BUILDING A COM/ATL/APC LOCAL SERVER

Bob Coker
rcoker06@gmail.com
May 28, 2015

OVERVIEW

1. Project Background and Description

Before continuing building the COM/ATL/APC client, build the initial COM/ATL server. The article for this is at: http://c-l-associates.com/Home/Building_a_COM_ATL_Local_Server.

The is the third of a multi-part article on how to do the following:

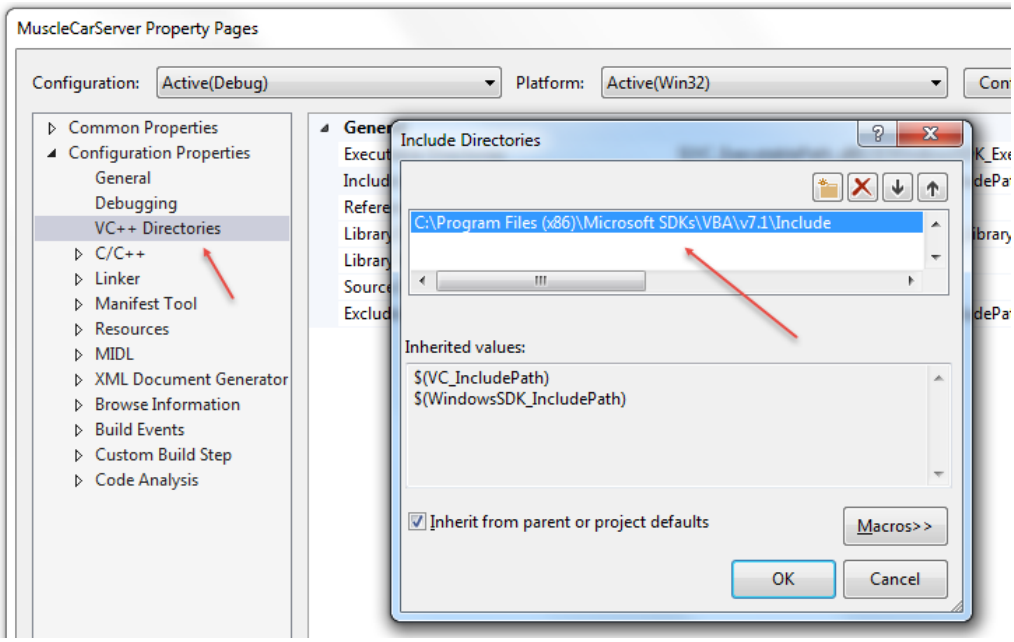
- Build a C++ ATL local server. This server will use user defined types and SAFEARRAYs to pass data.
- Build a C# client that communicates to an ATL local server.
- Pass user defined types to an ATL local server from a C# client.
- Modify the ATL server to communicate with VBA scripts using the APC library.

This article will show how to add code to the COM/ATL server developed in article 1 to allow the COM/ATL server to be scripted by VBA code. When an event is fired from the COM/ATL server this VBA code will get executed. This allows the COM/ATL behavior to be customized.

To build an APC server, you must have the VBA SDK. It appears that Microsoft no longer supports this SDK. So I assume you either have it from past downloads. Otherwise you can contact Summit Software at http://summsoft.com/ to get the SDK. When you obtain this SDK, you will need to install the x86 Vba71.msi file, the Vba71_1033.msi (English) and the vcredist_x86.msi file. See SDK help file for details.

2. Building the COM/ATL/APC Server

1. Install the VBA SDK. I am using version 7.1 for this article, but the examples should work with any VBA SDK back to version 6.
2. Fire up Visual Studio. Make sure you are starting Visual Studio as an administrator.
3. Open the solution MuscleCars COM/ATL server developed previously.
4. Go to project settings, under VC++ directories, add a new include directory for the VBA SDK:



5. The VBA SDK of which the APC is a part works with COM event sinks. There are two parts to a COM event sink: a) the event source which is a part of the COM/ATL server, and the event sink, which in our case will be written in VBA code. It works like this: a) the COM/ATL client calls an event, the COM/ATL server fires the event and the VBA code executes code against that event. This sequence can be difficult to wrap your head around, but I will try in this article to explain how it all happens. From the first article, we discussed how we were going to build an application that allows cars to be rated. So the COM/ATL client built in part 2, will send a request for the COM/ATL server to GetRatings. The COM/ATL server execute its exposed GetRatings method. In that method it will fire the GetRatings event. This is a separate method in the COM/ATL server with the same name. This method will Invoke or fire the event passing in the required parameters. The VBA code will execute its GetRatings method with VBA code that we provide. By this means, rating data will be passed back to the client. First, we need to add another method to the IDispatch interface. Add the method GetRatings as follows:

```
interface IMuscleCarApp : IDispatch
{
        [id(2), helpstring("Get ratings")] HRESULT GetRatings([in, out] CarInfo* carInfo);
};
```

6. In the MuscleCarApp.h file we need to add the declaration for this method:

```
STDMETHOD(GetRatings)(CarInfo* carInfo);
```

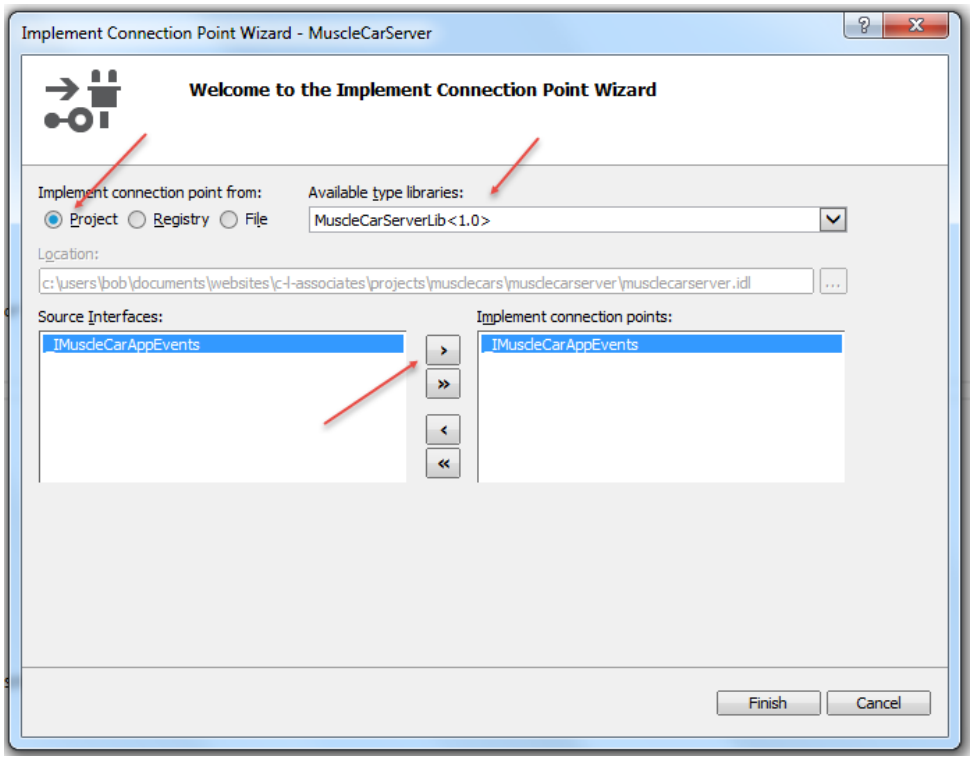
7. In the MuscleCarApp.cpp file we need to add the method body:

```
STDMETHODIMP CMuscleCarApp::GetRatings(CarInfo* carInfo)
{
     return S_OK;
}
```

8. Now go back to the IDL file and add the event declaration. We have made things a little interesting. Many of the illustrations on the web have no parameters or just one parameter for their events. Also, many illustrations when calling an event with multiple parameters use the same data type. I will show later why this is important and show you a major pitfall to avoid when calling events:

```
[id(1), helpstring("Get ratings")] HRESULT GetRatings([in] BSTR make, [in] BSTR model, [in] BSTR
trim, [in, out] double* rating);
```

9. Go to class view. Right mouse click the CMuscleCarApp class. Select Add, Add Connection Point. Select MuscleCarServerLib from the type library drop down. When wizard is complete, click finish:



10. Open up the _IMuscleCarAppEvents file. You will see the implementation for the Fire_GetRatings event:

```
HRESULT Fire_GetRatings(BSTR make, BSTR model, BSTR trim, double * rating)
```

11. Look at the code for Fire_GetRatings. This code was written for you. Notice how the avarParams array is setup. Notice how the parameters are backwards. The last parameter to the function is the first in the array. This is the reason, I chose to use different data types when calling this function to illustrate and emphasize this aspect of calling the Invoke method:

```
HRESULT Fire_GetRatings(BSTR make, BSTR model, BSTR trim, double * rating)
{
        HRESULT hr = S_OK;
        T * pThis = static_cast<T *>(this);
        int cConnections = m_vec.GetSize();

        for (int iConnection = 0; iConnection < cConnections; iConnection++)
        {
                pThis->Lock();
                CComPtr<IUnknown> punkConnection = m_vec.GetAt(iConnection);
                pThis->Unlock();

                IDispatch * pConnection = static_cast<IDispatch *>(punkConnection.p);

                if (pConnection)
                {
                        CComVariant avarParams[4];
                        avarParams[3] = make;
                        avarParams[3].vt = VT_BSTR;
                        avarParams[2] = model;
                        avarParams[2].vt = VT_BSTR;
                        avarParams[1] = trim;
                        avarParams[1].vt = VT_BSTR;
```

```
                        avarParams[0].byref = rating;
                        avarParams[0].vt = VT_R8|VT_BYREF;
                        CComVariant varResult;

                        DISPPARAMS params = { avarParams, NULL, 4, 0 };
                        hr = pConnection->Invoke(1, IID_NULL, LOCALE_USER_DEFAULT,
                        DISPATCH_METHOD, &params, &varResult, NULL, NULL);
                }
        }
        return hr;
}
```

12. Now go back to the GetRatings method body in the MuscleCarApp.cpp file. We first get the arrays that we need to process the ratings. Next we get the array bound, assuming all arrays are the same size. We loop through the arrays, and get the make, model and trim for each car. Then we fire our event. After event is fired, we put the value for the rating back into our rating array, which will eventually be used by our client application. So the event sequence is rather simple once you get used to it. The client calls a regular method, not the event itself. The regular method calls or fires the event. Later the VBA code we hook up to the event will be Invoked by our event (remember the Invoke call in the event fire method). Finally the event returns and our client data is updated with the results:

```
STDMETHODIMP CMuscleCarApp::GetRatings(CarInfo* carInfo)
{
        long lMake = -1;
        long uMake = -1;

        pMakeArray = carInfo->Make.parray;
        pModelArray = carInfo->Model.parray;
        pTrimArray = carInfo->Trim.parray;
        pRatingArray = carInfo->Rating.parray;

        ::SafeArrayGetUBound(pMakeArray, 1, &uMake);
        ::SafeArrayGetLBound(pMakeArray, 1, &lMake);

        BSTR bstrMake;
        BSTR bstrModel;
        BSTR bstrTrim;
        double dRating;

        for (long i = lMake; i <= uMake; i++)
        {
                ::SafeArrayGetElement(pMakeArray, &i, (void*)&bstrMake);
                ::SafeArrayGetElement(pModelArray, &i, (void*)&bstrModel);
                ::SafeArrayGetElement(pTrimArray, &i, (void*)&bstrTrim);

                Fire_GetRatings(bstrMake, bstrModel, bstrTrim, &dRating);

                ::SafeArrayPutElement(pRatingArray, &i, (void*)&dRating);
        }

        return S_OK;
}
```

13. No we need to start setting up the APC code. Open the stdafx.h file and edit so that it looks like the code below. Make sure you can compile when done:

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently,
// but are changed infrequently

#pragma once

#ifndef STRICT
#define STRICT
#endif
```

```
#include "targetver.h"

#define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows headers
// Windows Header Files:
#include <windows.h>

#define _ATL_APARTMENT_THREADED

//#define _ATL_NO_AUTOMATIC_NAMESPACE

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS       // some CString constructors will be explicit

#define ATL_NO_ASSERT_ON_DESTROY_NONEXISTENT_WINDOW

#include "resource.h"
#include <atlbase.h>
#include <atlcom.h>
#include <atlctl.h>
#include <atlsafe.h>

#define APC_IMPORT_MIDL

#include <apcCpp.h>

using namespace MSAPC;
using namespace VBIDE;
using namespace Office;

#include "evalkey.h"

#include <assert.h>
```

14. Now open up the stdafx.cpp file and add code as follows. Make sure the project compiles without errors:

```
// stdafx.cpp : source file that includes just the standard includes
// MuscleCarServer.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

#include <apcguids.h>
#ifdef _ATL_STATIC_REGISTRY
#include <statreg.h>
#include <statreg.cpp>
#endif

#include <apcimpl.cpp>
```

15. Now open up the MuscleCarApp.h file and add code the following code to the top of the file:

```
using namespace MSAPC;
```

16. In the MuscleCarApp.h file modify the CMuscleCarApp class to derive from CApcHost. Also add the private variables below:

```
class ATL_NO_VTABLE CMuscleCarApp :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CMuscleCarApp, &CLSID_MuscleCarApp>,
        public IConnectionPointContainerImpl<CMuscleCarApp>,
        public CProxy_IMuscleCarAppEvents<CMuscleCarApp>,
        public IDispatchImpl<IMuscleCarApp, &IID_IMuscleCarApp, &LIBID_MuscleCarServerLib,
        /*wMajor =*/ 1, /*wMinor =*/ 0>,
        public CApcHost<CMuscleCarApp>

....

private:
        bool m_bWinHelp;
```

```
        int m_lcidVBA;
```

17. In the MuscleCarApp.h file add the public helper GetDispatch as a public member below. Also add a public member for m_apcInit:

```
// IUnknown methods
IDispatch *GetDispatch(void);

public:
        BOOL m_apcInit = FALSE;
```

18. In the MuscleCarApp.cpp file add a global variable g_pIDE to the top of the file. APC has an IDE to help edit VBA code. We will show how this works later:

```
// Global Variables
IApcIde* g_pIDE = NULL;
```

19. In the MuscleCarApp.h file initialize the variables m_bWinHelp and m_lcidVBA in the constructor. Also, add a public declaration for the destructor:

```
CMuscleCarApp()
{
        m_bWinHelp = false;             // HTML Help
        m_lcidVBA = 1033;               // English
}
~CMuscleCarApp();
```

20. Go back to the MuscleCarApp.cpp file and add a destructor. Compile the application:

```
CMuscleCarApp::~CMuscleCarApp()
{
        // End VBA Session.
        ApcHost.Destroy();
}
```

21. Go back to the MuscleCarApp.idl file and add a two methods for APC use. One is for initializing APC from client side. The other is for showing the APC IDE:

```
interface IMuscleCarApp : IDispatch{
...
[id(3), helpstring("Initialize APC")] HRESULT InitializeAPC();
[id(4), helpstring("Show IDE")] HRESULT ShowIDE();
```

22. Go back to the MuscleCarApp.h file and add a two methods:

```
public:
        STDMETHOD(InitializeAPC)();
        STDMETHOD(ShowIDE)();
```

23. Now the implementation of these the GetDispatch method in the MuscleCarApp.cpp file:

```
IDispatch* CMuscleCarApp::GetDispatch(void)
{
        return (IMuscleCarApp *)this;
};
```

24. Now the implementation of the InitializeAPC method in the MuscleCarApp.cpp file. This code will initialize the APC host:

```
STDMETHODIMP CMuscleCarApp::InitializeAPC()
{
        HRESULT hr;

        // Check for APC activation
        if (m_apcInit)
        {
```

```
                return S_OK;
        }

        // Create APC host
        BSTR bstrLicKey = SysAllocString(wszEvalLicKey);
        IDispatch* iDisp = GetDispatch();
        hr = ApcHost.Create(NULL, L"MuscleCar APC Server", iDisp, bstrLicKey, m_lcidVBA);
        SysFreeString(bstrLicKey);
        if (hr)
        {
                return hr;
        }

        m_apcInit = TRUE;
        return S_OK;
}
```

25. Next the implementation of the ShowIDE method in the MuscleCarApp.cpp file. Compile the application:

```
STDMETHODIMP CMuscleCarApp::ShowIDE()
{
        HRESULT hr = S_OK;
        VARIANT_BOOL vbVisible = VARIANT_TRUE;

        //      Check for prior initialization
        if (g_pIDE)
        {
                hr = g_pIDE->APC_PUT(Visible)(vbVisible);
                return hr;
        }

        //      Show/initialize IDE
        if (SUCCEEDED(hr = this->ApcHost->APC_GET(Ide)(&g_pIDE)))
        {
                hr = g_pIDE->APC_PUT(Visible)(vbVisible);
                g_pIDE->Release();
        }
        return hr;
}
```

26. The APC has a concept of a project and a project item. We will need to add support for these in the CMuscleCarApp header file. You will see how these work up ahead:

```
class ATL_NO_VTABLE CMuscleCarApp :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CMuscleCarApp, &CLSID_MuscleCarApp>,
        public IConnectionPointContainerImpl<CMuscleCarApp>,
        public CProxy_IMuscleCarAppEvents<CMuscleCarApp>,
        public IDispatchImpl<IMuscleCarApp, &IID_IMuscleCarApp, &LIBID_MuscleCarServerLib,
        /*wMajor =*/ 1, /*wMinor =*/ 0>,
        public CApcHost<CMuscleCarApp>,
        public CApcProject<CMuscleCarApp>,
        public CApcProjectItem<CMuscleCarApp>
```

27. Next we are going to add support for a new project. A new APC project is a blank project with no VBA code. Later we will have a LoadProject, SaveProject and a CloseProject method. The use cases are as follows:

    a. Create a new project
        i. Initialize APC
        ii. New project
        iii. Show IDE
        iv. Input VBA code
        v. Save project
        vi. Close project

    b. Open a previous project:
        i. Initialize APC

ii. Load project
iii. Show IDE
iv. Edit VBA code
v. Save project
vi. Close project

28. Add a public method to the MuscleCarApp.h file for the NewProject. Also add the following private variables to the file:

```
private:
        IStorage* m_pStg;
        bool m_bProjectCreated;

...
STDMETHOD(NewProject)();
```

29. Initialize these variables in the constructor of the MuscleCarApp.h file:

```
CMuscleCarApp()
{
        m_bWinHelp = false;             // HTML Help
        m_lcidVBA = 1033;               // English

        m_bProjectCreated = false;
        m_pStg = NULL;
}
```

30. Go back to the MuscleCarApp.idl file and add a method for NewProject:

```
interface IMuscleCarApp : IDispatch{
...
[id(5), helpstring("New project")] HRESULT NewProject();
```

31. Implement the NewProject method to the MuscleCarApp.cpp file. Notice here we create a storage file for our APC project which will allow for the saving of the eventual VBA code:

```
STDMETHODIMP CMuscleCarApp::NewProject()
{
        HRESULT hr = S_OK;

        //      Check that project has already been created
        if (m_bProjectCreated)
        {
                return hr;
        }

        if (SUCCEEDED(hr = StgCreateDocfile(NULL, STGM_TRANSACTED | STGM_READWRITE |
        STGM_SHARE_EXCLUSIVE | STGM_CREATE | STGM_DELETEONRELEASE,
        0, &m_pStg)))
        {
                if (SUCCEEDED(hr = ApcProject.Create(this->ApcHost, axProjectNormal,
                L"MuscleCarProject")) && SUCCEEDED(hr = ApcProject.InitNew(m_pStg)))
                {
                        if (ApcProjectItem)
                        {
                                ApcProjectItem.Detach();
                        }
                        hr = ApcProjectItem.Define(ApcProject, GetDispatch(),
                        axTypeHostProjectItem, L"MuscleCars", NULL);
                        m_bProjectCreated = true;
                }
        }

        if (FAILED(hr))
        {
                return hr;
        }
        return hr;
```

```
}
```

32. Go back to the MuscleCarApp.idl file and add a method for LoadProject:

```
interface IMuscleCarApp : IDispatch{
...
[id(6), helpstring("Load project")] HRESULT LoadProject(BSTR fileName);
```

33. Add a public method to the MuscleCarApp.h file for LoadProject:

```
STDMETHOD(LoadProject)(BSTR fileName);
```

34. Implement the LoadProject method to the MuscleCarApp.cpp file:

```
HRESULT CMuscleCarApp::LoadProject(BSTR fileName)
{
	IStorage* pstgRoot, *pstgData;
	IStream* pstmData;
	HRESULT hr = S_OK;

	if (FAILED(hr = StgOpenStorage(fileName, NULL, STGM_TRANSACTED | STGM_READWRITE/* |
	STGM_SHARE_EXCLUSIVE*/, NULL, 0, &pstgRoot)))
	{
		return hr;
	}

	if (SUCCEEDED(hr = pstgRoot->OpenStorage(L"ProjectStg", NULL, STGM_DIRECT | STGM_READ |
	STGM_SHARE_EXCLUSIVE, NULL, 0, &pstgData)))
	{
		if (SUCCEEDED(hr = pstgData->OpenStream(L"ProjectData", NULL, STGM_DIRECT |
		STGM_READ | STGM_SHARE_EXCLUSIVE, 0, &pstmData)))
		{
			ULONG read = 0;
			DWORD dwID;
			if (FAILED(hr = pstmData->Read(&dwID, sizeof dwID, &read)))
				return hr;
			ApcProjectItem.Register(ApcProject, GetDispatch(), dwID);
			pstmData->Release();
		}
		pstgData->Release();
	}
	else
	{
		return hr;
	}

	if (SUCCEEDED(hr = ApcProject.Open(this->ApcHost, axProjectNormal)))
	{
		if (SUCCEEDED(hr = ApcProject.Load(pstgRoot)))
		{
			if (SUCCEEDED(hr = ApcProject.FinishLoading()))
			{
				m_pStg = pstgRoot;
			}
		}
	}

	if (FAILED(hr))
	{
		return hr;
	}

	return hr;
}
```

35. Go back to the MuscleCarApp.idl file and add a method for SaveProject:

```
interface IMuscleCarApp : IDispatch{
```

```
...
[id(7), helpstring("Save project")] HRESULT SaveProject(BSTR fileName);
```

36. Add another public method to the MuscleCarApp.h file for SaveProject:

```
STDMETHOD(SaveProject)(BSTR fileName);
```

37. Implement the SaveProject method to the MuscleCarApp.cpp file:

```
HRESULT CMuscleCarApp::SaveProject(BSTR fileName)
{
	HRESULT hr = S_OK;
	IStorage* pstgRoot, *pstgData;
	IStream* pstmData;

	if (!m_pStg)
	{
		NewProject();
	}

	pstgRoot = m_pStg;

	hr = StgCreateDocfile(fileName, STGM_TRANSACTED | STGM_READWRITE | STGM_SHARE_EXCLUSIVE |
	STGM_CREATE, 0, &pstgRoot);
	if (FAILED(hr))
	{
		return hr;
	}

	if (SUCCEEDED(hr = pstgRoot->CreateStorage(L"ProjectStg", STGM_TRANSACTED | STGM_READWRITE
	| STGM_SHARE_EXCLUSIVE | STGM_CREATE, 0, 0, &pstgData)))
	{
		if (SUCCEEDED(hr = pstgData->CreateStream(L"ProjectData", STGM_DIRECT |
		STGM_READWRITE | STGM_SHARE_EXCLUSIVE | STGM_CREATE, 0, 0, &pstmData)))
		{
			ULONG writ = 0;
			DWORD dwID = ApcProjectItem.ID();
			if (FAILED(hr = pstmData->Write(&dwID, sizeof dwID, &writ)))
				return hr;
			pstmData->Release();
		}
		if (SUCCEEDED(hr))
			pstgData->Commit(STGC_DEFAULT);
		else
		{
			pstgData->Revert();
			pstgData->Release();
			pstgRoot->Revert();
			goto Error;;
		}
		pstgData->Release();
	}

	if (FAILED(hr)) goto Error;

	if (SUCCEEDED(hr = ApcProject.Save(pstgRoot, false)))
	{
		if (SUCCEEDED(hr = ApcProject.SaveCompleted(pstgRoot)))
		{
			pstgRoot->Commit(STGC_DEFAULT);
		}
		else
		{
			pstgRoot->Revert();
		}
	}
	else
	{
```

```
                pstgRoot->Revert();
        }

        if (FAILED(hr)) goto Error;

        if (pstgRoot != m_pStg)
        {
                m_pStg->Release();
                m_pStg = pstgRoot;
        }

        ApcProject->APC_PUT(DisplayName)(fileName);
        SysFreeString(fileName);

Error:
        if (FAILED(hr))
        {
                this->ApcHost->APC_RAW(BeginModalDialog)();
                this ->ApcHost->APC_RAW(ShowError)(hr);
                this ->ApcHost->APC_RAW(EndModalDialog)();
        }

        return hr;
}
```

38. Go back to the MuscleCarApp.idl file and add a method for CloseProject:

```
interface IMuscleCarApp : IDispatch{
...
[id(8), helpstring("Close project")] HRESULT CloseProject();
```

39. Add another public method to the MuscleCarApp.h file for CloseProject:

```
STDMETHOD(CloseProject)();
```

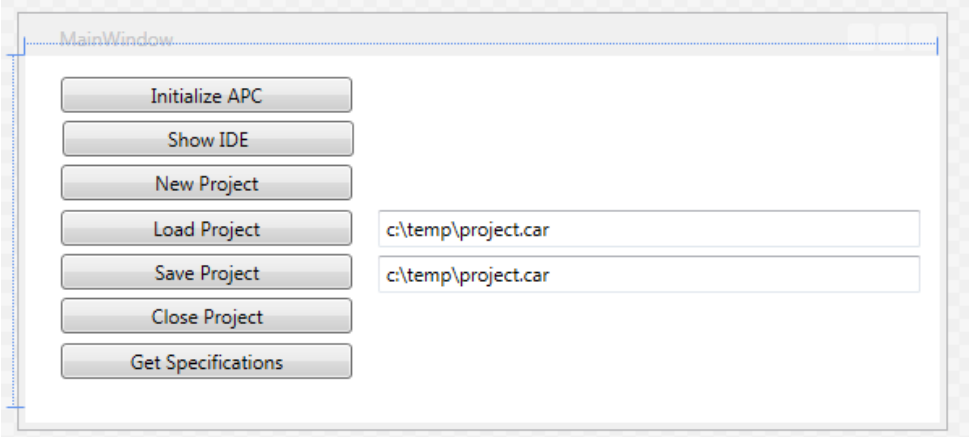
40. Implement the CloseProject method to the MuscleCarApp.cpp file:

```
HRESULT CMuscleCarApp::CloseProject()
{
        HRESULT hr = S_OK;

        ApcProject.Close();
        this->m_apcInit = FALSE;
        this->ApcHost.Destroy();

        if (m_pStg)
        {
                m_pStg->Release();
                m_pStg = NULL;
        }

        m_bProjectCreated = false;

        return hr;
}
```

41. Compile the project. We are done for now with the server. Next we will implement code in the client to exercise the methods we created in the server.

3. Editing the COM/ATL/APC Client

1. Now go back to the client application we created in part two of this series.
2. First we will add 6 new buttons to test our APC/ATL server. Also, we will add two text boxes. In the two textboxes add a path to the location of the CAR file. The CAR file will be the saved project file for your VBA code file:



3. Your XAML should look like the following after you add click handlers for all your buttons:

```
<Window x:Class="MuscleCarClient.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="222" Width="573">
        <Grid Margin="0,0,2,10">
                <Button Content="Initialize APC" Name="BtnInitializeApc"
                HorizontalAlignment="Left" Margin="22,13,0,0" VerticalAlignment="Top" Width="180"
                Click="BtnInitializeApc_Click"/>
                <Button Content="Show IDE" x:Name="BtnShowIde" HorizontalAlignment="Left"
                Margin="23,40,0,0" VerticalAlignment="Top" Width="180" Click="BtnShowIde_Click"/>
                <Button Content="New Project" x:Name="BtnNewProject" HorizontalAlignment="Left"
                Margin="22,67,0,0" VerticalAlignment="Top" Width="180"
                Click="BtnNewProject_Click"/>
                <Button Content="Load Project" x:Name="BtnLoadProject" HorizontalAlignment="Left"
                Margin="22,95,0,0" VerticalAlignment="Top" Width="180"
                Click="BtnLoadProject_Click"/>
                <TextBox x:Name="FileName1" Height="23" Margin="218,95,10,0" TextWrapping="Wrap"
                Text="c:\temp\project.car" VerticalAlignment="Top"/>
                <Button Content="Save Project" x:Name="BtnSaveProject" HorizontalAlignment="Left"
                Margin="22,122,0,0" VerticalAlignment="Top" Width="180"
                Click="BtnSaveProject_Click"/>
                <TextBox x:Name="FileName2" Height="23" Margin="218,123,10,0" TextWrapping="Wrap"
                Text="c:\temp\project.car" VerticalAlignment="Top"/>
                <Button Content="Close Project" x:Name="BtnCloseProject"
                HorizontalAlignment="Left" Margin="22,149,0,0" VerticalAlignment="Top"
                Width="180" Click="BtnCloseProject_Click"/>
                <Button Content="Get Specifications" x:Name="BtnGetSpecifications"
                HorizontalAlignment="Left" Margin="22,177,0,0" VerticalAlignment="Top"
                Width="180" Click="BtnGetSpecifications_Click"/>
        </Grid>
</Window>
```

4. Implement the click handler for the BtnInitializeApc button:

```
private void BtnInitializeApc_Click( object sender, RoutedEventArgs e )
{
        //      Initialize the APC
        _muscleCarApp.InitializeAPC();
}
```

5. Implement the click handler for the BtnShowIde button:

```
private void BtnShowIde_Click( object sender, RoutedEventArgs e )
{
        //      Show IDE
        _muscleCarApp.ShowIDE();
}
```

6. Implement the click handler for the BtnNewProject button:

```
private void BtnNewProject_Click( object sender, RoutedEventArgs e )
{
        _muscleCarApp.NewProject();
}
```

7. Implement the click handler for the BtnLoadProject button:

```
private void BtnLoadProject_Click( object sender, RoutedEventArgs e )
{
        _muscleCarApp.LoadProject( FileName1.Text );
}
```

8. Implement the click handler for the BtnSaveProject button:

```
private void BtnSaveProject_Click( object sender, RoutedEventArgs e )
{
        _muscleCarApp.SaveProject( FileName2.Text );
}
```

9. Finally, implement the click handler for the BtnCloseProject button:

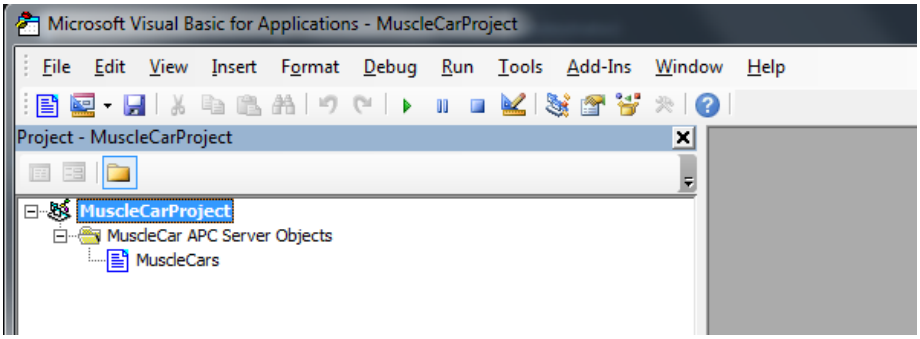
```
private void BtnCloseProject_Click( object sender, RoutedEventArgs e )
{
        _muscleCarApp.CloseProject();
}
```

10. Edit the code GetSpecifications click handler to add in the GetRatings method and the rating array:

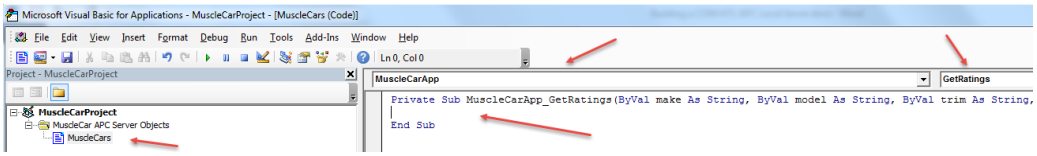
```
private void BtnGetSpecifications_Click( object sender, RoutedEventArgs e )
{
        var carInfo = new CarInfo();

        var make = new string[] { "Chevrolet", "Dodge", "Ford" };
        var model = new string[] { "Camaro", "Challenger", "Mustang" };
        var trim = new string[] { "SS", "R/T", "Shelby GT350" };
        var engine = new string[3];
        var engineLiters = new double[3];
        var horsePower = new double[3];
        var rating = new double[3];

        carInfo.make = make;
        carInfo.model = model;
        carInfo.trim = trim;
        carInfo.Engine = engine;
        carInfo.EngineLiters = engineLiters;
        carInfo.Horsepower = horsePower;
        carInfo.rating = rating;

        _muscleCarApp.GetSpecifications( ref carInfo );
        _muscleCarApp.GetRatings( ref carInfo );

        var aMake = (Array)carInfo.make;
        var aModel = (Array)carInfo.model;
        var aTrim = (Array)carInfo.trim;
        var aEngine = (Array)carInfo.Engine;
        var aEngineLiters = (Array)carInfo.EngineLiters;
        var aHorsePower = (Array)carInfo.Horsepower;
        var aRatings = (Array)carInfo.rating;

        for( var i = 0; i < 3; i++ )
        {
                var msg = string.Format( "Make: {0}\nModel: {1}\nTrim: {2}\nEngine: {3}\nLiters:
                {4}\nHorsePower: {5}\nRating: {6}", aMake.GetValue( i ), aModel.GetValue( i ),
```

```
                aTrim.GetValue( i ), aEngine.GetValue( i ), aEngineLiters.GetValue( i ),
                aHorsePower.GetValue( i ), aRatings.GetValue(i) );
                MessageBox.Show( msg );
                }
        }
```

11. Now run the application. Click the Initialize APC button. Click the New Project button. Click Show IDE button:



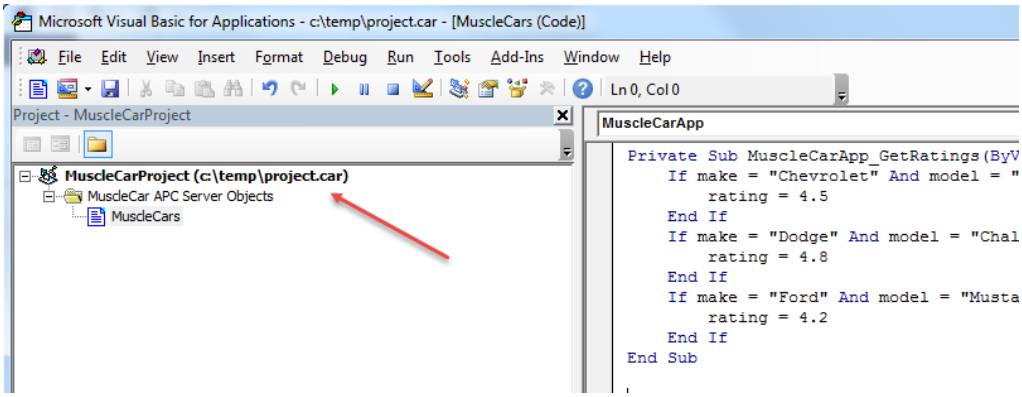
12. Now click the MuscleCars item in the left tree, select the MuscleCapApp in the left dropdown and the GetRatings event handler in the right tree:



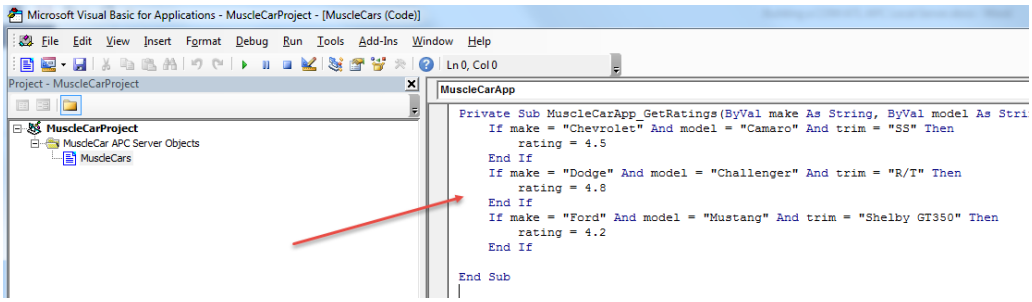
13. Now edit the code inside the handler to the following:

```
Private Sub MuscleCarApp_GetRatings(ByVal make As String, ByVal model As String, ByVal
trim As String, rating As Double)
    If make = "Chevrolet" And model = "Camaro" And trim = "SS" Then
        rating = 4.5
    End If
    If make = "Dodge" And model = "Challenger" And trim = "R/T" Then
        rating = 4.8
    End If
    If make = "Ford" And model = "Mustang" And trim = "Shelby GT350" Then
        rating = 4.2
    End If
End Sub
```

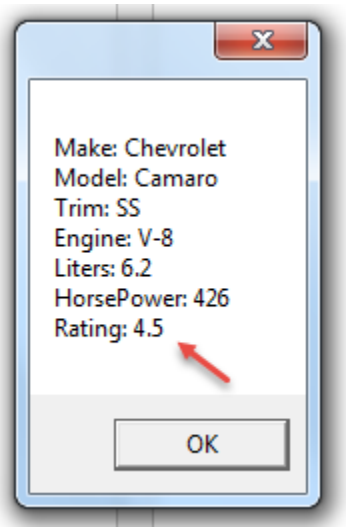
14. Now click the save project button. You should see the IDE display the file name:



15. Now click the close project button. The IDE should close. Close the MainWindow dialog of the MuscleCar client. If at any point, you get a server error concerning sharing violation when loading or saving project, you might need to kill the MuscleCar server in Task Manager. I have not addressed this in the code.
16. Now run the application. Click the Initialize APC button. Click the Load Project button. Click Show IDE button. You should see your code in the IDE:



17. Now click the Get Specifications button. You should see the ratings displayed:



4. Conclusion

1. We have shown a powerful technology in this article. A software vendor can have a product that is sold to customers (Muscle Car server). Software developers can use the Muscle Car server in our case and integrate this into their custom software (Muscle Car Client). The client software sold to 3rd parties can be customized via the VBA IDE. Instead of business logic being embedded in the client, it can be customized

by the 3rd party. This is a powerful technology that could make your application stand out from your competition. This technology outlined in these articles is not very well known and it is certainly considered a legacy solution. However, what other way would you do this ? You could build your own scripting engine – that's a lot of work and it would not be VBA. VBA is used for all macro creation for Microsoft products. It also can be used to script web pages as in classic ASP. So a user that knows VBA will be comfortable creating macros against the events you expose in your ATL/APC server. Other articles in this series can be found at: http://c-l-associates.com/Home/Com.

2.