Platforms, Frameworks & Libraries » COM / COM+ » ActiveX
Beginner

VC6Win2K, MFC, ATL, COM, Dev

Posted: **8 May 2001**
Views: **185,393**
Bookmarked: **69 times**

Handling VB ActiveX Events in Visual C++ client

By Amit Dey

This article shows how to handle custom events generated in a VB ActiveX component in a Visual C++ client.

Download source files - 86 Kb

Introduction

This tutorial is an extension to my article on building VC clients for VB ActiveX DLLs. Having read such an article, the question about handling events naturally comes to the reader's mind.

Here I'm going to show you how to handle custom events generated in VB ActiveX components in a Visual C++ client. First we are going to build a MFC client and then turn our attention to creating an ATL client. Doing all of this is not tough at all, as you shall see. A framework like MFC makes it very easy for the programmer to receive event notifications from an ActiveX code component. A word before we move on, I assume that the reader is conversant with VB ActiveX technology, Automation and MFC COM and IDL (Interface Definition Language).

VB ActiveX Components

A VB ActiveX component is a unit of code that follows the COM specification for providing objects. It exposes much of its functionality through one or more interfaces. These software components have a high reusability factor.
As the component needs to communicate with the client, they can be implemented as an **in-process** (read DLL) component and an **out-process** (read EXE) component.

In an in-process component (ActiveX DLL components), the communication between the server and the client is implemented in the address space of the client application. Though this makes them faster than ActiveX EXE components, who need to be loaded in their own address space, the biggest drawback is that a faulty DLL will crash the client, and in turn, the object. That tends to bring everybody down.:-)

VB Events

An event can be simply defined as something that occurs during the applications lifetime. Events allow a class to communicate to other classes and programs. A component *raises* an event to notify the client about the completion of some task. This event can be caught by the client, and the client can respond to the event as it sees fit.

Custom events provide event-handling capabilities to classes and components. The object that generates the event is known as the **event source** and the object that responds to an event is known as the **event sink**. First, we are going to fire a custom event from a VB ActiveX DLL and handle the notification in an MFC client. In other words, we have to build an event sink in an MFC client that responds to events generated by a VB ActiveX component, which acts as the event source. The code in the event sink is executed when the event is fired.

To declare a custom event called `evtTaskDone` in VB, use the **Event** keyword in the **General Declarations** section of a class module like:

Collapse

```
Public Event evtTaskDone()
```

This custom event can then be fire by using the **`RaiseEvent`** keyword like:

Collapse

```
RaiseEvent evtTaskDone
```

With all that in mind, we now roll up our sleeves and dig into some code. First, we are going to build a VB ActiveX DLL that is the source of the event.

Building the event source

Fire up VB and in the New Project dialog, choose **ActiveX DLL** and click **Open**. VB creates a new DLL component project called Project1, having a single class Class1. Go to Project->Properties and set the Project Name as **VBEvents**. In the Project Explorer View, right-click on Class1 and choose to remove it from the project. **Note:** We could also have chosen to use this class, but then we wouldn't have seen the VB Class Builder.

Now right-click again on the Project Explorer View and add a single **Class Module** to the project. In the **Add Class Module** dialog, choose **VB Class Builder** and click on Open.

In Class Builder, go to **File->New->Class** and add a new class called `clsEventSrc` Accept the default values and click on OK. Next, go to **File->New->Event** and add a single event to this class called `evtNotify`. Update all the changes to the project and close the Class Builder window.

Next, click on **Tools->Add Procedure** and add a new procedure to the `clsEventSrc` class called `prcFireEvent` to the class you just created like:

Collapse

```
Public Sub prcFireEvent()
RaiseEvent evtNotify
End Sub
```

The procedure simply fires our event. Save everything and go to **File->Make** to build VBEvents.dll and register the component.

Building the MFC client

Our MFC client, is a plain old Appwizard generated Dialog-based application with additional **Automation support**. As usual, open VC++ 6.0 and create a new MFC Appwizard EXE project called **MFCClient**. Hit Build to build the project, and take a break from all that hard work!

The OLE/COM Object Viewer is a nifty little tool that is shipped along with Visual C++ 6.0. It will help us generate the IDL file for the DLL component. Go to **Tools->OLE/COM Object Viewer** and open this tool. Next, in OLE/COM Object Viewer, click on **File->View Typelib** and navigate to the *VBEvent.dll* file that we have previously built. Ready for some magic? Click on Open and open up **ITypeLib Viewer** Can you can view the IDL file? Whoa! Save the file through **File->Save As**, as **VBEvents.IDL** and close the tool. We have no need for it at present.

Next in our VC++ project, add this IDL file to the project. In FileView, right click on the IDL file and choose **Settings**. In the **MIDL** tab, set the **Output header file name** to VBEvents.h and the **UUID filename** to *VBEvents_i.c*. Also deselect the **MkTyplib compatible** option.

Save everything and in FileView, right-click on the VBEvents.IDL file and choose **Compile**. This will build the typelibrary and generate the necessary files.

Examine the MIDL generated *VBEvents_i.c* file. It contains all the UUID definitions that the client can use to build a sink object. In *VBEvents.h*, notice the dual interface `_clsEventSrc`. The component's `dispinterface __clsEventSrc` is identified by `DIID___clsEvent`. This is the event source for out custom event.

The next step is to add an sink object to connect to the source event. Fortunately, for us, MFC makes the task of building an event sink as easy as 1-2-3 (and well,4-5-6) With a couple of MFC macros, you can delegate much of the intricacy involved behind building a sink to MFC. First, add a new `CCmdTarget` derived class to the project called **MFCSink**. In the ClassWizard choose to select the **Automation** option. This is our sink object with Automation support.

Then import the server's typelib in the client with #import. If you haven't read my previous article, read it here. Otherwise, go right ahead and use code like:

Collapse

```
#import "VBEvents.dll" rename_namespace("MFCClient")
using namespace MFCClient;
```

There's nothing new to this code. While you are there in *stdafx.h*, also #include the *afxctl.h* file

Next, open *MFCSink.cpp* and modify the `INTERFACE_PART` macro so that the second parameter (*IID*) is the IID of the event source, in our case `DIID___clsEventSrc`. Your interface map should look like:

Collapse

```
BEGIN_INTERFACE_MAP(MFCSink, CCmdTarget)
INTERFACE_PART(MFCSink, DIID___clsEventSrc, Dispatch)
END_INTERFACE_MAP()
```

Next, in the `DISPATCH` Map of the MFCSink class add a `DISP_FUNCTION_ID` macro for each of the events defined in the source interface that you want to handle. My `DISPATCH` Map looks like:

Collapse

```
BEGIN_DISPATCH_MAP(MFCSink, CCmdTarget)
        //{{AFX_DISPATCH_MAP(MFCSink)

        DISP_FUNCTION_ID(MFCSink, "evtNotify",1,evtNotify,
        //}}AFX_DISPATCH_MAP

END_DISPATCH_MAP()
```

In Classview, right-click on the `IMFCSink` interface and add a single Method `evtNotify()`. Notice, that as per our DISPATCH Map, this method takes no parameters and returns a void. Our implementation of this method displays a simple `MessageBox` and looks like:

Collapse

```
void MFCSink::evtNotify()
{
// TODO: Add your dispatch handler code here

AfxMessageBox("Event notification handled in MFC client");
}
```

All that remains for us to do is hook up and terminate the connection appropriately in the client code. MFC makes this job very easy with **`AfxConnectionAdvise()`** and correspondingly `AfxConnectionUnadvise()`. If you are not familiar with these functions, now would be a good time to look up their documentation.

Moving on, declare three variables in the dialog class header as :

Collapse

```
_clsEventSrc *m_pSrc;
MFCSink *m_pSink;
DWORD m_dwCookie;
```

The first is a pointer to the interface through which we shall fire the event. The second is a pointer to the sink object. Lastly, the *m_dwCookie* variable is a cookie that stores the number of connections that has been established. We'll need this when we want to disconnect from the event source. In our case, we set this to 1 in the dialog class constructor. Don't forget to #include the *VBEvents_i.c* file. The code to

establish the connection in the dialog's `OnInitDialog()` member function looks like:

Collapse

```
CoInitialize(NULL);  /*Initialize COM system*/
m_pSink=new MFCSink; /*create an instance of the sink obje

/*create source object*/
HRESULT hr=CoCreateInstance(CLSID_clsEventSrc,NULL,CLSCTX_

if(SUCCEEDED(hr))
        LPUNKNOWN m_pUnk=m_pSink->GetIDispatch(FALSE);

if(SUCCEEDED(hr))
{
        /*establish the connection*/

        if (AfxConnectionAdvise(m_pSrc,DIID___clsEventSrc,
                return TRUE;
        else
                return FALSE;
}
else
        return FALSE;
```

As we have setup a connection, we also need to disconnect when the dialog is destroyed. We can do that in the dialog's `OnDestroy()`. In ClassView, right-click the dialog class and Add a Windows message handler to handle `WM_DESTROY` messages. In the handler, add the following code to successfully disconnect.

Collapse

```
LPUNKNOWN m_pUnk=m_pSink->GetIDispatch(FALSE);

AfxConnectionUnadvise(m_pSrc,DIID___clsEventSrc,m_pUnk,FAI
if(m_pSink!=NULL)
{
        delete m_pSink; /*the sink destructor must be publ
        m_pSink=NULL;
        m_pSrc=NULL;
}
```

With everything in place, we now need to fire the event. Simply call:

Collapse

```
m_pSrc->prcFireEvent();
```

anywhere in your code where you want to fire the event.

ATL Client

Building a pure ATL client means a little more typing than the MFC client. But it's a lot easier than creating connectable objects in raw C++. Remember, that the sink has to support `IDispatch`. So that means at a minimum, implementing 7 methods.(3 for `IUnknown` and 4 for `IDispatch`). To our relief, ATL provides the `IDispEventSimpleImpl<>` and `IDispEventImpl<>` template classes that helps us in quickly creating `dispinterface` sink objects. These is a host of information and code available for creating ATL sinks for `dispinterface` based source objects that you might want to lookup. Relevant Microsoft KB Articles Q:181277, Q:181845 and Q:194179

Back to the task at hand, to make our client very efficient, we'll use an `IDispEventSimpleImpl` derived class. First create a new ATL/COM AppWizard generated EXE project called **ATLClient**. To this add a dialog called `ATLClientDlg`. The dialog has two buttons, one to setup the connection and the other to fire the event. Next import te server's typelib with #import as described in the MFC client section above. Moving on to the sink object, the declaration looks like:

Collapse

```
#define IDC_SRCOBJ 1
static _ATL_FUNC_INFO OnEventInfo = {CC_STDCALL, VT_EMPTY,

class CSinkObj : public IDispEventSimpleImpl&ltIDC_SRCOBJ,
{
public:
        HWND m_hWndList;
```

```
        CSinkObj(HWND hWnd = NULL) : m_hWndList(hWnd)
        {
        }

BEGIN_SINK_MAP(CSinkObj)
        //Make sure the Event Handlers have __stdcall cal
        SINK_ENTRY_INFO(IDC_SRCOBJ, __uuidof(__clsEventSrc
END_SINK_MAP()

        // Event handler

        HRESULT __stdcall evtNotify()
        {
                // output string to list box

                TCHAR buf[80];
                wsprintf(buf, "Sink : Notification Event F
                AtlTrace("\n%s",buf);
                return S_OK;
        }
};
```

All I have done is add a sink map to the `IDispEventSimpleImpl`-derived class and then add a sink entry corresponding to each event of a source interface that I would like to handle. The `ATL_FUNC_INFO` structure helps us pass parameters to event handlers. In our event handler however, we do nothing fancy. Just a simple debug message would do.

In the dialog class, add variables:

Collapse

```
private:
        CSinkObj* m_pSink;
        _clsEventSrc *pEvent;
```

The dialog class's OnConnect() looks like:

Collapse

```
LRESULT OnConnect(UINT,WORD,HWND hWndCtrl,BOOL& bHandled)
{
    m_pSink=new CSinkObj(hWndCtrl);

    HRESULT hr=CoCreateInstance(CLSID_clsEventSrc,NULL,CLS
                                __uuidof(_clsEventSrc),(vo
    if(SUCCEEDED(hr))
    {
        m_pSink->DispEventAdvise(pEvent);
    }
    return hr;
}
```

As before, call:

Collapse

```
pEvent->prcFireEvent();
```

when you want to fire the event. Don't forget to use `DispEventUnadvise()` to disconnect when the dialog is destroyed.

That's it! We have built both an MFC and an ATL client that responds to events generated by a VB ActiveX DLL code component. The code and project files were built with Visual C++ 6.0 SP3 under Win95.

I have included another project **VBTimer** consisting of a VB ActiveX DLL and respective ATL client project files. This project does something a little more sophisticated than our first VB DLL, which fires an event without any parameters. The ActiveX DLL implements a VB Timer that fires an event with an single parameter (timer count) after every 1 second interval. This event is caught by the ATL client that displays the timer count in the output window.

References

NIIT Technical Reference
Microsoft KB Articles Q181845,Q181277 and Q194179

License

About the Author

Amit Dey

Member

Amit Dey is a freelance programmer from Bangalore,India. Chiefly programming VC++/MFC, ATL/COM and PocketPC and Palm platforms. Apart from programming and CP, he is a self-taught guitar and keyboard player.

He can be contacted at visualcdev@hotmail.com

Occupation: Web Developer
Location: India

Other popular COM / COM+ articles:

- Introduction to COM - What It Is and How to Use It.
  A tutorial for programmers new to COM that explains how to reuse existing COM components, for example, components in the Windows shell.
- COM in plain C
  How to create/use COM components in plain C, without MFC, ATL, WTL, or any other framework.
- Understanding Classic COM Interoperability With .NET Applications
  Discusses how existing COM components can be used from managed code.
- Getting the most out of IDispatch
  A C++ class that makes it extremely easy to use a COM object, even in console apps
- Introduction to COM Part II - Behind the Scenes of a COM Server
  A tutorial for programmers new to COM that explains the internals of COM servers, and how to write your own interfaces in C++

Article Top Sign Up to vote for this article

FAQ Search

Noise Tolerance Medium Layout Normal Per page 25 Update

Msgs 1 to 25 of 39 (Total in Forum: 39) (Refresh) First Prev Next

Very Big Question	charfeddine_ahmed	3:09 10 Jul '06
Error while compiling my project	mahajan suyog	4:03 15 Jul '05
Create an ActiveX object in a VC++ Application	Anonymous	8:39 13 Apr '05
Unsolvable compilation problem with DIID___clsEventSource	Kza Wah	23:49 9 Jan '05
simply superbbbb!!	Sivakumar R	0:38 1 Jul '04
I have to ask	PFlorin	23:57 9 Jun '04
GetIDispatch Failure MSVC Remote Debugger	nnoydb	6:03 23 Mar '04
creating event dll in mfc shared dll	Xins	18:00 10 Mar '04
Re: creating event dll in mfc shared dll	charfeddine_ahmed	21:57 11 Jul '06
How to pass data?	hewang	15:29 30 Jan '04
VB Client, MFC server?	darwinw	19:01 17 Nov '03
Problem with my ActiveX	hajer	3:13 11 Jul '03
Loosing connections	satya1975	9:37 10 May '03
wow great	satya1975	14:27 8 May '03
Using out-of-process ActiveX EXE	Anonymous	3:01 28 Apr '03
Re: Using out-of-process ActiveX EXE	Amit Dey	11:00 16 May '03
Re: Using out-of-process ActiveX EXE	uu3510251	17:57 22 Apr '04
AfxConnectionAdvise()	guzial	1:03 11 Dec '02
How to connectable objects in raw C++	Jack Modulator	23:40 16 Aug '02
Re: How to connectable objects in raw C++	Amit Dey	11:03 19 Aug '02
What if we use an ActiveX EXE	Anonymous	13:17 30 May '02
Misleading title	VC++ programmer	11:26 20 Dec '01
What happen with VC+MFC ActiveX Controls	Xavi Navarro	22:12 13 Feb '02
Re: What happen with VC+MFC ActiveX Controls	Amit Dey	4:27 14 Feb '02
Re: What happen with VC+MFC ActiveX Controls	Xavi Navarro	2:36 15 Feb '02

Last Visit: 8:41 1 Jul '09 Last Update: 8:41 1 Jul '09 1 2 Next »

General News Question Answer Joke Rant Admin