Visual C++ and DHTML

Program examples compiled using Visual C++ 6.0 compiler on Windows XP Pro machine with Service Pack 2 and some Figure screen snapshots have been taken on Windows 2000 server. The Internet Information Services version is IIS 4.x/5.x/6.x on Windows 2000 Server SP 4 and Windows XP Pro SP 2. The Internet Explorer is 6.x. Topics and sub topics for this tutorial are listed below. A complete information about IIS installation, configuration and testing a Web site is dumped HERE and how to setup FTP server also included HERE. Both Web and FTP servers were done on Windows 2000 Server SP4. Don't forget to read Tenouk's small disclaimer.

Index:

Introduction

Visual C++ 6.0 supports DHTML through both MFC and ATL. Both MFC and ATL give you complete access to the DHTML object model. Unfortunately, access to the object model from languages like C++ is done through OLE Automation (`IDispatch`) and in many cases isn't as cut-and-dried as some of the scripts we looked at earlier. The DHTML object model is exposed to C++ developers through a set of COM objects with the prefix `IHTML` (`IHTMLDocument`, `IHTMLWindow`, `IHTMLElement`, `IHTMLBodyElement`, and so on). In C++, once you obtain the document interface, you can use any of the `IHTMLDocument2` interface methods to obtain or to modify the document's properties.
You can access the all collection by calling the `IHTMLDocument2::get_all` method. This method returns an `IHTMLElementCollection` collection interface that contains all the elements in the document. You can then iterate through the collection using the `IHTMLElementCollection::item` method (similar to the parentheses in the script above). The `IHTMLElementCollection::item` method supplies you with an `IDispatch` pointer that you can call `QueryInterface()` on, requesting the `IID_IHTMLElement` interface. This call to `QueryInterface()` will give you an `IHTMLElement` interface pointer that you can use to get or set information for the HTML element.
Most elements also provide a specific interface for working with that particular element type. These element-specific interface names take the format of `IHTMLXXXXElement`, where `XXXX` is the name of the element (`IHTMLBodyElement`, for example). You must call `QueryInterface()` on the `IHTMLElement` object to request the element-specific interface you need. This might sound confusing (because it can be!). But don't worry - the MFC and ATL sections in this module contain plenty of samples that demonstrate how it all ties together. You'll be writing DHTML code in no time.

MFC and DHTML

MFC's support for DHTML starts with a new `CView` derivative, `CHtmlView`. `CHtmlView` allows you to embed an HTML view inside frame windows or splitter windows, where with some DHTML work it can act as a dynamic form. Example MYEX36A demonstrates how to use the new `CHtmlView` class in a vanilla MDI application.
Follow these steps to create the MYEX36A example:

Run AppWizard and create MYEX36A. Choose **New** from Visual C++'s **File** menu. Then click the **Projects** tab, and select **MFC AppWizard (exe)**. Accept all defaults, except in Step 6 choose `CHtmlView` as the **Base Class**, as shown here.

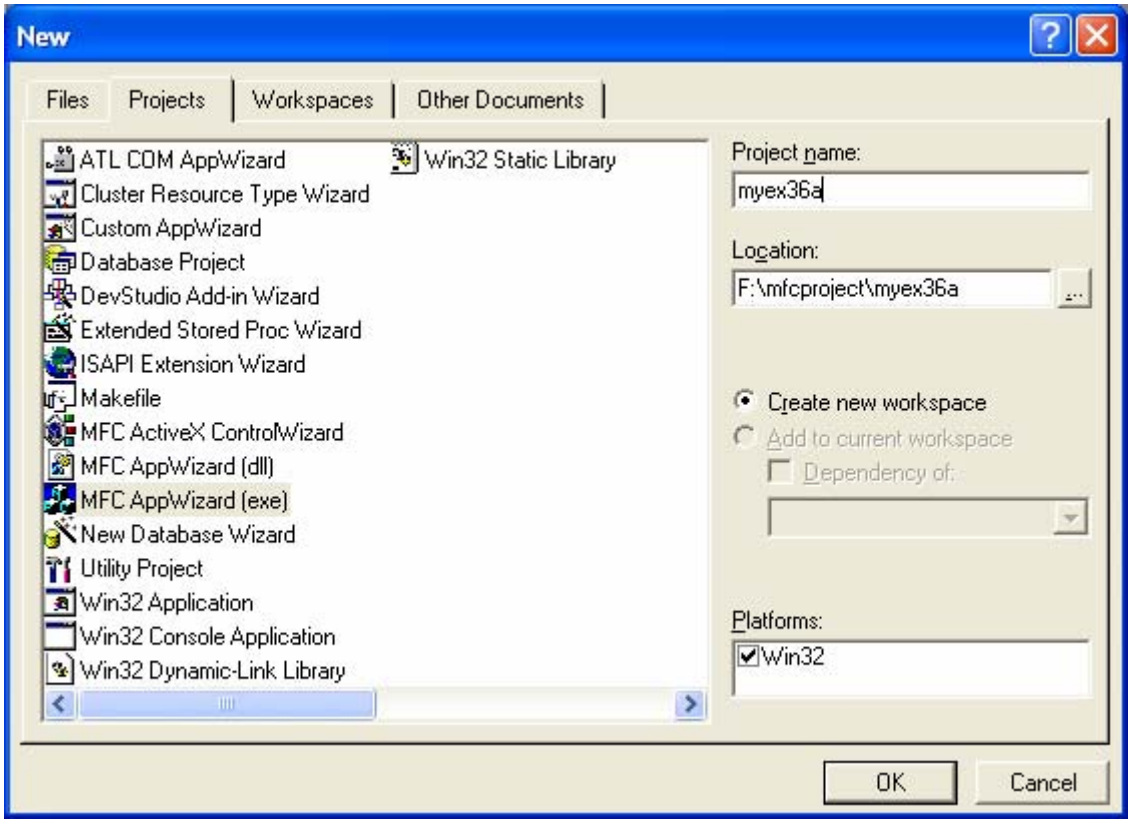

Figure 1: MYEX36A – Visual C++ MFC AppWizard new project dialog.

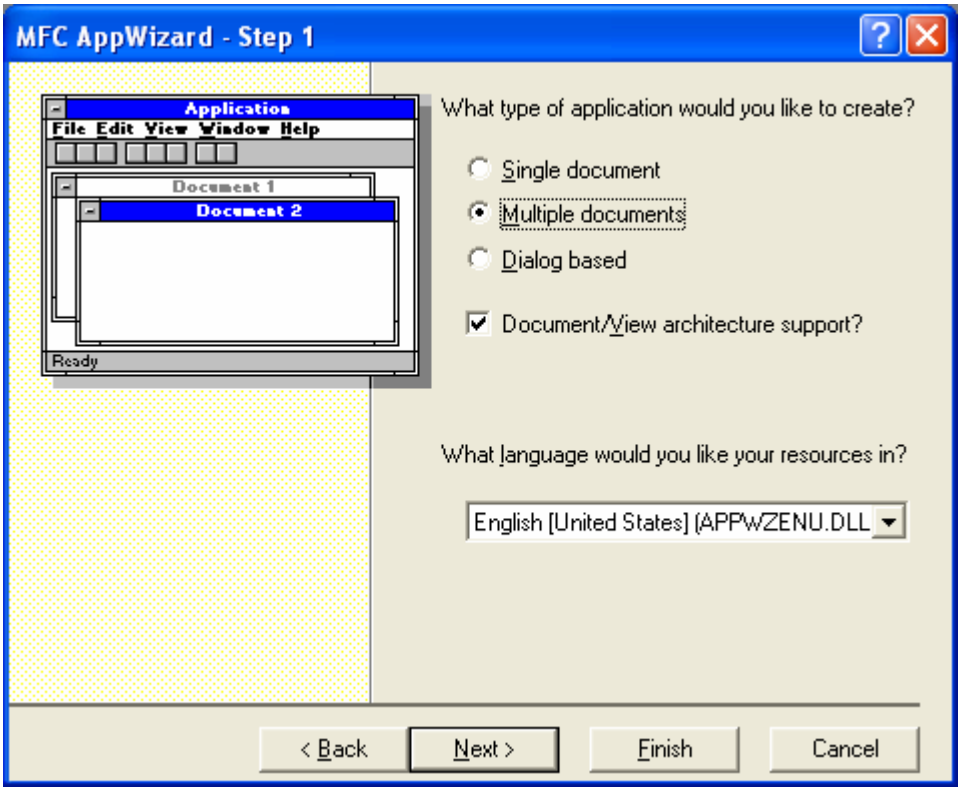

Figure 2: MYEX36A – AppWizard step 1 of 6.

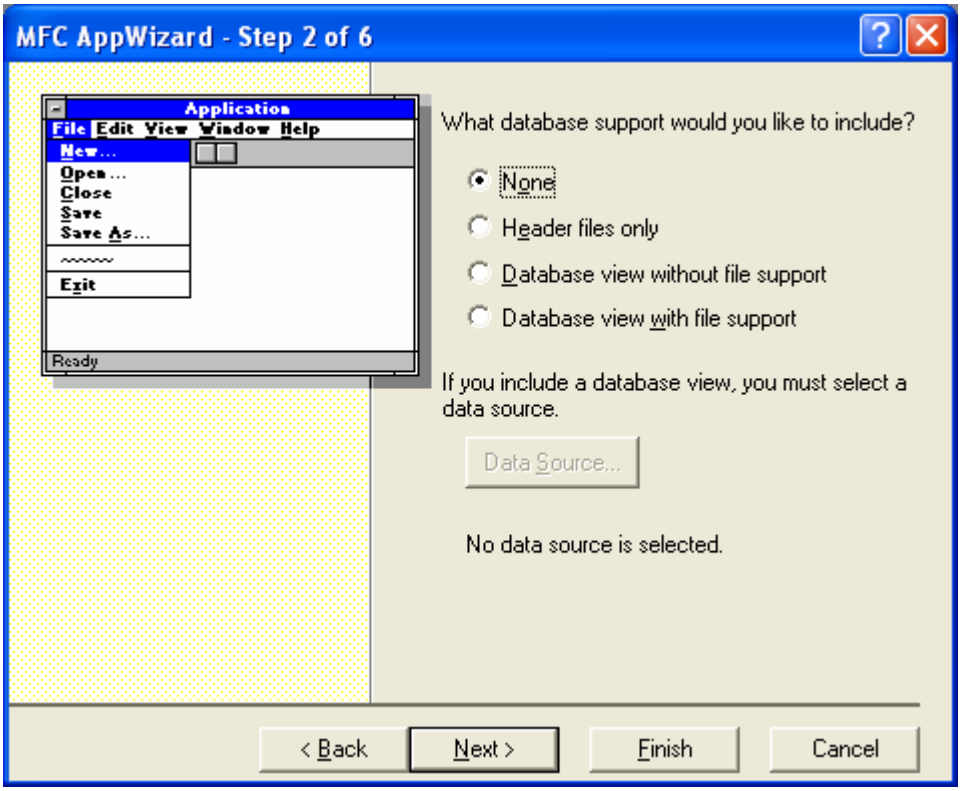

Figure 3: MYEX36A – AppWizard step 2 of 6.

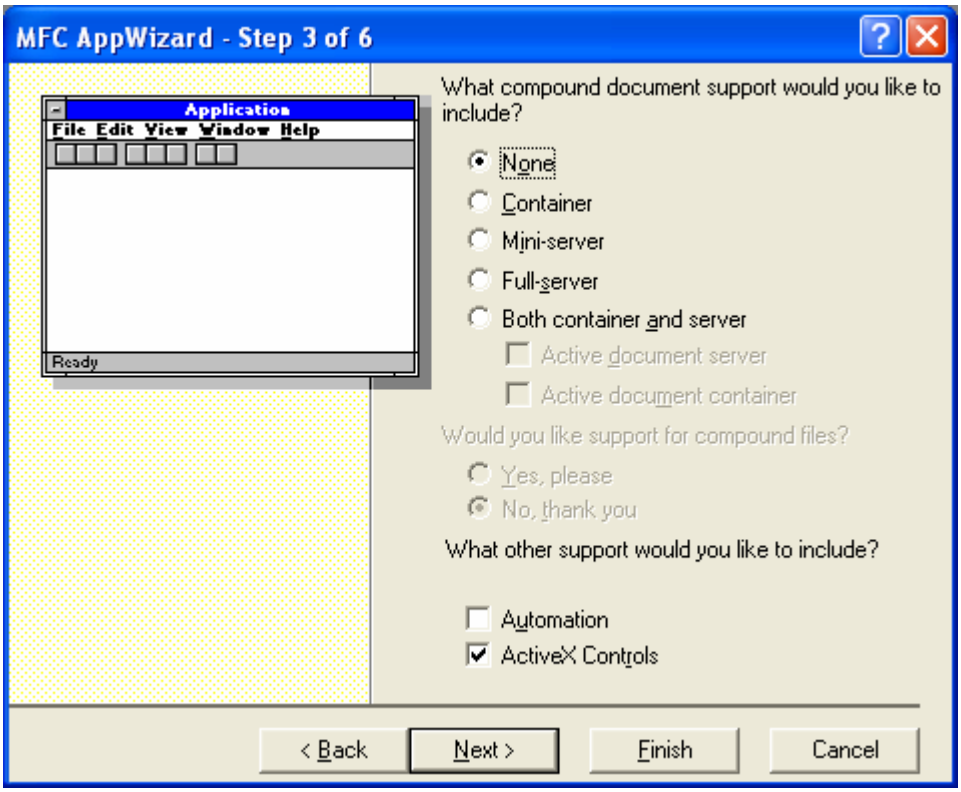

Figure 4: MYEX36A – AppWizard step 3 of 6.

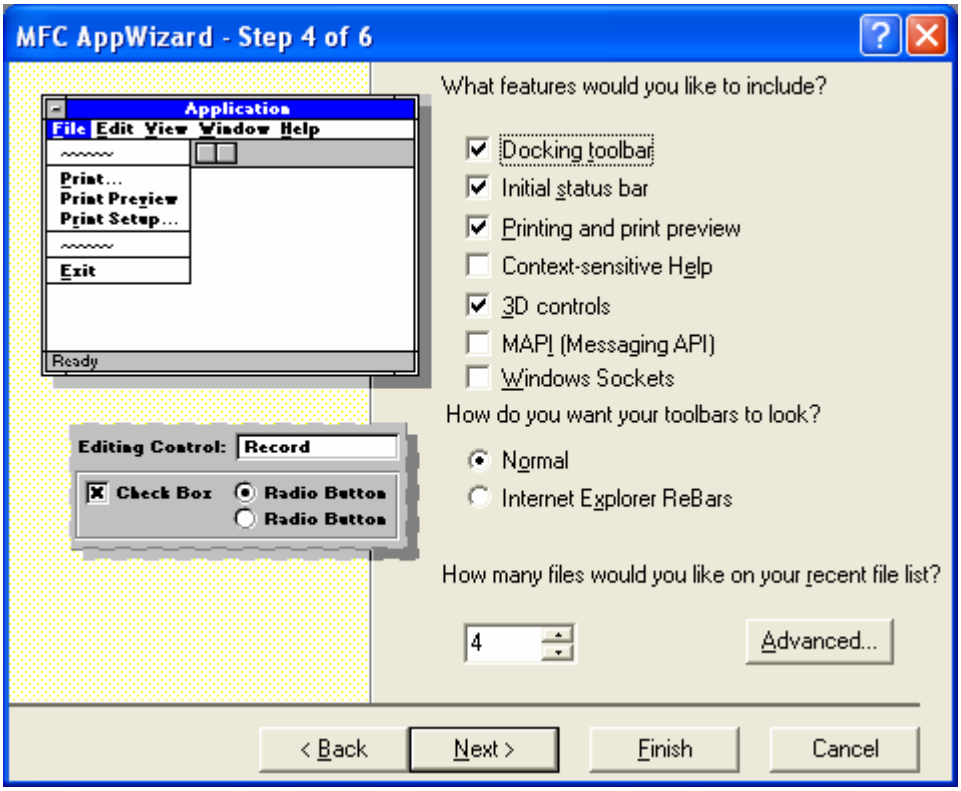

Figure 5: MYEX36A – AppWizard step 4 of 6.

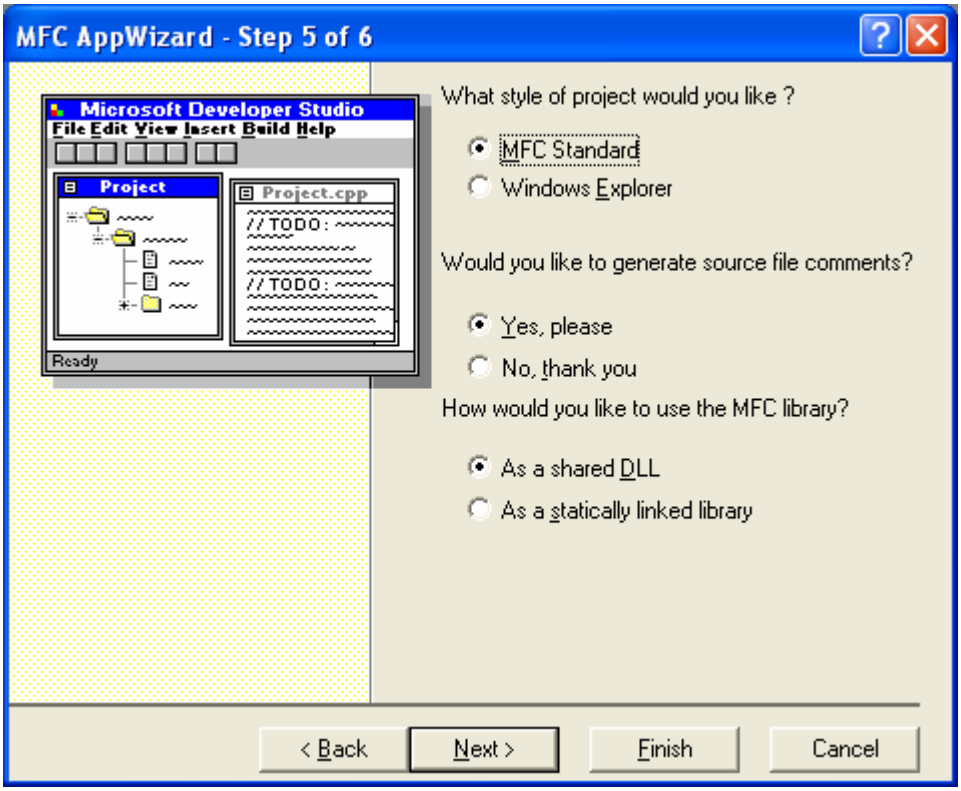

Figure 6: MYEX36A – AppWizard step 5 of 6.

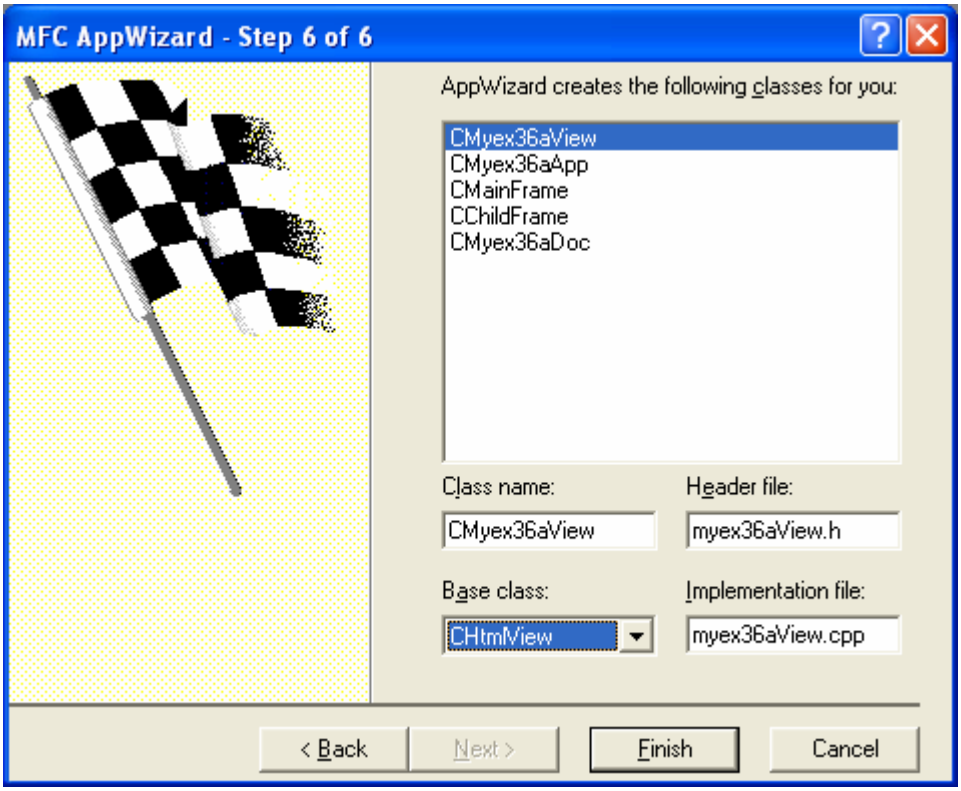

Figure 7: MYEX36A – AppWizard step 6 of 6, selecting `CHtmlView` as the View's base class.

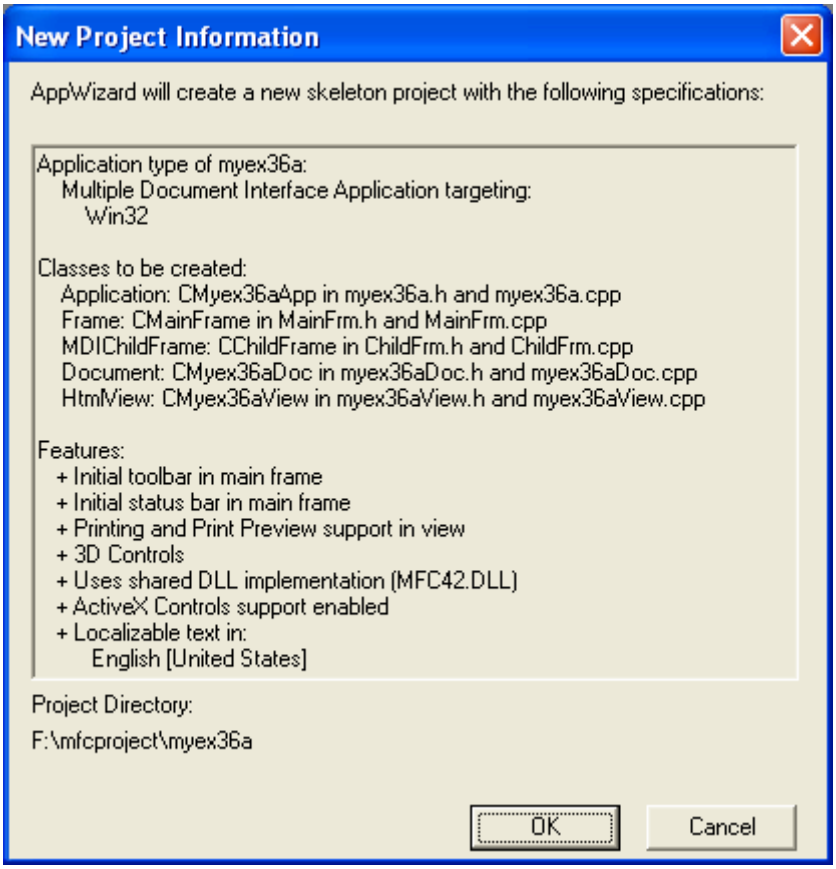

Figure 8: MYEX36A project summary.

Edit the URL to be loaded. In the `CMYEX36AView::OnInitialUpdate` function, you will see this line:

```
Navigate2(_T("http://www.microsoft.com/visualc/"),NULL,NULL);
```

You can edit this line to have the application load a local page or a URL other than the Visual C++ page. In this case change the `www` to `msdn`. Compile and run. Figure 9 shows the application running with the default Web page.

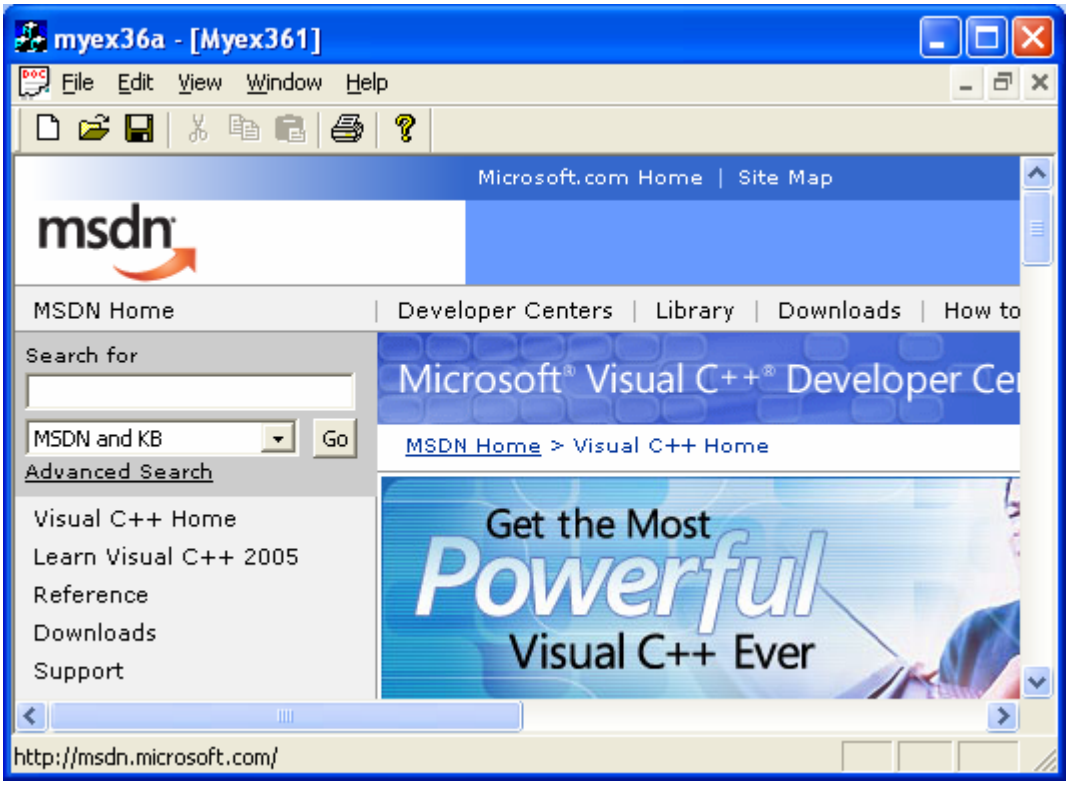

Figure 9: The MYEX36A program output example.

Now let's create a sample that really shows how to use DHTML with MFC. MYEX36B creates a `CHtmlView` object and a `CListView` object separated by a splitter. The example then uses DHTML to enumerate the HTML elements in the `CHtmlView` object and displays the results in the `CListView` object. The end result will be a DHTML explorer that you can use to view the DHTML object model of any HTML file. Here are the steps to create MYEX36B.

Run AppWizard and create myex36b. Choose **New** from Visual C++'s **File** menu. Then click the **Projects** tab, and select **MFC AppWizard (exe)**. Accept all the defaults but three: select **Single Document**, select **Windows Explorer** in Step 5, and select `CHtmlView` as the **Base Class** in Step 6. The options that you should see after finishing the wizard are shown in the graphic below.

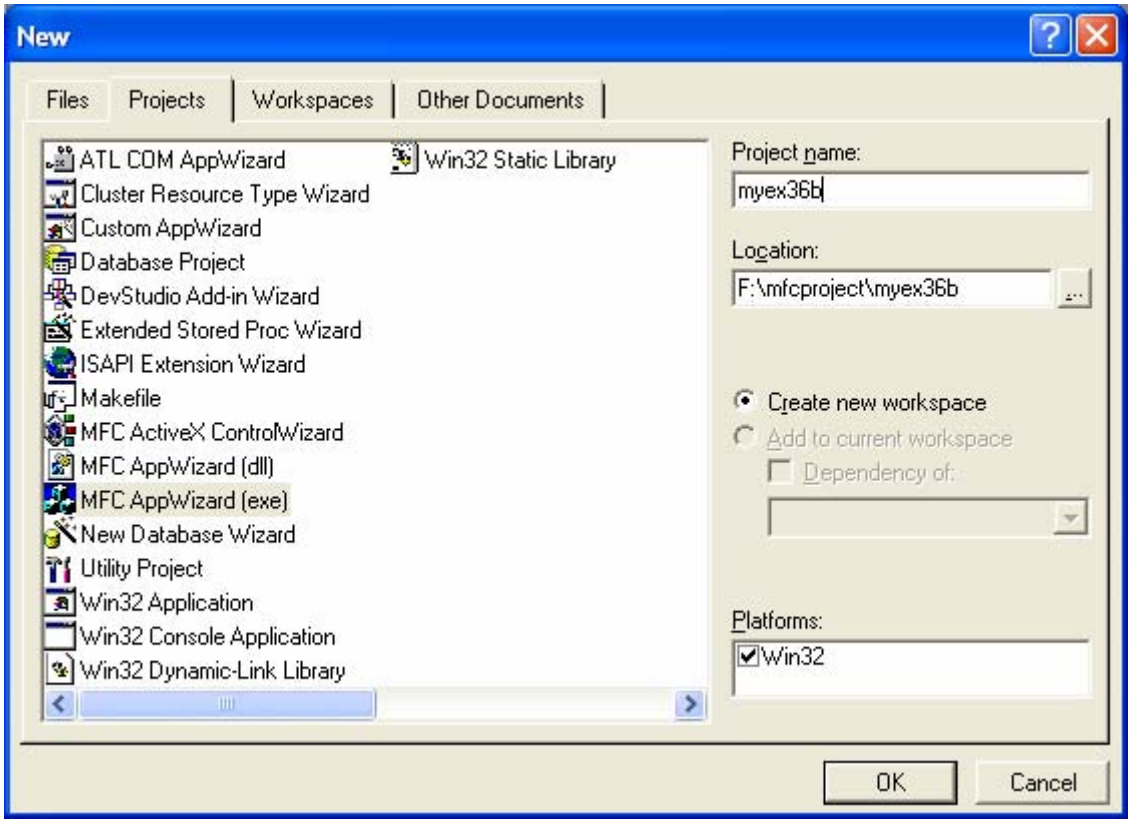

Figure 10: MYEX36B – Visual C++ MFC AppWizard new project dialog.

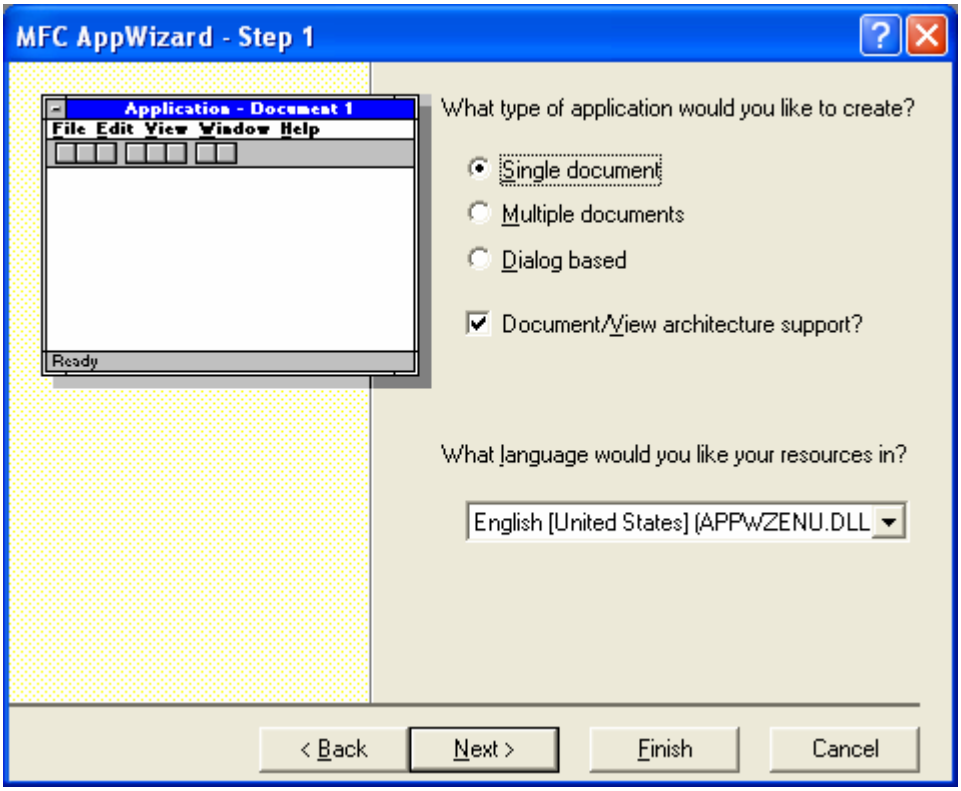

Figure 11: MYEX36B – AppWizard step 1 of 6.

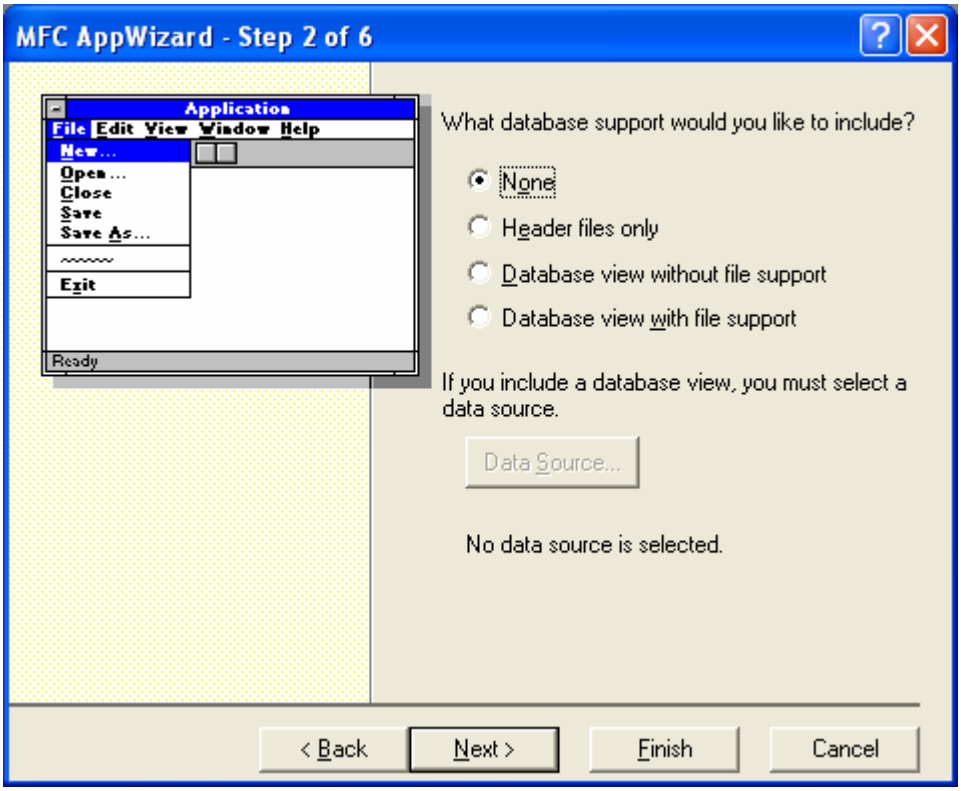

Figure 12: MYEX36B – AppWizard step 2 of 6.

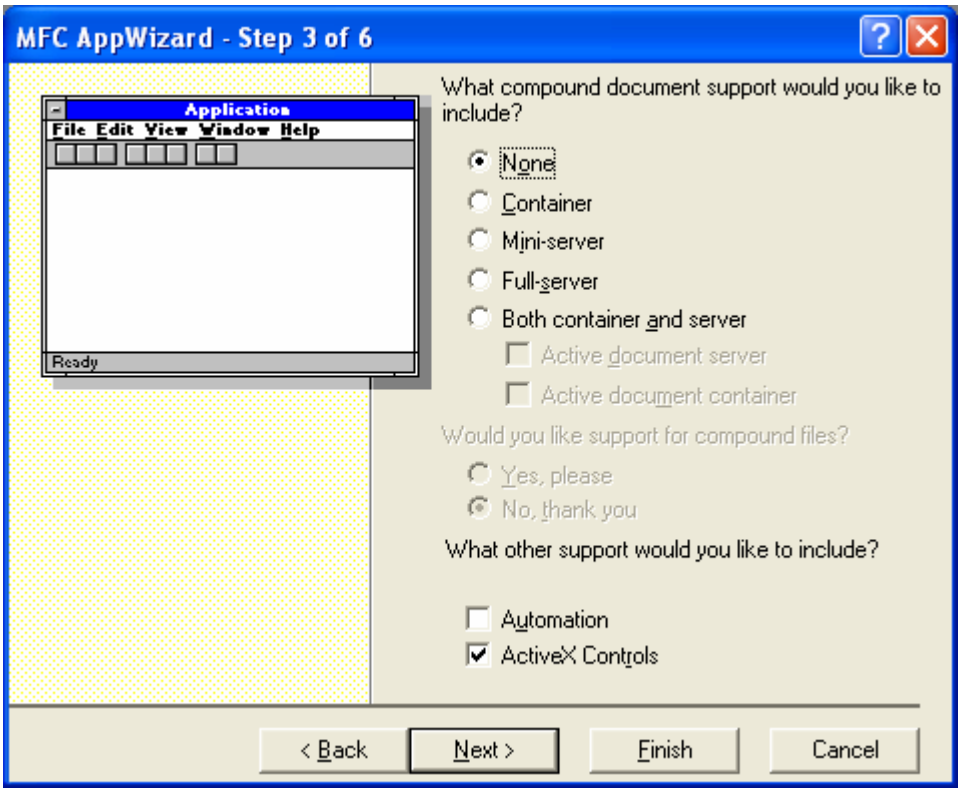

Figure 13: MYEX36B – AppWizard step 3 of 6.

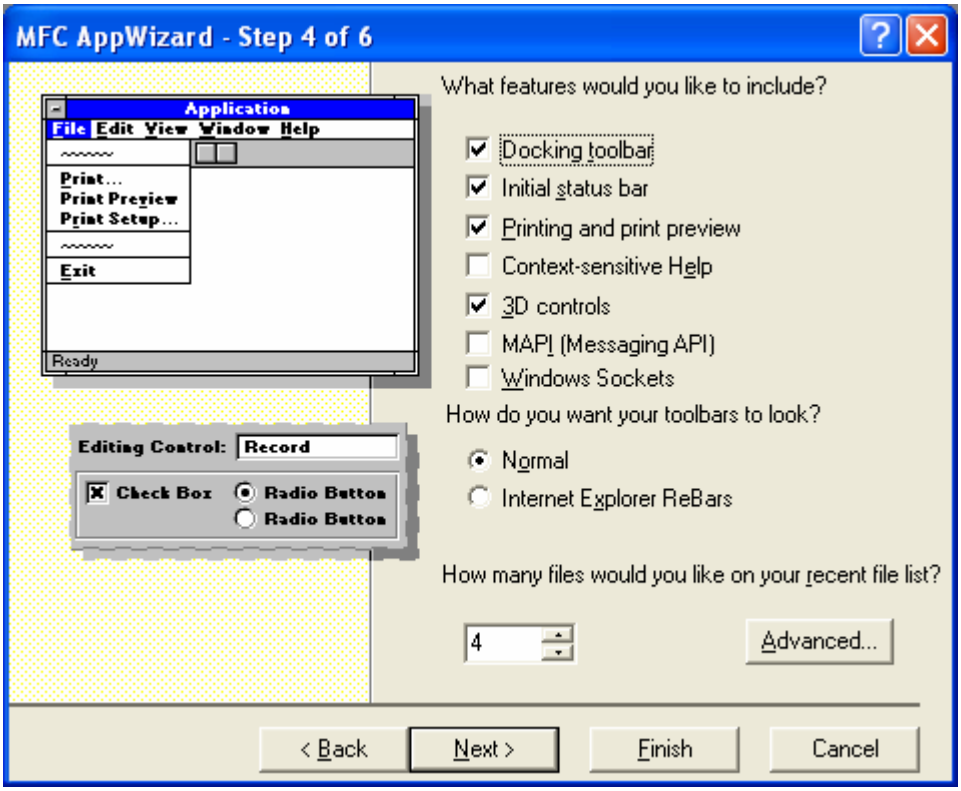

Figure 14: MYEX36B – AppWizard step 4 of 6.

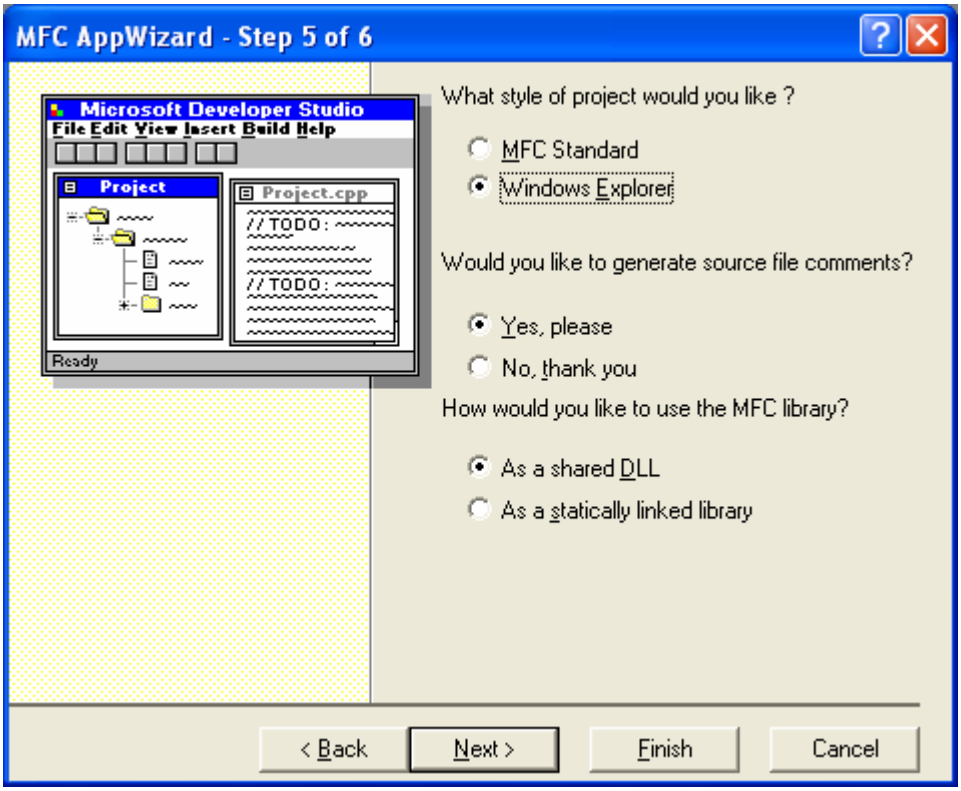

Figure 15: MYEX36B – AppWizard step 5 of 6, selecting Windows Explorer style.

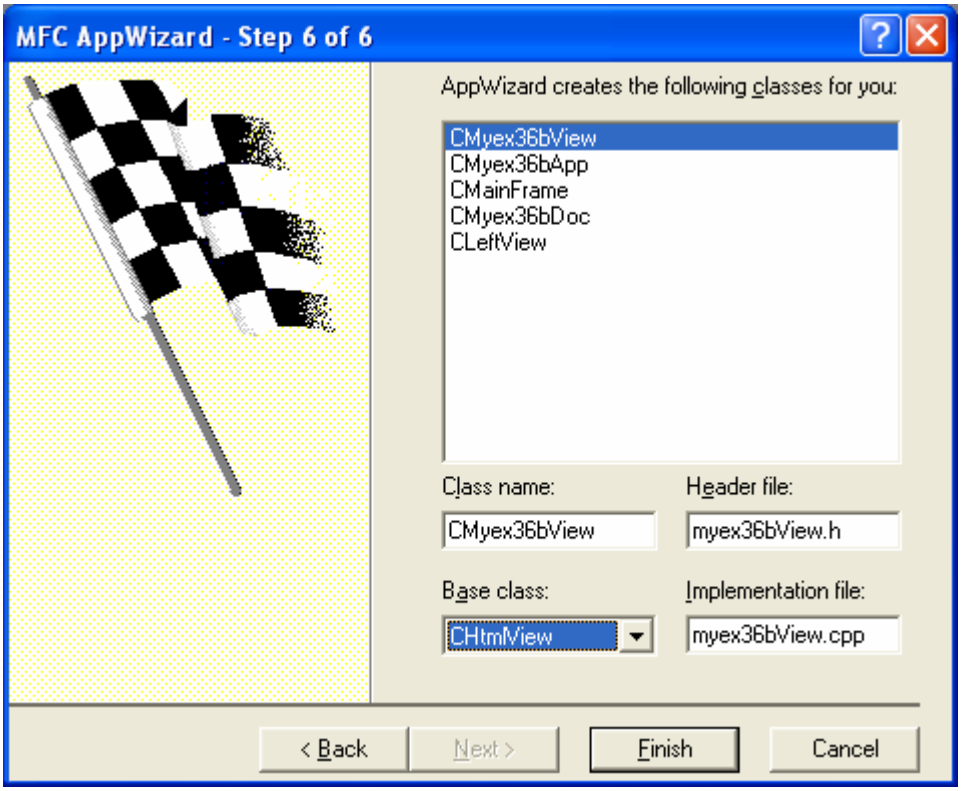

Figure 16: MYEX36B – AppWizard step 6 of 6, selecting `CHtmlView` as the View's base class.

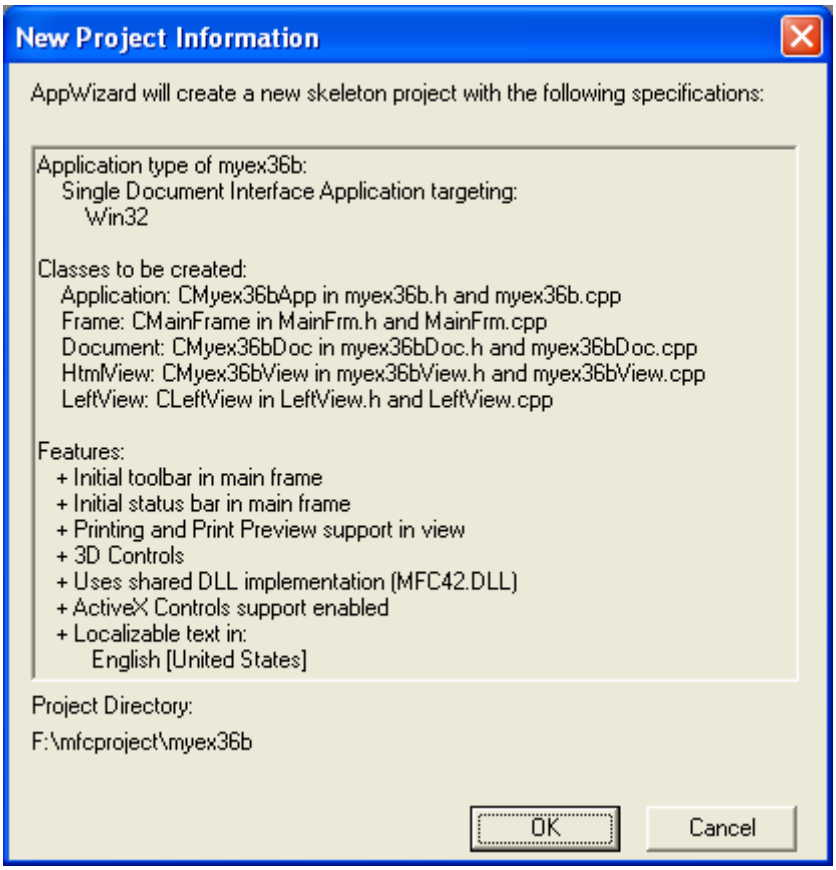

Figure 17: MYEX36B project summary.

Change the `CLeftView` to be a `CListView` derivative. By default, AppWizard makes the `CLeftView` of the splitter window a `CTreeView` derivative. Open the **LeftView.h** file, and do a global search for `CTreeView` and replace it with `CListView`. Open **LeftView.cpp** and do the same find and replace (Hint: Use Edit/Replace/Replace All.)

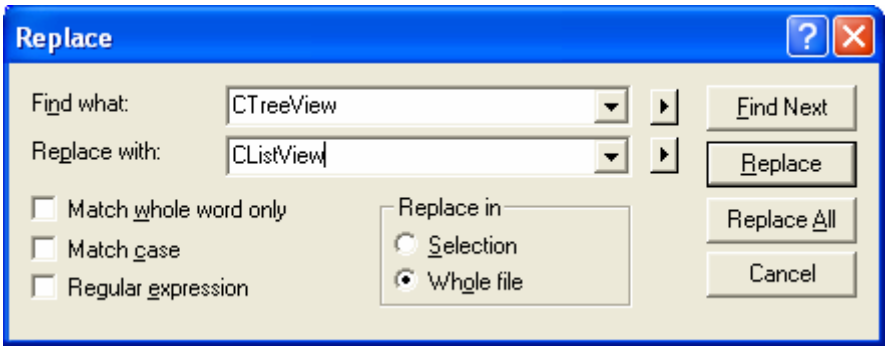

Figure 18: Replacing `CTreeView` with `CListView` in a file.

Edit the URL to be loaded. In the `CMyex36bView::OnInitialUpdate` function, change the URL to **http://www.microsoft.com/windows/ie/**.

```
void CMyex36bView::OnInitialUpdate()
{
    CHtmlView::OnInitialUpdate();

    // TODO: This code navigates to a popular spot on the web.
    //  change the code to go where you'd like.
    Navigate2(_T("http://www.microsoft.com/windows/ie/"),NULL,NULL);
}
```

Listing 1.

Add a `DoDHTMLExplore()` function to `CMainFrame`. First add the following declaration to the **MainFrm.h** file:

```
virtual void DoDHTMLExplore(void);
```

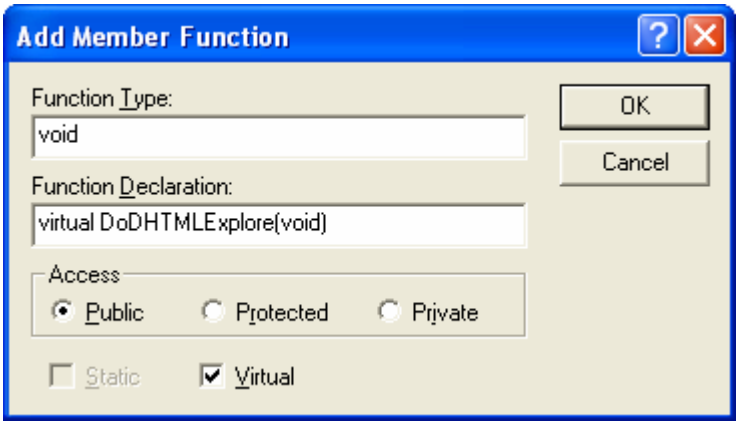

Figure 19: Adding virtual function.

```
// Implementation
public:
    virtual void DoDHTMLExplore(void);
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
```

Listing 2.

Now add the implementation for `DoHTMLExplore()` to **MainFrm.cpp**.

```
void CMainFrame::DoDHTMLExplore(void)
{

    CLeftView *pListView =
        (CLeftView *)m_wndSplitter.GetPane(0,0);

    CMyex36bView * pDHTMLView =
        (CMyex36bView *)m_wndSplitter.GetPane(0,1);

    //Clear the listview
    pListView->GetListCtrl().DeleteAllItems();
    IDispatch* pDisp = pDHTMLView->GetHtmlDocument();

    if (pDisp != NULL)
    {
        IHTMLDocument2* pHTMLDocument2;
        HRESULT hr;

        hr = pDisp->QueryInterface(IID_IHTMLDocument2, (void**)&pHTMLDocument2
);
        if (hr == S_OK)
        {
            IHTMLElementCollection* pColl = NULL;

            hr = pHTMLDocument2->get_all( &pColl );
            if (hr == S_OK && pColl != NULL)
            {
                LONG celem;
                hr = pColl->get_length( &celem );

                if ( hr == S_OK )
                {
                    for ( int i=0; i< celem; i++ )
                    {
                        VARIANT varIndex;
                        varIndex.vt = VT_UINT;
                        varIndex.lVal = i;
                        VARIANT var2;
                        VariantInit( &var2 );
                        IDispatch* pDisp;

                        hr = pColl->item( varIndex, var2, &pDisp );
                        if ( hr == S_OK )
                        {
                            IHTMLElement* pElem;

                            hr = pDisp->QueryInterface(
```

```
                                IID_IHTMLElement, (void **)&pElem);
                            if ( hr == S_OK )
                            {
                                BSTR bstr;
                                hr = pElem->get_tagName(&bstr);
                                CString strTag = bstr;
                                IHTMLImgElement* pImgElem;

                                //Is it an image element?
                                hr = pDisp->QueryInterface(
                                    IID_IHTMLImgElement,
                                    (void **)&pImgElem );
                                if (hr == S_OK)
                                {
                                    pImgElem->get_href(&bstr);
                                    strTag += " - ";
                                    strTag += bstr;
                                    pImgElem->Release();
                                }
                                else
                                {
                                    IHTMLAnchorElement* pAnchElem;

                                    //Is it an anchor?
                                    hr = pDisp->QueryInterface(
                                        IID_IHTMLAnchorElement,
                                        (void **)&pAnchElem );
                                    if (hr == S_OK)
                                    {
                                        pAnchElem->get_href(&bstr);
                                        strTag += " - ";
                                        strTag += bstr;
                                        pAnchElem->Release();
                                    }
                                }//end of else

                                pListView->GetListCtrl().InsertItem(
                                    pListView->GetListCtrl()
                                    .GetItemCount(), strTag);
                                pElem->Release();
                            }
                            pDisp->Release();
                        }
                    }
                }
                pColl->Release();
            }
            pHTMLDocument2->Release();
        }
        pDisp->Release();
    }
}
```

```
////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::DoDHTMLExplore()
{
    CLeftView *pListView = (CLeftView *)m_wndSplitter.GetPane(0,0);

    CMyex36bView * pDHTMLView = (CMyex36bView *)m_wndSplitter.GetPane(0,1);

    //Clear the listview
    pListView->GetListCtrl().DeleteAllItems();
    IDispatch* pDisp = pDHTMLView->GetHtmlDocument();

    if (pDisp != NULL)
    {
        IHTMLDocument2* pHTMLDocument2;
        HRESULT hr;

        hr = pDisp->QueryInterface(IID_IHTMLDocument2, (void**)&pHTMLDocumer
        if (hr == S_OK)
        {
            IHTMLElementCollection* pColl = NULL;
```

Listing 3.

Here are the steps that this function takes to "explore" the HTML document using DHTMLs:

1. First `DoHTMLExplore()` gets pointers to the `CListView` and `CHtmlView` views in the splitter window.
2. Then it makes a call to `GetHtmlDocument()` to get an `IDispatch` pointer to the DHTML document object.
3. Next `DoHTMLExplore()` gets the `IHTMLDocument2` interface.
4. With the `IHTMLDocument2` interface, `DoHTMLExplore()` retrieves the all collection and iterates through it. In each iteration, `DoHTMLExplore()` checks the element type. If the element is an image or an anchor, `DoHTMLExplore()` retrieves additional information such as the link for the image. The all collection loop then places the textual description of the HTML element in the `CListView` object.

Make sure that **Mainfrm.cpp** includes **mshtml.h**. Add the following line to the top of **Mainfrm.cpp** so that the `DoHTMLExplore()` code will compile.

```
#include <mshtml.h>
```

```
#include "stdafx.h"
#include "myex36b.h"

#include "MainFrm.h"
#include "LeftView.h"
#include "myex36bView.h"
#include <mshtml.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

Listing 4.

Add a call to `DoHTMLExplore()`. For this example, we will change the `CMyex36bApp::OnAppAbout` function to call the `DoDHTMLExplore()` function in the **myex36b.cpp** file. Replace the existing code with the following code:

```
void CMyex36bApp::OnAppAbout()
{
    CMainFrame * pFrame = (CMainFrame*)AfxGetMainWnd();
```

```
        pFrame->DoDHTMLExplore();

    }
```

```
// App command to run the dialog
void CMyex36bApp::OnAppAbout()
{
    CMainFrame * pFrame = (CMainFrame*)AfxGetMainWnd();
    pFrame->DoDHTMLExplore();
}
                        void __thiscall CMainFrame::DoDHTMLExplore(void)
/////////////////////////////////////////////////////////////////////////////
// CMyex36bApp message handlers
```

Listing 5.

Customize the list view. In the `CLeftView::PreCreateWindow` function (**LeftView.cpp**), add this line:

```
   cs.style |= LVS_LIST;
```

```
BOOL CLeftView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    cs.style |= LVS_LIST;
    return CListView::PreCreateWindow(cs);
}
```

Listing 6.

Finally, as in MYEX36A, change the URL address or local HTML file in the `CMyex36bView::OnInitialUpdate()`.

```
    void CMyex36bView::OnInitialUpdate()
    {
        CHtmlView::OnInitialUpdate();

        // TODO: This code navigates to a popular spot on the web.
        //  change the code to go where you'd like.
        Navigate2(_T("http://www.microsoft.com/windows/ie/"),NULL,NULL);
    }
```

```
void CMyex36bView::OnInitialUpdate()
{
    CHtmlView::OnInitialUpdate();

    // TODO: This code navigates to a popular spot on the web.
    //  change the code to go where you'd like.
    Navigate2(_T("http://www.microsoft.com/windows/ie/"),NULL,NULL);
}
```

Listing 7.

Compile and run. Compile and run the sample. Press the "?" toolbar item, or choose Help/About to invoke the explore function. Figure 20 shows the MYEX36B example in action.

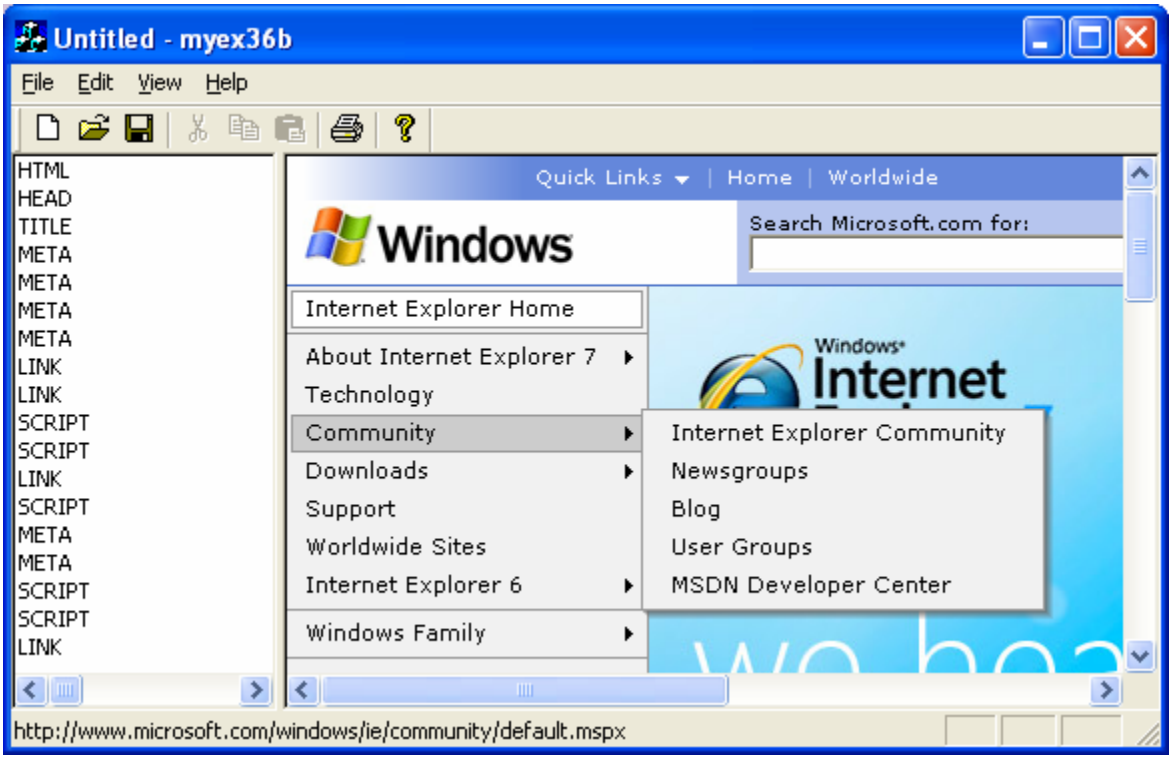

Figure 20: The MYEX36B program example in action.

Now that you've seen how to use DHTML and MFC, let's look at how ATL implements DHMTL support.

ATL and DHTML

ATL's support for DHTML comes in the form of an HTML object that can be embedded in any ATL ActiveX control. MYEX36C creates an ATL control that illustrates DHTML support. To create the example, follow these steps:

1. Run the **ATL COM AppWizard** and create myex36c. Choose **New** from Visual C++'s **File** menu. Then click the **Projects** tab, and select **ATL COM AppWizard**. Choose **Executable** as the server type.

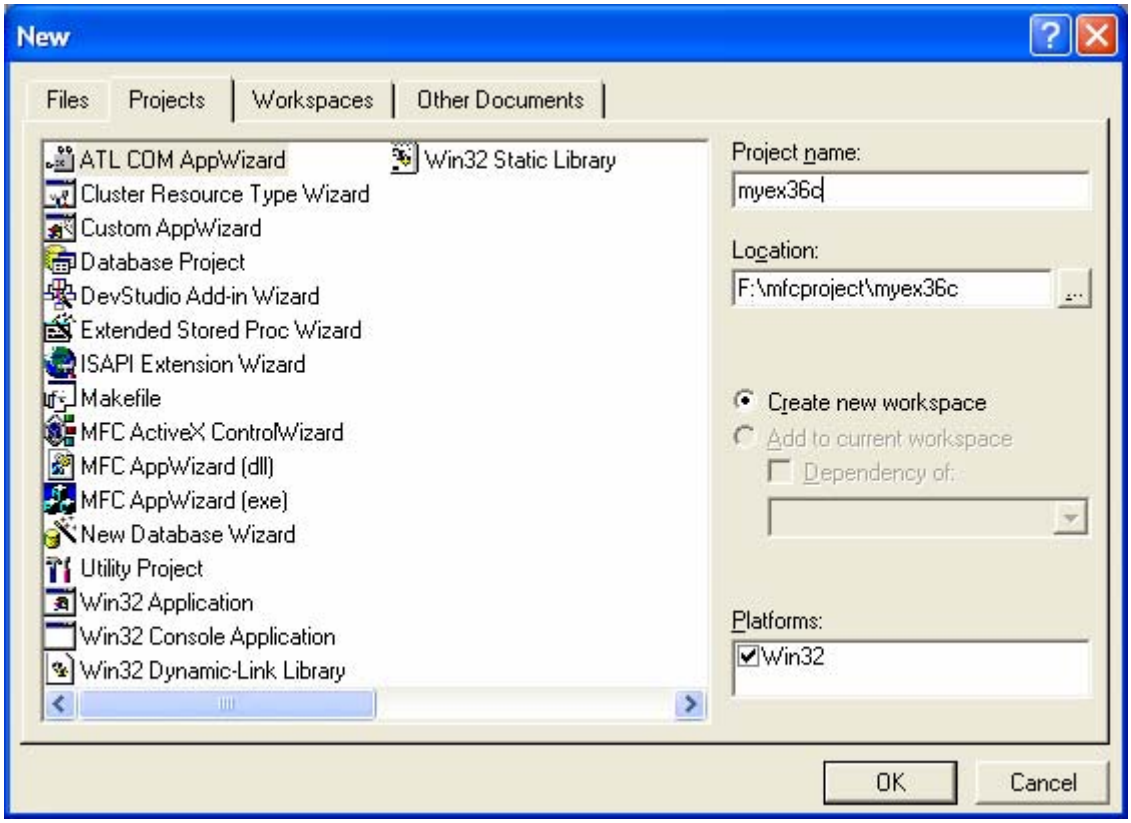

Figure 21: MYEX36C – ATL COM AppWizard new project dialog.

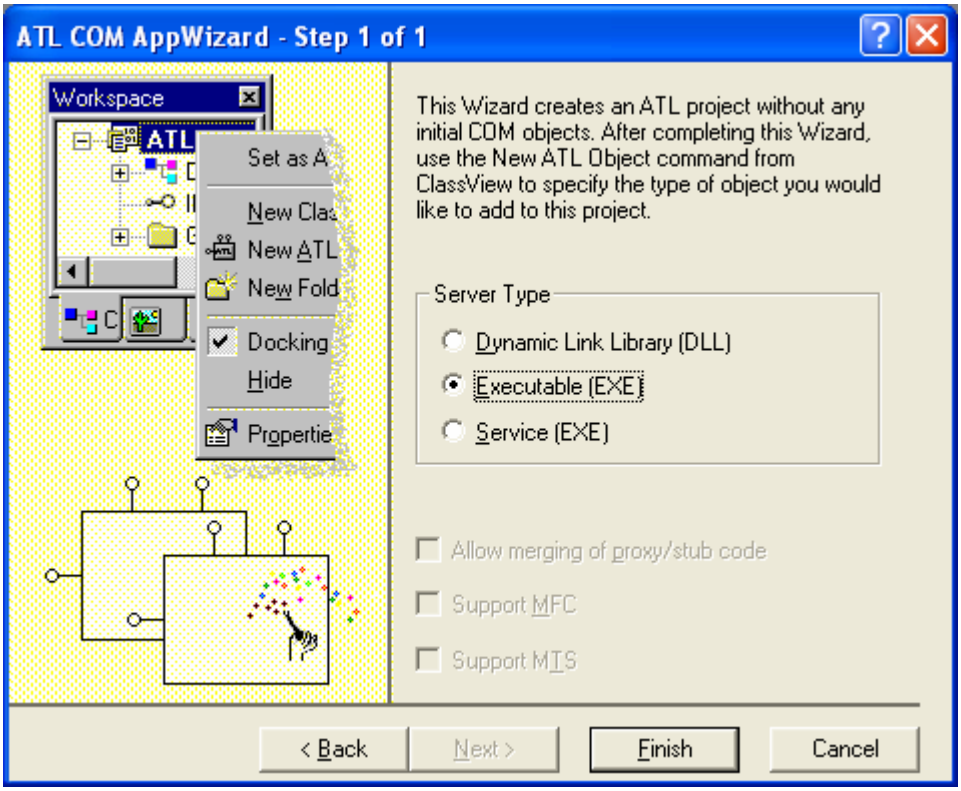

Figure 22: MYEX36C – ATL COM AppWizard step 1 of 1.

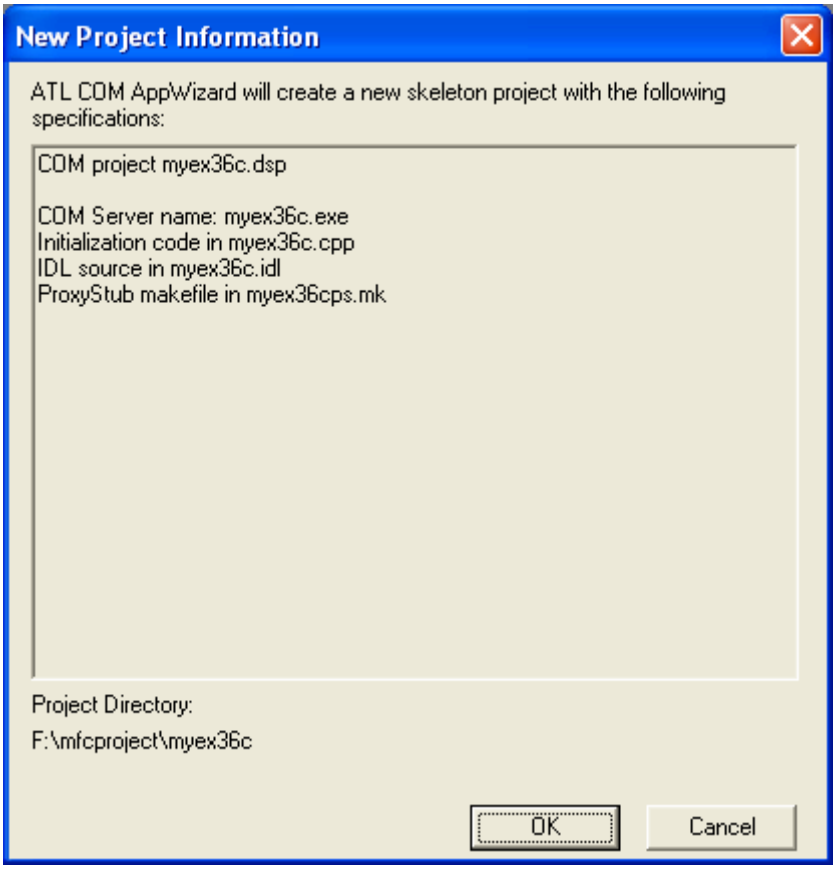

Figure 23: MYEX36C project summary.

2. Insert an HTML control. In ClassView, right-click on the myex36c classes item and select **New ATL Object**. Select **Controls** and **HTML Control** as shown in the graphic below.

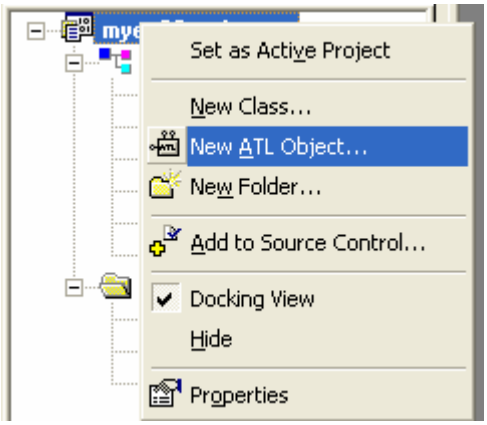

Figure 24: Adding new ATL object to project.

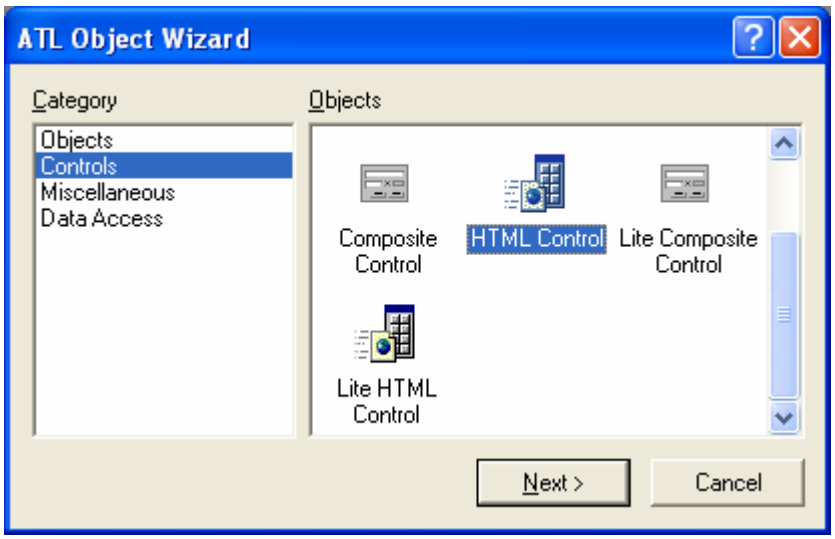

Figure 25: Selecting ATL object category.

3. Click **Next** and fill in the **C++ Short Name** as shown here.

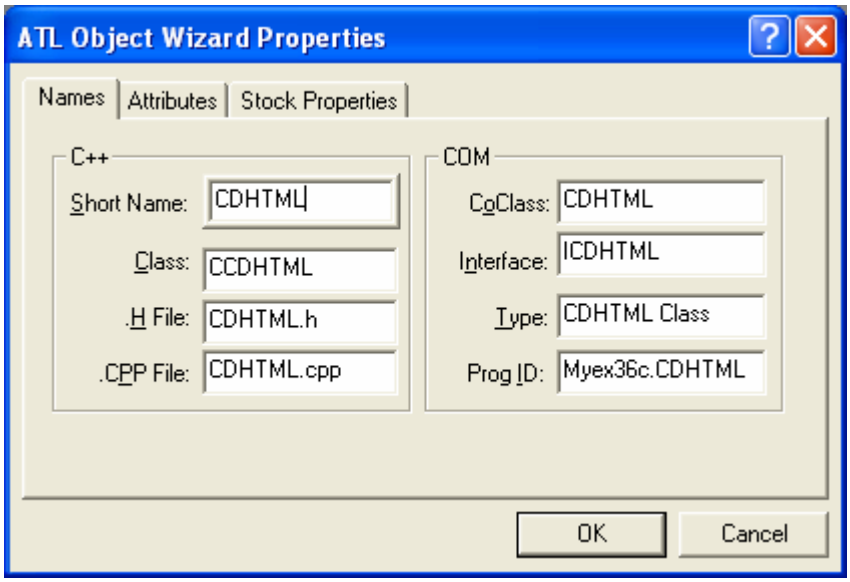

Figure 26: Entering the ATL object's name.

If you look at the `IDHTMLUI` object, you will see this stock implementation of the `OnClick()` handler:

```
STDMETHOD(OnClick)(IDispatch* pdispBody, VARIANT varColor)
{
    CComQIPtr<IHTMLBodyElement> spBody(pdispBody);
    if (spBody != NULL)
        spBody->put_bgColor(varColor);
    return S_OK;
}
```

```
// ICDHTMLUI
public:
    // Example method called by the HTML to change the <BODY> background color
    STDMETHOD(OnClick)(IDispatch* pdispBody, VARIANT varColor)
    {
        CComQIPtr<IHTMLBodyElement> spBody(pdispBody);
        if (spBody != NULL)
            spBody->put_bgColor(varColor);
        return S_OK;
    }
```

Listing 8.

The default `OnClick()` handler uses `QueryInterface()` on the `IDispatch` pointer to get the `IHTMLBodyElement` object. The handler then calls the `put_bgColor` method to change the background color.

4. Compile, load, and run the control to see the ATL DHTML code in action. After you build the project, select **ActiveX Control Test Container** from the **Tools** menu. In the test container, select **Insert New Control** from the **Edit** menu and choose **CDHTML Class** from the list box. Figure 29 shows the resulting ActiveX control that uses DHTML to change the background when the user clicks the button.

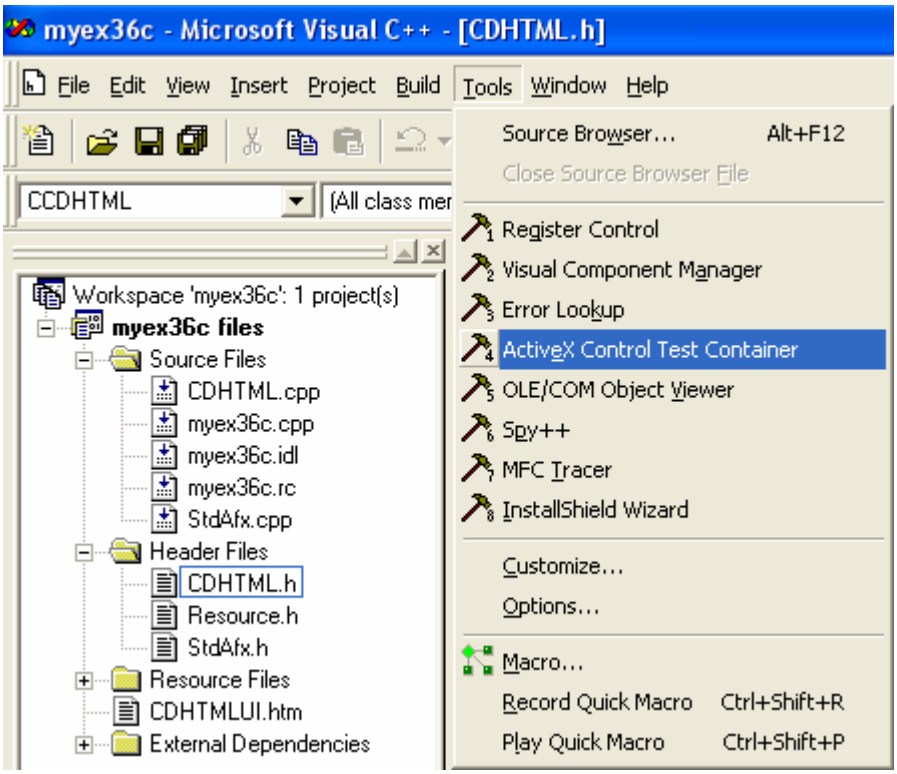

Figure 27: Invoking the ActiveX Control Test Container.

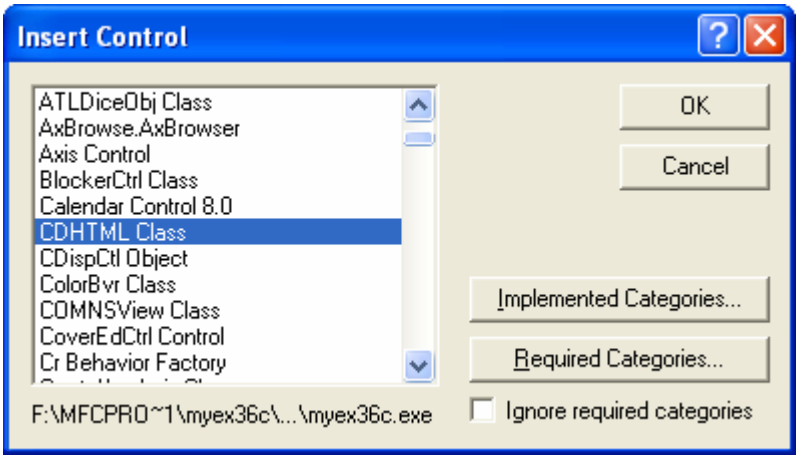

Figure 28: Inserting **CDHTML Class** control to ActiveX Control Test Container.

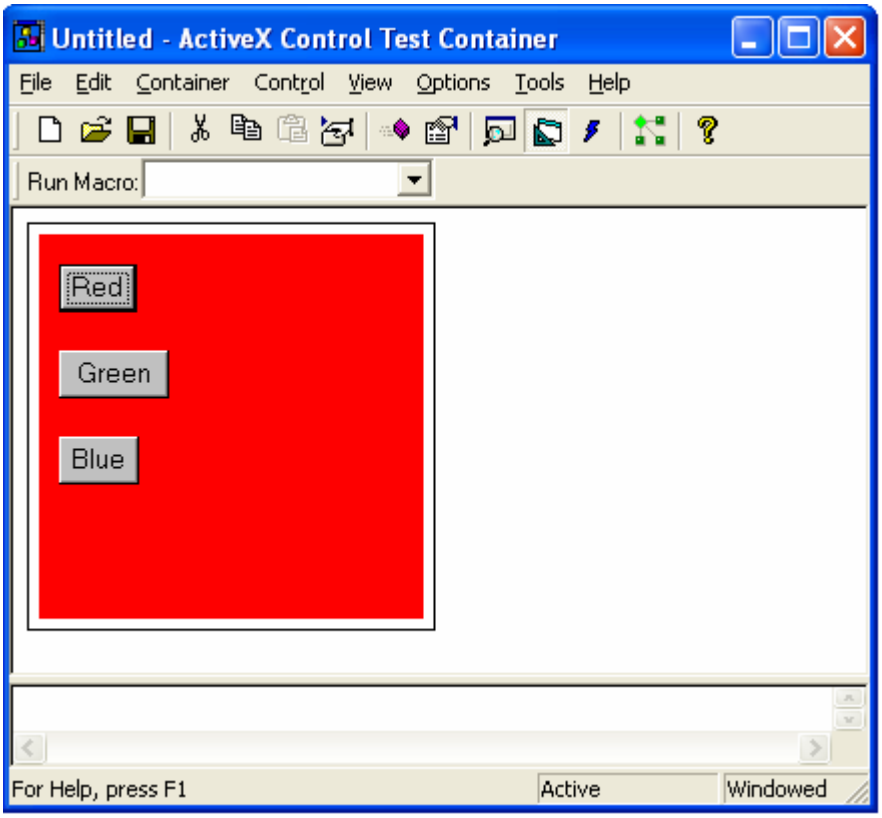

Figure 29: MYEX36C ActiveX control program example.

----------End----------

Further reading and digging:

1. MSDN MFC 6.0 class library online documentation - used throughout this Tutorial.
2. MSDN MFC 7.0 class library online documentation - used in .Net framework and also backward compatible with 6.0 class library.
3. MSDN Library
4. Windows data type.
5. Win32 programming Tutorial.
6. The best of C/C++, MFC, Windows and other related books.

7. Unicode and Multibyte character set: Story and program examples.