CenturaPro™

Visit us at www.ProPublishing.com! Hot Ideas for Centura® Developers

Applying ActiveX in CTD 1.5

Tony Vinayak

Raise your hand if you've programmed DDE with disdain. Nod your head if you fell in love with OLE2—almost. Heads up if you want to know all about ActiveX implementation in CTD 1.5.

Now that Microsoft has finally gotten it right with COM—after earlier attempts with DDE, OLE, and OLE 2—Centura Team Developer has caught up to supporting the technology with the 1.5 release. In this release, CTD supports ActiveX objects of various kinds. But first, a bit of a background on COM and ActiveX, before we go full steam ahead with the implementation details.

What is COM? Isn't ActiveX equivalent to COM? Isn't it souped-up OLE 2 anyway? Don't blame yourself if you're a bit confused about the ever-changing terminology parlance—blame Microsoft. Let's not get into the definitions war; there's enough out there already to quench your curiosity. Have a look, for instance, at www.microsoft.com/com for a plethora of white papers on the topic. For now, let's summarize what we really need to get going.

SQLWindows 32

What's COM?

The Component Object Model promotes the virtues of "plug-and-play" development, by encouraging development and usage of re-usable components. COM is a programming framework that dictates how re-usable objects should be constructed and shared with other applications on the same machine and/or across the network. COM objects expose their functionality via "interfaces." An interface is a collection of like-minded function calls that can be invoked by other applications. A COM object can sport multiple interfaces. Major benefits that accrue out of COM technology include:

- Binary rather than source-code re-usability. Components built using Visual Basic can be used by CTD developers.
- Transparent cross-platform interoperability. You don't have to worry about the location of the COM object, whether it's local or remote. The underlying COM framework takes care of the necessary "data marshalling" to connect your application to the COM objects.
- Compatibility across versions of components.

Even though COM is primarily available on Windows platforms, Microsoft is fully aware of the fact that to make it a compelling proposition for the IT managers, COM (and DCOM) will have to be made available across a variety of platforms like Unix and Macintosh.

Continues on page 3

March 1999

Volume 4, Number 3

PRO Publishing

Applying ActiveX in CTD 1.5 …

Continued from page 1

What's ActiveX then?

The COM specification has been around since 1993. OLE 2 was the first widespread implementation of COM technology. On an on-going basis, COM is finding its way into practically everything coming out of Redmond: Microsoft Office, Windows file system, Distributed Internet Architecture (DNA), etc. Marketing machine that Microsoft is, initially OLE 2 was chosen as the "brand name" for COM. Subsequently, ActiveX became the harbinger. Officially, ActiveX is "…a group of technologies implemented using COM." Lately, ActiveX controls are gaining prominence, especially since they're lightweight and designed with the Internet in mind. But remember, ActiveX means lots more than those flashy ActiveX controls that you download into your Web browser. Various components of MS Office act as ActiveX "servers," the heavy-duty workhorses.

Look at **Figure 1**.

The container application could be a Windows application developed using any of the popular development tools, including CTD, which invokes the functionality of COM objects. The diagram shows various possible process locations of COM objects (I'll use the terms COM and ActiveX interchangeably from here on out in this article). Of course, the container application doesn't have to concern itself with location details as long as the COM object is registered locally on that machine. All that the container needs to know is the interface. Which brings us to a bit of a background on COM automation.

COM automation

Initially, when the COM framework was developed, it required the container application to get a pointer to the required interface function of the COM object. That was great for languages like C++, but left the 4GLs out of contention, since they generally don't support pointers. At the behest of its Visual Basic development group, Microsoft came out with "COM Automation,", which doesn't require pointer manipulation. Any COM-automation-compliant object sports an interface called "IDispatch" (all interfaces, as a notion, begin with an "I"). IDispatch provides a function called "Invoke," which acts as a funnel to all other interfaces (and properties) of the object. The good news is that practically all COM objects out there provide this interface (also known as "dispinterface"). Microsoft greatly encourages its usage, to enhance its universal appeal. As a matter of fact, if you examine the MFC code generated by the ActiveX wizard in VC++ 6.0, you'll find that the COM objects are being accessed via their dispinterfaces. It should, then, come as no surprise to you that CTD 1.5 also uses dispinterfaces.

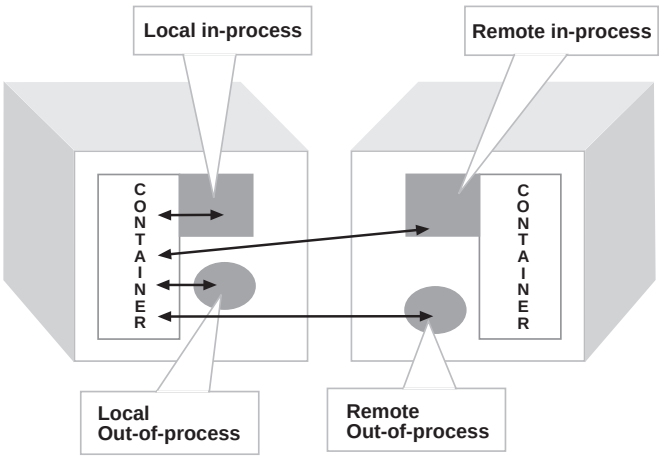

Figure 1. In-process and out-of-process COM components.

COM support in CTD 1.5

CTD 1.5 provides a much-improved interface for invoking ActiveX components. Prior to that, CTD provided QuickOLE quick object and some SalOLE* functions, which had several limitations (remember good ol' Uncle Fred?). All that's history now. Here's the laundry list of things you can now do, rather comprehensively, with the new ActiveX support:

- All the ActiveX components registered on your machine are listed in the controls palette (**Figure 2**). You can then drop one of them (such as the MS Calendar control) onto your design window, and CTD automagically generates the supporting functions for accessing its properties and methods.

- You can use the Attributes Inspector at design time to view and modify the ActiveX component's properties (**Figure 3**).

- The Outline Options box lists the events supported by the ActiveX component, just like it lists all other messages. The events, however, are a tad different; they come loaded with parameters, which can contain much more information than merely wParam and lParam. The parameters are listed separately in the outline (**Figure 4**).

Figure 2. The Controls Palette lists all the ActiveX components registered on your computer.

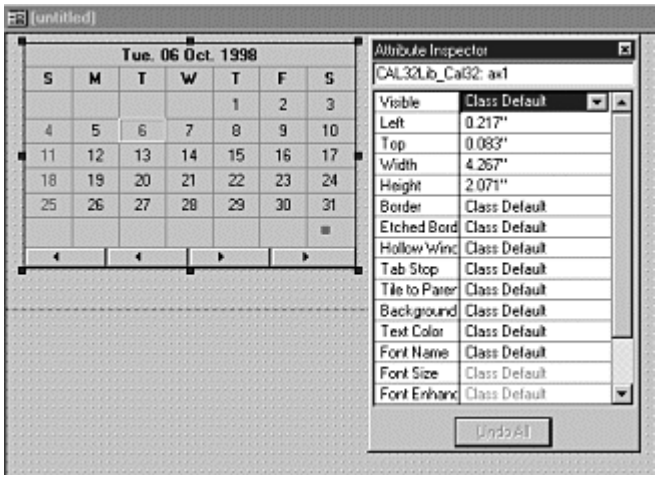

Figure 3. You can use the Attributes Inspector to view and edit an ActiveX control's properties.

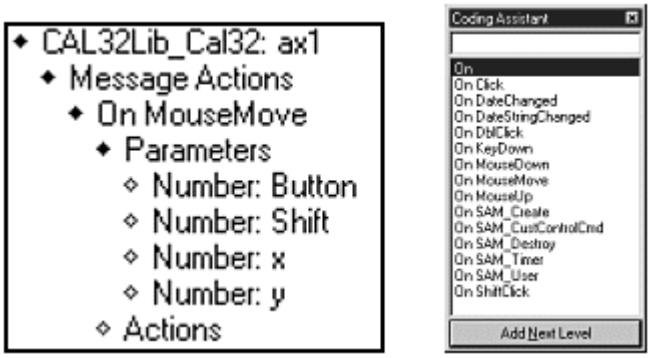

Figure 4. The Outline Assistant lists all events supported by the ActiveX control.

- An ActiveX wizard guides you through generating SAL functional classes for interfacing with the ActiveX components (**Figure 5**). These functional classes, as a matter of fact, are wrapper functions around COM automation's Invoke( ) function. This not only makes your code more readable, it also provides better type-checking at compile-time. CTD ships with Automation.APL, which contains all the base classes necessary for COM automation: Object, Variant, SafeArray, and OleErrorInfo (for error handling). We'll put them to use later on in the article.

- You can embed ActiveX components like Word or Excel into your own windows. This is supported by a pre-mediated UI negotiation. Check out the new "ActiveX Menu Group" in the menu editor. Online help talks about the menu-merge algorithm. In a subsequent release of CTD, you can also expect ActiveX linking (using monikers).

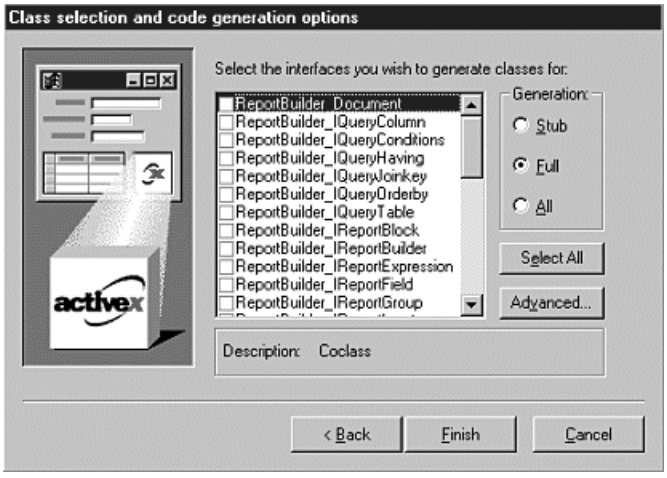

Figure 5. The ActiveX Wizard.

Automating Word 97

With basics out of the way, let's get going with the real business: remote controlling ActiveX servers like MS Office. Let's focus on Word.

While there are at least a couple of versions of Word out there, for now we'll focus on Word 97. While all this discussion in principle would also be applicable to Word 95, the code implementations would differ. Why? Because Microsoft changed the Word object hierarchy a bit and made Visual Basic for Applications (VBA) the standard macro language for all Office 97 apps.

Before unleashing CTD 1.5's ActiveX wizard on Word 97, stop. Take a deep breath. Roll up your sleeves, and go get that Office 97 CD. To get the necessary online documentation on the Word object hierarchy and help on various Word methods and properties, you'll need to install the Visual Basic Help for Word. By default, it's *not* installed by the standard Office 97 installer. You'll find it buried under Word Help install options. To invoke online help for Microsoft Visual Basic for Word, start Word and choose Tools | Macro | Visual Basic Editor. Now you can press F1 to invoke the online help. A quick search on "hierarchy" will lead you to the Word Objects Hierarchy diagram, shown in **Figure 6**.

Become familiar with this hierarchy. You'll quickly realize, for instance, that to get anything meaningful done on a Word document, you'll need to traverse the *Application —> Documents —> Document* hierarchy. The Document object, in turn, contains other objects like Bookmarks, Characters, and Tables. This Object hierarchy is actually much better organized than in previous versions of Word, where there was a single object with hundreds of methods.

Once you run the ActiveX Wizard on the Microsoft Word 8.0 Object Library, it generates functional classes for various objects in the Word Objects hierarchy (not

necessarily conforming to the hierarchy for class derivation, though. In fact, most generated functional classes are derived from the "Object" class provided by Automation.apl). One thing though: In the Wizard, in the "Class selection and code generation options," make sure to choose the Full radio button and Select All pushbutton. This will generate classes for the entire Word Object hierarchy. Don't use the All radio button though; it renders your code non-compilable! It so happens that various Office modules, courtesy of Microsoft, share common naming conventions for their constants. When the Wizard generates code, it creates the following APLs:

- "Microsoft Word 8.0 Object Library.apl"
- "Microsoft Office 8.0 Object Library.apl"
- "Microsoft Visual Basic for Applications Extensibility.apl

Each of these APLs, alas, have some similarly named constants, which will all be merged together by the All radio button option. Besides, choosing All will unnecessarily bloat your code anyway.

Let's say, from your CTD application, you want to open a Word template and put some text in it. Sounds like a fairly common requirement; the marketing department has created a wonderful marketing brochure as a Word template that needs to be mailed out to your customer base. Your CTD app is going to pull out the customer contact information from your relational database, and mail-merge it with the Word template. Now, I know that in Word I can create bookmarks, which can be programmatically replaced with text. Armed with the Word Object hierarchy, we kind of know where to start: at the Application object, of course. In the code generated by the ActiveX wizard, we notice the Word__Application functional class, which contains a bunch of member functions. But where do I go from there? There's an easy way to answer that question: Word's macro recorder. It generates the VBA code, which can then be retro-fitted into SAL. Let's do the following in Word:

1. Create a new document and type any arbitrary text in it. More importantly, create a bookmark and call it Name. Save the file as Market.doc.

2. Go to Tools | Macro | Record New Macro…

3. We need to see the code for opening a file and locating a bookmark. While the macro is recording, do just that: Open the file, "Marketing.doc," and choose Insert | Bookmark from the menu. That'll bring up the Bookmarks dialog box. Choose the bookmark called "Name," and click the "Go To" button. Stop the macro recorder. Now bring up the Visual Basic Editor, and it'll display the generated VBA code in it. It should look like the following:

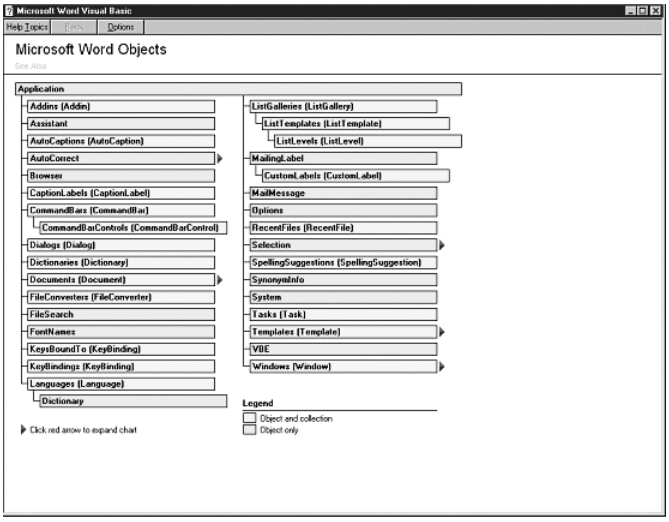

Figure 6. The Microsoft Word 97 objects hierarchy.

```
Sub Test()
Documents.Open FileName:="Market.doc",
  ConfirmConversions:=False, ReadOnly:= _False,
  AddToRecentFiles:=False,
  PasswordDocument:="", PasswordTemplate:= _"",
  Revert:=False, WritePasswordDocument:="",
  WritePasswordTemplate:="",
  _Format:=wdOpenFormatAuto
  Selection.GoTo What:=wdGoToBookmark,
  Name:="Name"
    With ActiveDocument.Bookmarks
    DefaultSorting = wdSortByName
    ShowHidden = False
    End With
End Sub
```

This VBA code yields the following vital pieces of information:

- Objects that need to be created (Documents and Bookmarks).

- Methods that need to be invoked (Documents.Open).

- Method parameters and their data types. The only missing piece is which of the parameters are optional.

You can get that information from the VBA online help for the "Open" method.

With a simple twist of fate, the SAL code for opening a Word document, finding a bookmark, and setting some text looks something like the following pseudo-code:

```
! Create a Word__Application object
Call WordApplication.Init()
!Make it visible
WordApplication.PropSetVisible (TRUE)
! Create a Word__Documents object
WordApplication.PropGetDocuments ( WordDocuments)
! Open a document. It takes variant params
Call vFileName.SetString ("Market.doc")
Call vConfirmConversions.SetBoolean (FALSE )
!… and a bunch of other params for the open method
!   initialized here (code omitted)
Call WordDocuments.Open (vFileName, vConfirmConversions,
    …, WordDocument)
! WordDocument in the above line is the opened document
! Get the bookmarks
Call WordDocument.PropGetBookmarks (WordBookmarks)
!Get to our bookmark named "Name", by looping
```

```
!through all bookmarks in the doc
WordBookmarks.PropGetCount (n)
Set nIndex = 1
Set bFoundBookmark = FALSE
While nIndex <= n
  Call vIndex.SetNumber ( nIndex, VT_I2)
  Call WordBookmarks.Item ( vIndex, WordBookmark)
  Call WordBookmark.PropGetName ( sName (
  If sName = "Name"
    ! got it
    Set bFoundBookmark = TRUE
    Break
  Set nIndex = nIndex + 1
If bFoundBookmark
  Call WordBookmark.Select_()
  Call WordBookmark.PropGetRange ( WordRange)
  Call WordRange.InsertAfter ("Whatever text!")
```

You'll find more finished code in the accompanying sample program, Word ActiveX Sample.app. Well, surely there's a bit more work that needs to be done in the SAL code as compared to what appears in the VBA code; but this is a good place to get the basic structure of your code.

Error handling

When things go wrong at runtime, which often happens while you're getting a grip on COM automation, who you gonna call? Case in point: When automating Word from your CTD application, the Documents.Open method, the VBA documentation says, takes one mandatory parameter (the file name) while all other parameters are optional. However, in your code, if you use the MakeOptional( ) method on all those other parameters (such as vConfirmConversions.MakeOptional( )), you get an ugly error message that looks like **Figure 7**.

Figure 7. Default error handling.

That doesn't tell you much, does it? This is the error message generated by the COM runtime environment. My suggestion: Turn off the default error handling and customize it. In the process, you get to know something more about the error too. Here's how:

1. Call SalActiveXAutoErrorMode (FALSE).

2. Create an instance of OleErrorInfo class in your code. This class is essentially a structure that holds error information.

3. Call the Object.GetLastError( ) method to fill up the OleErrorInfo structure.

Look at the accompanying sample code for the exact implementation. Custom error handling on the Documents.Open( ) method yields error number 2147352571. Microsoft Visual Studio provides a utility called ErrLookup.exe, which decodes this value (from WINERROR.H file) to "Type Mismatch." That's some further progress in your quest. To cut to the chase: when the ActiveX server application (Word in this case) says that a parameter is optional, it doesn't expect the parameter to be passed at all in the method invocation. Sadly, CTD doesn't support a variable number of parameters in function calling, and it needs to pass parameters, optional or not. And therein lies the problem. In this case, we really need to initialize those "optional" parameters to a zero, or "", or a FALSE value, as shown in the pseudo-code listing. By the way, it doesn't mean that MakeOptional( ) is never going to work. It's just that an "optional" parameter can have slightly different connotations for different ActiveX servers. In case of Word, either you don't pass in the optional parameter at all, but if you do, you'd better initialize it to a value! Clear as mud? **CP**

Download WordActiveXSample.ZIP from this issue's Table of Contents at www.ProPublishing.com or find it on this month's Companion Disk.

Tony Vinayak is a Senior Consultant with Categoric Software Corp., the makers of zero-latency corporate alerts. You can reach him at tvinayak@categoric.com.