBJECT> tag:

```
<OBJECT ID="SpinButton1" WIDTH=16 HEIGHT=32 CLASSID="CLSID: 79176FB0-B7f2-11CE-97EF-00AA006D2776">
 <PARAM NAME="Size" VALUE= "423; 846">
</OBJECT>
```

This <OBJECT> tag creates an ActiveX spin button control with up and down arrows to change a value. Java programmers will recognize the <PARAM> tag, because it has the same attributes (NAME and VALUE) and the same usage as it does with the Java <APPLET> tag.

Because this HTML code is added automatically by the ActiveX Control Pad, you do not need to enter it yourself into a Web page's HTML file.

The ID attribute of the <OBJECT> tag gives the object a name. One of the biggest advantages of VBScript and ActiveX is the capability of one object to communicate with another object. A VBScript program can be used for one element on a page-for example, a <FORM> button-to modify another program, such as an ActiveX control. ID is needed for one object to know how to contact another.

The CLSID attribute identifies the type of

object and provides some identifying characteristics of the object. In the preceding example, the CLSID was set to a complicated string of numbers and letters:
CLSID:79176FB0-B7f2-11CE-97EF-00AA006D2776
This has two parts. The section before the colon, CLSID, identifies this object as an ActiveX control. Another example of an identifier would be java:, representing an applet programmed in that language.
The section after the colon indicates some registration information that reveals where the ActiveX control can be found on the user's Windows system. ActiveX controls are downloaded to the user's system and run locally. The CLSID gives the browser enough information to find, identify, and run the control. It also creates a unique identifier for the ActiveX control. No matter how many ActiveX controls are implemented across the Internet, each will use part of the CLSID to establish its identity.
In addition to being usable on the World Wide Web, ActiveX controls have an advantage over other Internet programming solutions such as Java applets and Netscape plug-ins.
These controls can be used immediately in other applications. For example, a control that performs an image editing task on the Web can be plugged into a software program as easily as it was placed on a page.
An ActiveX-enabled Web browser will behave differently if it encounters a new control than if it has seen the control previously.
If you are using a browser that can handle ActiveX controls and you come to a page containing a control, a check will be made to determine whether you have downloaded the control previously. This check will use the CLSID attribute of the <OBJECT> tag to determine whether the ActiveX control is present on your system.
Because ActiveX controls are executed on the user's system, there is obvious potential for a programmer to run malicious code. In order to run an ActiveX control, you need some means of identifying the author as a trustworthy source.
The VeriSign company is handling ActiveX developer certification for a large number of the existing controls. The certificate window that opens when you encounter a new control on a Web page has a link to a control verification source such as VeriSign and probably a link to the developer's Web site.
After the control has been downloaded and executed, it remains on the user's computer so that it does not have to be reloaded each time the control is found on a Web page. The only time that an ActiveX control will be downloaded more than once is when a new version is offered that the user does not yet have.
This enables much quicker access to an ActiveX control than is possible with Java applets, which download again each time they are encountered. However, the disadvantage is that ActiveX controls take up space on a user's hard drive.
To see a sampling of the ActiveX controls that have been made available, the Internet information service CNET has introduced an ActiveX file directory and news site.
When a control is on your system, you can use it as a component in your own Web pages and software projects. CNET's ActiveX site has many controls available that cater to programmers in need of useful components.
Now, there are four ways to write an ActiveX control: Microsoft Foundation Classes (MFC); ActiveX Template Library; BaseCtrl framework; Visual J++™ (COM objects only).
Short for ***A**ctiveX **D**ata **O**bjects,* Microsoft's newest high-level interface for data objects, ADO is designed to eventually replace *Data Access Objects (DAO)* and *Remote Data Objects (RDO)*. Unlike RDO and DAO, which are designed only for accessing relational databases, ADO is more general and can be used to access all sorts of different types of data, including web pages, spreadsheets, and other types of documents.
Together with OLE DB and ODBC, ADO is one of the main components of Microsoft's *Universal Data Access (UDA)* specification, which is designed to provide a consistent

way of accessing data regardless of how the data are structured.

ADO provides developers with a powerful, logical object model for programmatically accessing, editing, and updating data from a wide variety of data sources through OLE DB system interfaces. The most common usage of ADO is to query a table or tables in a relational database, retrieve and display the results in an application, and perhaps allow users to make and save changes to the data. Other tasks include: querying a database using SQL and displaying the results; accessing information in a file store over the Internet; manipulating messages and folders in an e-mail system; saving data from a database into an XML file; executing commands described with XML and retrieving an XML stream; saving data into a binary or XML stream; allowing a user to review and make changes to data in database tables; creating and reusing parameterized database commands; executing stored procedures; dynamically creating a flexible structure, called a **Recordset**, to hold, navigate, and manipulate data; performing transactional database operations; filtering and sorting local copies of database information based on run-time criteria; creating and manipulating hierarchical results from databases; binding database fields to data-aware components; creating remote, disconnected **Recordsets**.

ADO exposes a wide variety of options and settings in order to provide such flexibility. Therefore it's important to take a methodical approach to learning how to use ADO in an application, breaking down each of the goals into manageable pieces.

Most developers are very familiar with ADO's ancestors, DAO and RDO. A few items of note:

Cancel applies to asynchronous **Execute** and **Open**; an error will occur if called otherwise.

Recordset.Save writes the recordset to a physical file. No more writing out the data as text!

MarshalOptions controls whether a client-side recordset sends back all records or only the modified records.

PageSize, PageCount, AbsolutePage are ideally for Web sites. They allow you to request the 123rd page of records where there are 63 records on a page—no more counting through records.

Recordset.StayInSync is part of the data shaping available in ADODB. Version 2.1 adds grandchild aggregates, reshaping and parameterized commands using COMPUTE to the earlier data shaping, and hierarchical recordsets of version 2.0.

Using ADODB on the Web

ADO may be used in many environments, one of the most popular of which is Internet Information Services (IIS) Active Server Pages (ASP).

The ADO 2.8 SDK consists of the following components:

Microsoft ActiveX Data Objects (ADO) enable client applications to access and manipulate data from a variety of sources through an OLE DB provider. Its primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. ADO supports key features for building client/server and Web-based applications.

Microsoft ActiveX Data Objects (Multi-dimensional) (ADO MD) provides easy access to multidimensional data from languages such as Microsoft Visual Basic , Microsoft Visual C++ , and Microsoft Visual J++ . ADO MD extends Microsoft ActiveX Data Objects (ADO) to include objects specific to multidimensional data, such as the CubeDef and Cellset objects. With ADO MD you can browse multidimensional schema, query a cube, and retrieve the results.

Like ADO, ADO MD uses an underlying OLE DB provider to gain access to data. To work with ADO MD, the provider must be a multidimensional data provider (MDP) as defined by the OLE DB for OLAP specification. MDPs present data in multidimensional views as opposed to tabular data providers (TDPs) that present data in tabular views.

Remote Data Service (RDS) is a feature of ADO, with which you can move data from a server to a client application or Web page, manipulate the data on the client, and return updates to the server in a single round trip.

Microsoft ActiveX Data Objects Exten-

sions for Data Definition Language and Security (ADOX) is an extension to the ADO objects and programming model. ADOX includes objects for scheme creation and modification, as well as security. Because it is an object-based approach to manipulation, you can write code that will work against various data sources regardless of differences in their native syntaxes.

ADOX is a companion library to the core ADO objects. It exposes additional objects for creating, modifying, and deleting schema objects, such as tables and procedures. It also includes security objects to maintain users and groups and to grant and revoke permissions on objects.

If you are a Java or Visual Basic developer, you may have run into situations where you really wanted to use some key feature of the operating system. Let's say you want to be able to draw something using DirectX™. You can't do it directly with Visual Basic, and, if you try it in Java, you aren't going to be playing in the sandbox anymore (for those of you who aren't Java-aware, the "sandbox" is the name used for the virtual machine in which your Java applet runs). This virtual machine is not allowed access to your system's services, to protect you, the consumer of the Java applet, from inadvertently downloading a virus that reads from or writes to your hard disk. A way to get around the problem of not being able to access basic system services is to write an ActiveX control using the Win32 API and C++.

Unlike JAVA, ActiveX can also manage files.

Slowly but surely, ActiveX technology has taken over about 80% of the specific market.

The Visio® drawing component, available in Microsoft® Office Visio 2003, lets you build upon the functionality of rich diagramming capabilities through Visio automation. The Visio component can be embedded in a number of containers, ranging from Office documents to custom applications built in Visual Studio® .NET, so it's quite versatile as well. In this article, we'll show you how to drive the Visio drawing component from a Microsoft .NET-based client application to give your users the ability to manipulate graphics.

The Visio drawing component is essentially an ActiveX® wrapper for the main Visio library. Like any ActiveX component, it can be embedded in custom host applications developed using Visual Studio .NET 2003, Office 2003 containers (such as Microsoft Word), Microsoft Internet Explorer 5.5 and greater, and other ActiveX component containers. Although the component uses the ActiveX architecture, it is more of an embeddable application component than the typical ActiveX control found on a Web site. The Visio component provides the full functionality of the Visio application rather than a small subset of features. The Visio control differs from the Visio client application in two ways. First, the drawing control only supports single document interface (SDI) windows. The control architecture supports a single drawing in a single window. That means that all multiple document interface (MDI) windows in Visio are disabled in the control context. The Visio component allows access to neither the ShapeSheet window (although you can program against the ShapeSheet), nor the icon editor window. Second, the drawing control does not load Visual Basic® for Applications (VBA) or run VBA code. That means that any existing VBA code must be ported to host application code. It also means that users never see the Visio macro warning dialog when opening a Visio document in the control.

The best practice when using the Visio component is to intercept Visio events and display your own custom user interface. This allows you to integrate the Visio component more deeply into your application. It provides the Negotiate Toolbars and Negotiate Menus properties on the drawing component. Set these two properties to True and you can expose the Visio toolbars and menus in the drawing component using the application's CommandBar object or UI object.

The Microsoft® Office Visio® 2003 ActiveX® Control (Visio drawing control) offers the full functionality of the Visio application through the rich Visio object model,

as an embeddable component. You can drive the Visio drawing control programmatically by events or by code in your hosting application. Alternatively, the Visio drawing control can provide a diagramming environment for application users within the context of your own application's user interface (UI).

The Visio drawing control provides the functionality of the Visio application object model in a component. The Visio drawing control can be embedded in host applications developed using Microsoft Visual Studio® .NET 2003, Microsoft Office XP and Microsoft Office 2003 containers such as Microsoft Office Word 2003, Microsoft Internet Explorer 5.0 and later, and other Microsoft ActiveX® control containers. Once added, the Visio drawing control provides a drawing surface for displaying shapes.

This new level of integration allows the developer full control over the Visio user interface integration with the host application. The new functionality provides more power than just simply embedding a Visio drawing into an OLE container document, like Word. Using a Visio drawing as an OLE object allows you to view the diagram in the container application, link the OLE object to the actual Visio document to reflect changes, and edit the Visio drawing by activating the Visio application from within the container document.

In the case of in-place OLE activation, you are still working within the Visio user interface. You cannot create your own UI. You can't programmatically access the Visio document using Automation from the containing application. You are also limited to application hosts that implement an OLE container, which rules out technologies such as .NET Windows Forms.

Because the Visio drawing control is a programmable component, you can integrate your Visio solution code directly into the host container application. In previous releases of Visio, a developer writing a solution for the Visio client application needed to package the solution code in a COM add-in, Visio solution library (VSL), out-of-process executable, or in a Visual Basic for Applications (VBA) project in a document.

The Visio control simplifies solution architecture and the development process by allowing programming of the Visio Application object from the hosting application. A developer using the control in a custom application (such as a C# application) or Internet Explorer should program against the Visio object model directly in the host application, without separating the Visio logic into a COM add-in, VSL, or executable. Calling a Visio COM add-in, VSL, or executable from the hosting application unnecessarily complicates the debugging of the Visio integration.

The Visio drawing control supports many containers, including hosting applications built in Visual Studio. NET 2003, Visual Studio 6.0, Internet Explorer 5.0 or later, ASP.NET, and other ActiveX control containers. However, the Visio control cannot be directly embedded in:

- Another ActiveX control, like the Internet Explorer browser control.
- A Visio document.
- A Microsoft Office InfoPath™ 2003 form.

Note You can embed ActiveX controls into the Visio drawing control (as opposed to the Visio drawing control embedded in another ActiveX control). However, because the Visio drawing control will not execute VBA code in a Visio document, it's better to integrate other ActiveX controls into the host application project instead of the Visio document.

Internet Explorer 5.0 or later provides an excellent host container for the Visio drawing control, allowing developers to write script against the Visio object model in Microsoft Visual Basic Scripting Edition (VBScript) or ECMAScript as defined by the specification of the European Computer Manufacturers Association, such as JScript or JavaScript.

While you cannot embed the Visio drawing control directly onto an InfoPath 2003 form, InfoPath provides a solution task pane containing an Internet Explorer window. You can embed the Visio drawing control directly into the InfoPath solution task pane window and use the Document Object Model (DOM) to share data between the Visio drawing and

the InfoPath form.

When designing an application that uses the Visio drawing control, it's important to understand that the Visio control supports a single document in a single window. The control's single document interface (SDI) architecture results in the following considerations when designing the Visio drawing control integration with your application:

- **Use multiple instances of the Visio drawing control to display multiple Visio documents in your application.** Unlike the Visio client application, which can display multiple documents and windows at a time, the Visio drawing control can only display a single document per instance of the control. If the developer wants to display multiple Visio documents, the developer can embed multiple instances of the control in the application, with each instance loading a separate Visio document.

- **Do not depend on VBA for programming logic.** VBA is not included with the Visio drawing control. As a result, documents loaded in the Visio drawing control do not execute any VBA code associated with the document. The control's lack of a VBA run-time environment prevents the distribution of legitimate code or malicious macros via documents loaded by the Visio control. It also means that the user of an application hosting the control will never see the Visio application's security dialog box warning about macros in the document.

- **Use the Visio ShapeSheet® programmatically.** The control does not provide access to the Visio ShapeSheet user interface, which is a separate window in the Visio application. However, the ShapeSheet itself still exists for the Visio shapes and pages in the document loaded in the Visio drawing control. You can still edit ShapeSheet cells for your Visio document in the control using Visio Automation.

Use marker events for COM add-ins. If you must use a COM add-in, you will need a way to notify your COM add-in to respond to a user action in your document. Use the QUEUEMARKER function in the ShapeSheet or a persistent event with the document to queue a marker event to which your COM add-in responds. When you are porting your code from an existing Visio client application solution to the Visio control, include these design considerations in your planning.

- **Port VBA code.** You must port all existing VBA code into a COM add-in, or more preferably, your host application. You can keep most of your same algorithms and logic, as long as it makes sense working within the control's SDI architecture.
- **Port an existing Visio solution COM add-in, executable, or VSL to the container application.** While it might be initially easier to simply use an existing COM add-in, executable, or VSL with the host application, it is recommended that you integrate the Visio drawing control programming directly with the host application. By taking the time to port your code from your existing solution to your host application, you'll simplify deployment of your solution, and streamline the development, debugging, and maintenance process over time. You do not need to use a COM add-in unless you are working with the control in another Office container.
- **Re-evaluate data storage in shapes.** If your current Visio solution stores extensive data in shapes, consider re-architecting data storage out of Visio shapes and into data structures maintained or accessed by the host application. Visio can store as much data as you need, but it often makes more sense to keep Visio as a presentation layer component and use your host application as data storage or access to a data source. If the data is relatively static, though, and used heavily for modifying the appearance and layout of shapes, it makes more sense to store the information in the shape custom properties.
- **Think about event integration with the host application.** Unlike a Visio client application solution, the Visio drawing surface can respond to both user events and host application events. Consider how you want the drawing surface to respond to events fired by the host application, and how your host application needs to respond to events in the

Visio drawing surface. You can easily create a Visio drawing control instance in managed code. Completing this procedure to add the following assembly references to your project: **VisioOcx** The Visio drawing control primary interop assembly; **Visio** The Visio type library primary interop assembly; **AxVisOcx** An ActiveX control wrapper assembly that allows the control to be embedded within a Windows Form. The Windows Forms ActiveX Control Importer (Aximp.exe) generates this assembly automatically and adds it to your project. This wrapper assembly must be redistributed with the application. Multiple Visio drawing control instances on a form share the same wrapper class derived from the **AxHost** class.

Exposing the Microsoft as a Web Part

Web Parts are components of a Web Part Page that can be easily managed to define team workflow. A SharePoint site is composed of multiple Web Part Pages and Web Parts. Example Web Parts include the **Shared Documents** Web Part, which lists the files in a document library and provides menu items for checking files in and out of a document library.

The Microsoft Office Visio® 2003 ActiveX® Control, also known as the Visio drawing control, can be rendered in a Web Part. This allows a Visio document to appear on a page on a SharePoint site. However, a more compelling reason for implementing a Visio Web Part is that Web Parts can use standard interfaces to communicate with each other. Users can select from a menu to connect Web Parts that can exchange data, and these Web Parts can be developed completely independently of each other.

As a Web Part, the Visio drawing control provides an ideal smart client for a Web service. The Visio drawing control can serve as a visualization display surface for data provided by a Web service, such as the Human Workflow Services (HWS) available from Microsoft BizTalk™ Server. HWS provides a Web service API that client applications can use to provide workflow capabilities to the end user. The Visio drawing control gives the corporate developer the ability to track workflow information to the end user graphically.

A basic implementation for a Web Part inherits from the **Microsoft.SharePoint. WebPartPages.WebPart** class and overrides the **RenderWebPart** method. Additionally, your Web Part project must contain the following:

- A reference to the Microsoft.Sharepoint.dll assembly.
- A Web Part class file and a matching .dwp file. A .dwp file is an XML file that references a Web Part's assembly, namespace, and class name, plus optional property settings for the Web Part.
- A Manifest.xml file for the Web Part that contains XML elements describing setup information. The Manifest.xml file is used by the Stsadm.exe tool during installation of a Web Part.

Web Parts share common properties, such as **Title**, **Height**, and **AllowClose**. These properties are displayed in a frame when the site administrator adds a new Web Part or modifies an existing one. Developers can add custom properties to Web Parts by adding standard .NET properties to the Web Part class implementation.

Microsoft Active X technology proved to be very up to date as it has been included and integrated in .NET standards and it is still expanding an the IT market, especially for the Internet.

Bibliography

1.Kenneth Lassesen, ADODB: ActiveX Data Objects 2.1, Microsoft Corporation

2.Nancy Winnick Cluts, Creating ActiveX Components in C++, Microsoft Corporation

3.Nancy Cluts, Microsoft ActiveX Controls Overview, Microsoft Corporation

4.W Ernst, The Components of ActiveX, O'Reilly &Assoc.Press,1998

5.Mark Bukovec, Programming with the Microsoft Office Visio 2003 ActiveX Control, September 2003

6.*** - ActiveX – http//www.Microsoft.com

7.*** - VBScript &ActiveX– www.Activex.org