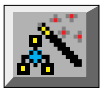

by Richard Hale Shaw

GET STARTED WITH CONTROLWIZARD

CREATE OLE CONTROLS QUICKLY IN VISUAL C++ WITH THE CONTROL DEVELOPMENT KIT AND ITS WIZARDS.

Last month I showed you how to create a simple OCX that subclasses a Windows edit control, and how that control might be used in a future VB or VC++ environment. This month, I want to show you what actually takes place when you use the OLE Control Development Kit (CDK) and its ControlWizard component to create a new OCX. In subsequent columns, I'll examine the steps required to complete a control of your own.

As we get closer to the release of new visual development environments—environments that let you use OCXs as well as create them—I'll explore the issues of interaction between OCXs and the applications that actually can use them: OCX control containers.

But frankly, it irks me to no end that you can't use OCXs in any VB or VC++ development environment on the market as of March 1995. Yeah, yeah, you can use Access or Visual FoxPro, but I'm talking about language-based visual development environments, in contrast with database systems that double as development environments. Until we can really work with and test OCXs in a visual development environment based on VB, OCX development is going to go nowhere fast.

You know what I'm talking about when I refer to such an environment. But NDAs being what they are, I can't say so in print, at least. Microsoft's VB team has given me very specific, limited permission to show how to use OCXs in my talks at conferences such as VBITS. So, until these new environments get closer to release, and the trade press starts its usual round of NDA-breaking, I'll be vague about the unreleased environment in which you can use OCXs. I can say this: it should arrive sometime this year, and hopefully it won't be too many months away by the time you read this.

Added to \WINDOWS\SYSTEM:	
OC25.DLL	Release version: 16-bit OLE control
OC25D.DLL	Debug version: 16-bit OLE control
Added to \WINDOWS\SYSTEM32 (Windows NT):	
OC30.DLL	Release version of 32-bit OLE controls with ANSI/DBCS support
OC30D.DLL	Debug version of 32-bit OLE controls with ANSI/DBCS support
OC30U.DLL	Release version of 32-bit OLE controls with Unicode support
OC30UD.DLL	Debug version of 32-bit OLE controls with Unicode support

TABLE 1 ***Signs That the CDK Moved In.*** *The Control Development Kit Setup program installs two 16-bit and four 32-bit OCX DLLs into your system directory.*

INSIDE THE OCX CDK

The key to building OCXs is Visual C++ 2.x, which comes with a version of the OLE CDK already on the CD, offering support for building both 16- and 32-bit OCXs. To build 32-bit OCXs, you must install the 32-bit CDK. For Win16 OCXs, use the 16-bit CDK. In either case, the installation program will add a subdirectory (by default, CDK16 or CDK32) of files to your MSVC or MSVC20 directory structure, or somewhere else if you choose.

A 16- or 32-bit ControlWizard for creating OCXs and a Test Container for testing them have been added as well. If you're using VC++ 1.5, the CDK setup program will install a new ClassWizard that includes support for adding Events to your OCX. The 32-bit ClassWizard in VC++ 2.0 already has this feature, and doesn't need to be replaced.

The bottom line? You'll need VC++ 2.x, but it contains everything you need to create both 32-bit and 16-bit OCXs.

That's the good news. But you'll need to know C++ as well as Visual C++, MFC, and their basic OLE support if you're going to be productive building and creating your own OCXs. If you're familiar with MFC, especially the CWnd and CDialog classes, you'll feel right at home with the MFC extensions that provide OCX support. But if you don't already know how to take advantage of the Visual C++ environment and MFC, you'll be lost.

Thus including a Visual C++ programming column such as this one, in a Visual Basic programming magazine such as this one, makes sense after all. Your job is to be productive with VB. My job is to help you be productive where VC++, and the OCX CDK in particular, can help.

The OCX CDK specifically gives you the OCX ControlWizard, MFC extensions to encapsulate OCXs as MFC objects, a bunch of source code samples, online documentation, and the OCX Test Container application. Plus, you'll use the VC++ ClassWizard for adding OCX properties, methods, and events to an OCX (the update to ClassWizard in VC++ 1.5 is required: the original ClassWizard does not have OLE Events support).

At press time (March 1995), I had received VC++ 2.1, which

Richard Hale Shaw is a contributing editor to Visual Basic Programmer's Journal, PC Magazine, Windows Tech Journal, and Microsoft Systems Journal. *He's currently completing* Visual Programming++, *a book about Visual C++ and MFC programming, to be published by Addison-Wesley. He lives in Ann Arbor, Michigan and can be reached on CompuServe at 72241,155 or the Internet at 3998368@mcimail.com.*

includes additions to both 16- and 32-bit MFC (2.52 and 3.1), as well as a few new goodies in AppWizard for creating in-process OLE Automation servers.

In addition to those components, the CDK install adds a tool for registering a control (you must update the registry before you can use a control), and if you've installed the 16-bit CDK on an NT machine, a batch file, BLDTYPLB.BAT, will be installed for creating type libraries.

In any case, your VC++ Tools menu will be updated to include entries for running ControlWizard, registering the control, and running the Test Container. And if you're running VC++ 1.5, the Tools menu will include an entry for building the type library with BLDTYPLB.BAT as well. Type library builds will be automatic with VC++ 2.0, because they're added to the make process by ControlWizard. Finally, the CDK SETUP will install a bunch of new DLLs on your system (see Table 1).

The COleControl Interface

BoundPropertyChanged	Notifies the container that a bound property has been changed.
COleControl	Creates a COleControl object.
ControlInfoChanged	Called after the set of mnemonics handled by the control has changed.
DisplayError	Displays stock Error events to the control's user.
DoPropExchange	Serializes the properties of a COleControl object.
DoSuperclassPaint	Redraws an OLE control that has been subclassed from a Windows control.
EnableSimpleFrame	Enables simple frame support for a control.
ExchangeExtent	Serializes the control's width and height.
ExchangeStockProps	Serializes the control's stock properties.
ExchangeVersion	Serializes the control's version number.
OnClick	Called to fire the stock Click event.
OnDoVerb	Called after a control verb has been executed.
OnDraw	Called when a control is requested to redraw itself.
OnDrawMetafile	Called by the container when a control is requested to redraw itself using a metafile device context.
OnEdit	Called by the container to UI-Activate an OLE control.
OnKeyDownEvent	Called after the stock KeyDown event has been fired.
OnKeyPressEvent	Called after the stock KeyPress event has been fired.
OnKeyUpEvent	Called after the stock KeyUp event has been fired.
OnProperties	Called when the control's "Properties" verb has been invoked.
OnSetExtent	Called after the control's extent has changed.
OnSetObjectRects	Called after the control's dimensions have been changed.
OnShowToolBars	Called when the control has been UI activated.
SelectFontObject	Selects a custom Font property into a device context.
SelectStockFont	Selects the stock Font property into a device context.
SetControlSize	Sets the position and size of the OLE control.
SetInitialSize	Sets the size of an OLE control when first displayed in a container.
SetModifiedFlag	Changes the modified state of a control.

TABLE 2 ***The COleControl Interface Functions.*** *Here are a handful of the 100-plus member functions in COleControl. You'll find the entire class documented in the online help file that comes with the OLE CDK.*

THE CDK AND MFC

The OLE CDK adds several new classes to MFC that encapsulate support for building OLE Controls. You need to know about two key classes: COleControl and COlePropertyPage. You derive a class from COleControl to represent the control you're building, and derive another from COlePropertyPage to add support to the dialog or form used for displaying or setting the control's properties at design time.

COleControl is directly derived from MFC's CWnd class, and consequently inherits all of CWnd's standard windowing functionality. COleControl adds to CWnd the ability to fire events, as well as method and property support, although some of this was already in place with additions to CCmdTarget—the MFC mes-

IF YOU'RE FAMILIAR WITH MFC, YOU'LL BE AT HOME WITH THE MFC EXTENSIONS THAT PROVIDE OCX SUPPORT.

saging class, and the base class of CWnd—as far back as MFC 2.5. COleControl also allows a control to be inserted into an OLE container application.

The method and property support provided by COleControl and CCmdTarget make it possible for the container application (the application that *uses* the control) to change data in the control, or to trigger services provided by the control. But how does the control post a notification back to the container? You may have read my earlier Visual Programming column discussing the whole issue of callback notifications, and how an automation client can expose its own IDispatch interface to let a server call back into it [*VBPJ* March 1995]. So you'd think that a control would have to do the same—and it does. But the control's IDispatch is cleverly hidden and encapsulated by COleControl's support for OLE events. By defining events in the control, you can specify the mechanism for posting notifications back to the control container application, and then add event handling to the container to respond (I'll discuss events in detail in a future column).

COleControl offers more than 100 member functions (see Table 2) that provide support for control data persistence, ambient and stock properties and methods, events, and data binding, to name a few. In this respect it rivals its parent. CWnd itself is one of MFC's largest classes, with around 150 member functions. As I develop controls in this column over the next several months, I'll explore even more of COleControl's capabilities.

COlePropertyPage is, by contrast, a much less complex class to deal with: you'll basically use it to design a form-based UI so a developer can access and change a control's properties. With only 15 member functions (see Table 3) and derivation from MFC's CDialog class, adding property access is not terribly complex. I'll use this class in future columns to add a dialog control to let your OCX users change the values of properties exposed by the control.

USING CONTROLWIZARD

Once you're ready to build your first OCX, ControlWizard is the tool you use to get started. Like AppWizard, ControlWizard can

generate the code for a starter OCX, and lets you configure the generated code through a series of dialogs. Actually, it more closely resembles the VC++ 1.5 AppWizard, which requires that you know how to navigate through it, rather than the VC++ 2.0 AppWizard, which prompts you for the information it needs.

ControlWizard will generate the starter files you need to create a new OCX, including header and source files, resources, the make file, the ODL script for compiling a type library, and the module-definition file for building 16-bit OCXs.

While it's not necessary to use ControlWizard, you'd be crazy not to for two reasons. First, it gets you started with an OCX ready to be compiled and run. Second, it inserts all the pieces necessary to continue using tools such as ClassWizard so you can do as much of the work as possible using visual programming techniques.

If you're a masochist and insist on writing all your code by hand, don't bother with ControlWizard (or this column, for that matter). When you're done with ControlWizard, the generated code will include basic control drawing and data serialization facilities, the appropriate message and dispatch maps for supporting properties and methods, and even event maps.

As I mentioned, the CDK Setup program automatically adds a ControlWizard entry to the Tools menu of your VC++ environment. So when you're ready to build a new OCX, start by launching ControlWizard.

ControlWizard has the usual, AppWizard-style prompts (requesting the name of the project, class names, file names, whether you want to create a VC++-compatible make file, and whether the source code includes comments, for example). You can specify a project name that's used for the make file name, as well as the name of the OCX file. OCXs are stored in a DLL with an OCX extension. AppWizard uses the project name to create

The COlePropertyPage Interface

COlePropertyPage	Constructs a COlePropertyPage object.
GetObjectArray	Returns the array of objects being edited by the property page.
SetModifiedFlag	Sets a flag indicating whether the user has modified the property page.
IsModified	Indicates whether the user has modified the property page.
GetPageSite	Returns a pointer to the property page's IPropertyPageSite interface.
SetDialogResource	Sets the property page's dialog resource.
SetPageName	Sets the property page's name (caption).
SetHelpInfo	Sets the property page's brief help text, the name of its help file, and its help context.
GetControlStatus	Indicates whether the user has modified the value in the control.
SetControlStatus	Sets a flag indicating whether the user has modified the value in the control.
IgnoreApply	Determines which controls do not enable the Apply button.
OnHelp	Called by the framework when the user invokes help.
OnInitDialog	Called by the framework when the property page is initialized.
OnEditProperty	Called by the framework when the user edits a property.
OnSetPageSite	Called by the framework when the property frame provides the page's site.

TABLE 3 ***Functions of the COlePropertyPage Interface.*** *Dealing with COlePropertyPage is fairly straightforward. Its main purpose is to design a form-based user interface that gives developers a means to customize a control.*

the class names, but you can override this if you'd prefer.

The License Validation option will cause ControlWizard to generate an LIC file. The LIC file approach is identical to the scheme used by most VBXs, where the LIC file has to be present in the directory where the control is stored, in order to use the control at design time. The LIC file doesn't have to be present in order to use the control from a control container application. With this option checked, ControlWizard generates the LIC file for you, along with some code that will automatically be invoked at design time to check for the presence and validity of the LIC file. (Again, this will be covered in more detail in a future column.)

Another option, Show Insert Object Dialog, will make the control appear in the dialog that's displayed by an OLE Object Container in response to the user selecting the Object command from the Insert menu. The option will also add support for an "Edit" verb to your control's message-map, if selected. Don't select this option lightly: most current OLE containers aren't yet aware of controls. They can let you embed or insert an object server object, but they can't communicate with such an object that's managed by an OLE control. Because controls are hybrids of an object server and an automation server, only special types of applications—OLE control containers—can communicate with them and use them. So for the time being, leave this option unchecked.

Other options control the physical appearance of the control during its use. For instance, the Activate When Visible option will add code to tell the container to activate the control whenever it's visible. Another option, Simple Frame, lets you create a control that includes a simple window frame. Yet another prompts you to add an About Box to the control, if you wish.

The Invisible At Runtime option lets you create a control that is visible during program design, and invisible when used by the control container application. While some controls, such as those that subclass a Windows control, are meant to be displayed at both design and run time, others are not. This option lets you build a control that provides services to be used, but has no user interface; it shouldn't be seen by the user while the program is running. A good example is the Data Source VBX that Coromandel uses to encapsulate an ODBC data source: there's no reason for the user to see this control, but there's every reason for you to interact with it at design time.

Files Generated by ControlWizard

(Note: *prjname* is used in place of the project name you specify.)

prjname.MAK	The make file for this OCX.
prjname.CLW	The ClassWizard file of classes, events, messages, etc.
prjname.ODL	The Object Description Language script used to create the control's type library.
prjname.RC	The control's resource script.
RESOURCE.H	The common header file of resource IDs.
*prj*NACTL.BMP	The bitmap used when the control is displayed in a toolbar or palette.
prjname.DEF	The Windows' module-definition file for this OCX.
prjname.H	The OCX class header file with a CWinApp-derivative definition.
prjname.CPP	The OCX class implementation file with a CWinApp-deriviative implementation.
*prj*NACTL.H	The COleControl class derivative definition.
*prj*NACTL.CPP	The COleControl class derivative implementation.
*prj*NAPPG.H	The COlePropertyPage class derivative definition.
*prj*NAPPG.CPP	The COlePropertyPage class derivative implementation.
STDAFX.H	A Visual C++ precompiled header file.
STDAFX.CPP	A Visual C++ precompiled header implementation file.
MAKEHELP.BAT	A batch file for compiling RTF files into HLP files (if the Help option was selected).
prjname.HPJ	The Help project file.
CTRLCORE.RTF	The Rich Text Format file for creating on-line help.
*.BMP	Bitmaps used by the RTF file.
prjname.LIC	The LIC file used for controlling licensing (if option selected).

TABLE 4 ***Files Generated by ControlWizard.*** *Once ControlWizard finishes its magic, it creates a number of files for your project, each with a specific purpose. In this table,* prjname *is used in place of the project name you specify.*

BEGINNERS' BEST OPTIONS

Perhaps the two most important options, particularly for the beginner, are Subclass Window Control and Use VBX Control As Template. The Subclass Window Control adds code to specify the window class of the control. You can select Edit, Listbox, and Button, for example, to subclass standard Windows controls in your OCX and create versions of them customized to your own liking. This was the approach I used last month to create the PhoneEdit OCX.

The Use VBX Control as Template option lets you use a VBX on your system as the basis for a new OCX. When you select this option, ControlWizard will let you specify a VBX for it to read. Then it will use the interfaces in the VBX as the basis for the interfaces in the new control, with the same properties, and with a corresponding event structure.

While you must add the code to implement the new OCX interfaces, much of the groundwork is done for you by ControlWizard. Use this option if you're porting a VBX of your own to the OCX architecture, or if you have a favorite VBX that the VBX vendor is unwilling or unable to port to OCX. While you must add the implementation code, the resulting OCX will have all the same interfaces and should work just dandily with a new version of that VB application you wrote to use the VBX, once you've ported the application to VB4 (hopefully, later this year).

Finally, ControlWizard lets you name the classes and files it generates, and additionally lets you add more than one COleControl derivative to the project. This means you can create more than one OLE control that can be stored in the same resulting OCX file (a DLL, remember?). You can also use these options to specify the name by which the control will be known in the system registry.

If you use ControlWizard to generate an OLE Control project, you'll end up with a group of files that include a COleControl derivative for each control you specified. The default name for a single control class will be C*projectname*Ctrl, where *projectname* is the name of the OCX project you're building. Like any Visual C++ application, this same approach is used to name the header and implementation file for this class. There'll also be a C*projectname*PropPage class for the control's property page, and ControlWizard will generate a default dialog template for the control's first property page. In both cases, ControlWizard lets you override the class name and file names before you generate them.

Finally, ControlWizard generates a number of files (see Table 4). Just what's in these files, and what does the code in them do? I'll take that up in my next column. ■