

## ActiveX Document Servers and the Internet

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](#).

### Introduction

#### ActiveX Document Theory

#### ActiveX Document Servers vs. OLE Embedded Servers

#### Running an ActiveX Document Server from Internet Explorer

#### ActiveX Document Servers vs. ActiveX Controls

#### OLE Interfaces for ActiveX Document Servers and Containers

#### MFC Support for ActiveX Document Servers

#### COleServerDoc

#### CDocObjectServerItem

#### CDocObjectServer

#### COleDocIPFrameWnd

#### ActiveX Document Server Example MYEX35A

#### MYEX35A Phase 1: A Simple Server

#### MYEX35A From Scratch

#### Debugging an ActiveX Document Server

#### MYEX35A: Phase 2 - Adding More Functionality

#### The Coding Part

#### CMainFrame Class

#### CMyex35aApp Class

#### CMyex35aDoc Class

#### CMyex35aView Class

#### The Story of MYEX35A Phase 2: Adding WinInet Calls

#### Displaying Bitmaps on Buttons

#### ActiveX Document Server Example MYEX35B

#### MYEX35B From Scratch

#### The Coding Part

#### CMainFrame Class

#### CInPlaceFrame Class

#### CMyex35bDoc Class

#### CMyex35bView Class

#### The Story

#### Field Validation in an MFC Form View

#### Generating POST Requests Under Program Control

#### The MYEX35B View Class

#### Building and Testing MYEX35B

#### ActiveX Document Servers vs. VB Script

#### Going Further with ActiveX Document Servers

### Introduction

An ActiveX document is a special file that you can download from a Web server. When the browser sees an ActiveX document file, it automatically loads the corresponding ActiveX document server program from your hard disk, and that program takes over the whole browser window to display the contents of the document. The Microsoft Internet Explorer

browser is not the only **ActiveX document container program**. The Microsoft Office Binder program also runs ActiveX document server programs, storing the several ActiveX documents in a single disk file. In the COM world, an ActiveX document server program is called a server because it implements a COM component. The container program (Internet Explorer or Office Binder) creates and controls that COM component. In the Internet world, the same program looks like a client because it can request information from a remote host (Microsoft Internet Information Server).

In this module, you'll learn about ActiveX document servers and ActiveX documents and you'll build two ActiveX document servers that work over the Internet in conjunction with Internet Explorer.

## ActiveX Document Theory

It's helpful to put ActiveX documents within the context of COM and OLE, which you already understand if you've read the other modules in this book. You can, however, get started with ActiveX document servers without fully understanding all the COM concepts.

## ActiveX Document Servers vs. OLE Embedded Servers

As you saw in [Module 27](#), an OLE embedded server program runs in a child window of an OLE container application and occupies a rectangular area in a page of the container's document (see Figure 28-1). Unless an embedded server program is classified as a mini-server, it can run stand-alone also. In embedded mode, the server program's data is held in a storage inside the container application's file. The embedded server program takes over the container program's menu and toolbar when the user activates it by double-clicking on its rectangle.

In contrast to an embedded server, an ActiveX document server takes over a whole frame window in its container application, and the document is always active. An ActiveX server application, running inside a container's frame window, runs pretty much the same way it would in stand-alone mode. You can see this for yourself if you have Microsoft Office 97 (not available in Office 2000/2003 and newer version). Office 97 for example, includes an ActiveX container program called **Binder** (accessible from the Office shortcut bar), and the Office applications (Microsoft Word, Microsoft Excel, and so on) have ActiveX server capability. Figure 1 shows a Word document and an Excel chart inside the same binder.

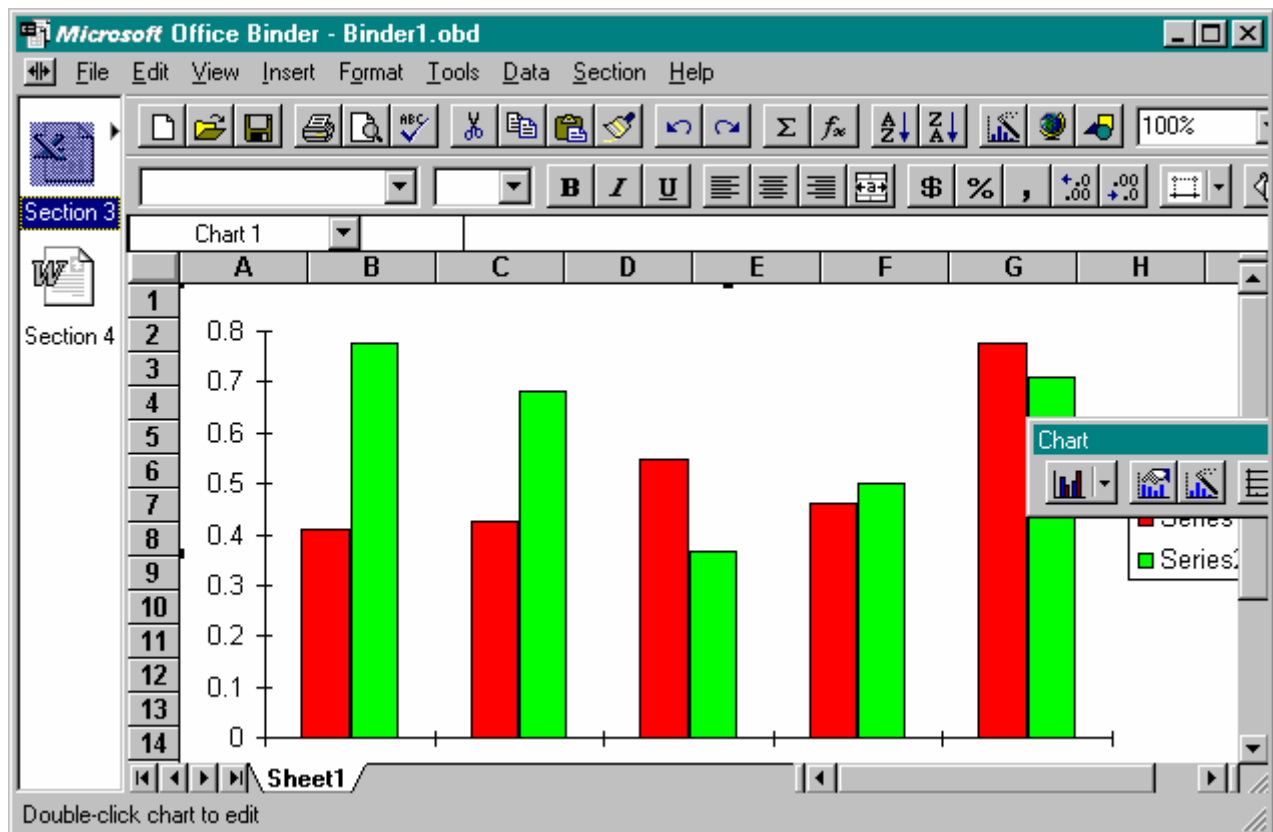


Figure 1: A Word document and an Excel chart inside a Microsoft Office Binder window.

Like an embedded server, the ActiveX document server saves its data in a storage inside the ActiveX container's file. When the Office user saves the Binder program from the **File** menu, Binder writes a single OBD file to disk; the file contains one storage for the Word document and another for the Excel spreadsheet. You can see this file structure yourself with the DFVIEW utility, as shown in Figure 2.

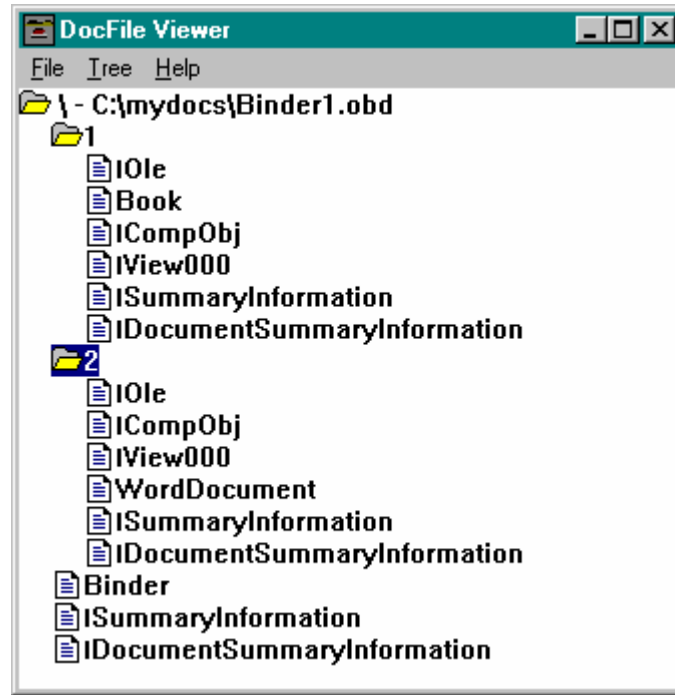


Figure 2: A file structure displayed by the DocFile Viewer.

## Running an ActiveX Document Server from Internet Explorer

Running an ActiveX document server from Internet Explorer is more fun than running one from Microsoft Office Binder (Internet Explorer refers to Internet Explorer 3.0 or greater). Rather than load a storage only from an OBD file, the server program can load its storage from the other side of the world. You just type in a URL, such as <http://www.DaliLama.in/SecretsOfTheUniverse.doc>, and a Microsoft Word document opens inside your Browser window, taking over the browser's menu and toolbar. That's assuming, of course, that you have installed the Microsoft Word program. If not, a Word document viewer is available, but it must be on your hard disk before you download the file.

An ActiveX document server won't let you save your changes back to the Internet host, but it will let you save them on your own hard disk. In other words, **File Save** is disabled but **File Save As** is enabled.

If you have Microsoft Office, try running Word or Excel in Internet Explorer now. The MYEX34A server is quite capable of delivering documents or worksheets to your browser, assuming that they are accessible from its home directory. Note that Internet Explorer recognizes documents and worksheets **not by their file extensions** but by the **CLSID** inside the files. You can prove this for yourself by renaming a file prior to accessing it.

## ActiveX Document Servers vs. ActiveX Controls

Both ActiveX document servers and ActiveX controls can run with and without the Internet. Both are compiled programs that can run inside a browser. The following table lists some of the differences between the two.

	ActiveX Document Server	ActiveX Control
Module type	EXE	Most often a DLL
Can run stand-alone	Yes	No
Code automatically downloaded and	No	Yes

registered by a WWW browser		
Can be embedded in an HTML file	No	Yes
Occupies the entire browser window	Yes	Sometimes
Can be several pages	Yes	Not usually
Can read/write disk files	Yes	Not usually

Table 1.

## OLE Interfaces for ActiveX Document Servers and Containers

ActiveX document servers implement the same interfaces as OLE embedded servers, including `IOleObject`, `IOleInPlaceObject`, and `IOleInPlaceActiveObject`. ActiveX document containers implement `IOleClientSite`, `IOleInPlaceFrame`, and `IOleInPlaceSite`. The menu negotiation works the same as it does for Visual Editing.

Some additional interfaces are implemented, however. ActiveX document servers implement `IOleDocument`, `IOleDocumentView`, `IOleCommandTarget`, and `IPrint`. ActiveX document containers implement `IOleDocumentSite`. The architecture allows for multiple views of the same document - sort of like the MFC document-view architecture - but most ActiveX document servers implement only one view per document.

The critical function in an OLE embedded server is `IOleObject::DoVerb`, which is called by the container when the user double-clicks on an embedded object or activates it through the menu. For an ActiveX document server, however, the critical function is `IOleDocumentView::UIActivate`. (Before calling this function, the container calls `IOleDocument::CreateView`, but generally the server just returns an interface pointer to the single document-view object.) `UIActivate` finds the container site and frame window, sets that window as the server's parent, sets the server's window to cover the container's frame window, and then activates the server's window.

It's important to realize that the COM interaction takes place between the container program (Internet Explorer or Binder) and the ActiveX document server (your program), which both are running on the client computer. We know of no cases in which remote procedure calls (RPCs) are made over the Internet. That means that the remote host (the server computer) does not use COM interfaces to communicate with clients, but it can deliver data in the form of storages.

## MFC Support for ActiveX Document Servers

MFC allows you to create your own ActiveX document server programs. In addition, Visual C++ 6.0 allows you to write ActiveX document containers. To get a server program, create a new **MFC AppWizard EXE** project and then check the **Active Document Server** check box, as shown in Figure 3. To create a container program, just make sure the **Active Document Container** check box is marked.

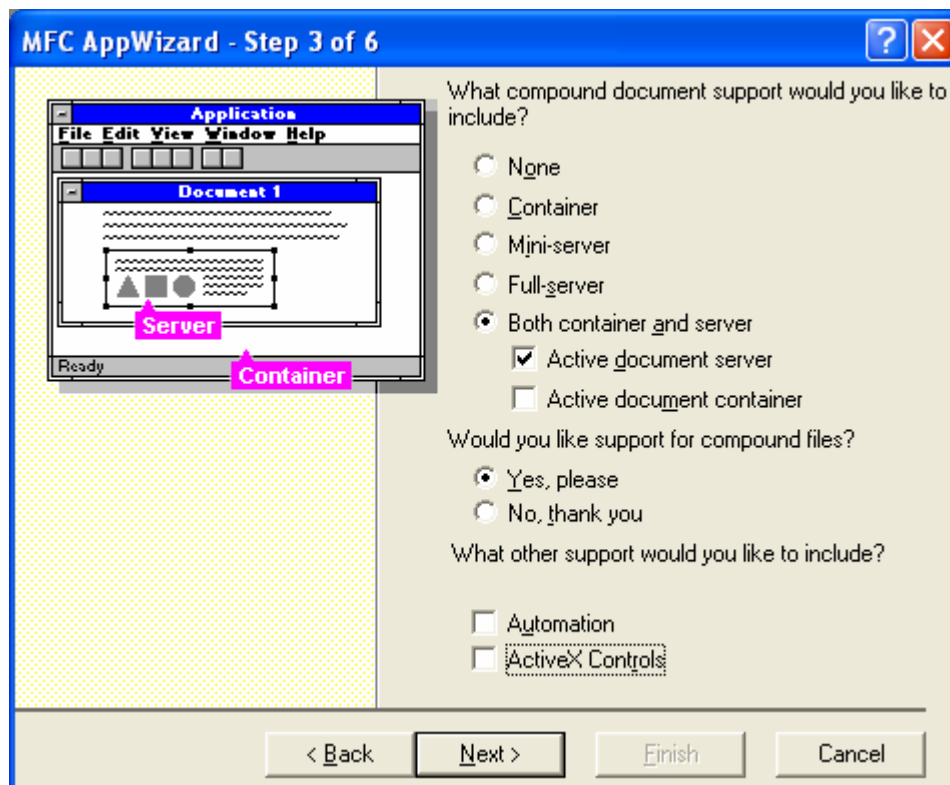


Figure 3: Step 3 of the MFC AppWizard.

Here's a rundown of the classes involved in MFC's ActiveX Document Server Architecture.

### **COleServerDoc**

As it is for any COM component, your ActiveX document server's document class is derived from `COleServerDoc`, which implements `IPersistStorage`, `IObject`, `IDataObject`, `IObjectInPlaceObject`, and `IObjectInPlaceActiveObject`. The COM interfaces and MFC classes discussed here were named before Microsoft introduced ActiveX technology. An ActiveX document server was formerly known as a **document object server** or a **doc object server**, so those are the names you'll see in the source code and in some online documentation.

### **CDocObjectServerItem**

This class is derived from the `COleServerItem` class used in embedded servers. Your ActiveX document server program has a class derived from `CDocObjectServerItem`, but that class isn't used when the program is running in ActiveX document mode.

### **CDocObjectServer**

This class implements the new ActiveX server interfaces. Your application creates an object of class `CDocObjectServer` and attaches it to the `COleServerDoc` object. If you look at `COleServerDoc::GetDocObjectServer` in your derived document class, you'll see the construction code. Thereafter, the document object and attached `CDocObjectServer` object work together to provide ActiveX document server functionality. This class implements both `IObjectDocument` and `IObjectDocumentView`, which means that you can have only one view per document in an MFC ActiveX document server. You generally don't derive classes from `CDocObjectServer`.

### **COleDocIPFrameWnd**

This class is derived from `COleIPFrameWnd`. Your application has a frame window class derived from `COleDocIPFrameWnd`. The framework constructs an object of that class when the application starts in embedded server mode or in ActiveX document server mode. In ActiveX document server mode, the server's window completely covers the container's frame window and has its own menu resource attached, with the identifier `IDR_SRVR_INPLACE` (for an SDI application).

## ActiveX Document Server Example MYEX35A

You could construct the MYEX35A example in two phases. The first phase is a plain ActiveX document server that loads a file from its container. The view base class is `CRichEditView`, which means the program loads, edits, and stores text plus embedded objects. In the second phase, the application is enhanced to download a separate text file from the Internet one line at a time, demonstrating that ActiveX document servers can make arbitrary WinInet calls.

### MYEX35A Phase 1: A Simple Server

The MYEX35A example in the download page is complete with the text download feature from Phase 2. You can exercise its Phase 1 capabilities by building it, or you can create a new application with AppWizard. If you do use AppWizard, you should refer to Figure 3 to see the AppWizard EXE project dialog and select the appropriate options. All other options are the default options, except those for selecting SDI (Step 1), setting the project's filename extension to **35a** using the **Advanced** button in Step 4, and changing the view's base class (`CRichEditView` - on the wizard's last page). You don't have to write any C++ code at all.

Be sure to run the program once in stand-alone mode to register it. While the program is running in stand-alone mode, type some text (and insert some OLE embedded objects) and then save the document as **test.35a** in your Internet server's home directory (`\scripts` or `\wwwroot` directory). Try loading **test.35a** from Internet Explorer and from Office Binder.

### MYEX35A From Scratch

Let build MYEX35A phase 1, an ActiveX document server from scratch. Follow the shown steps.

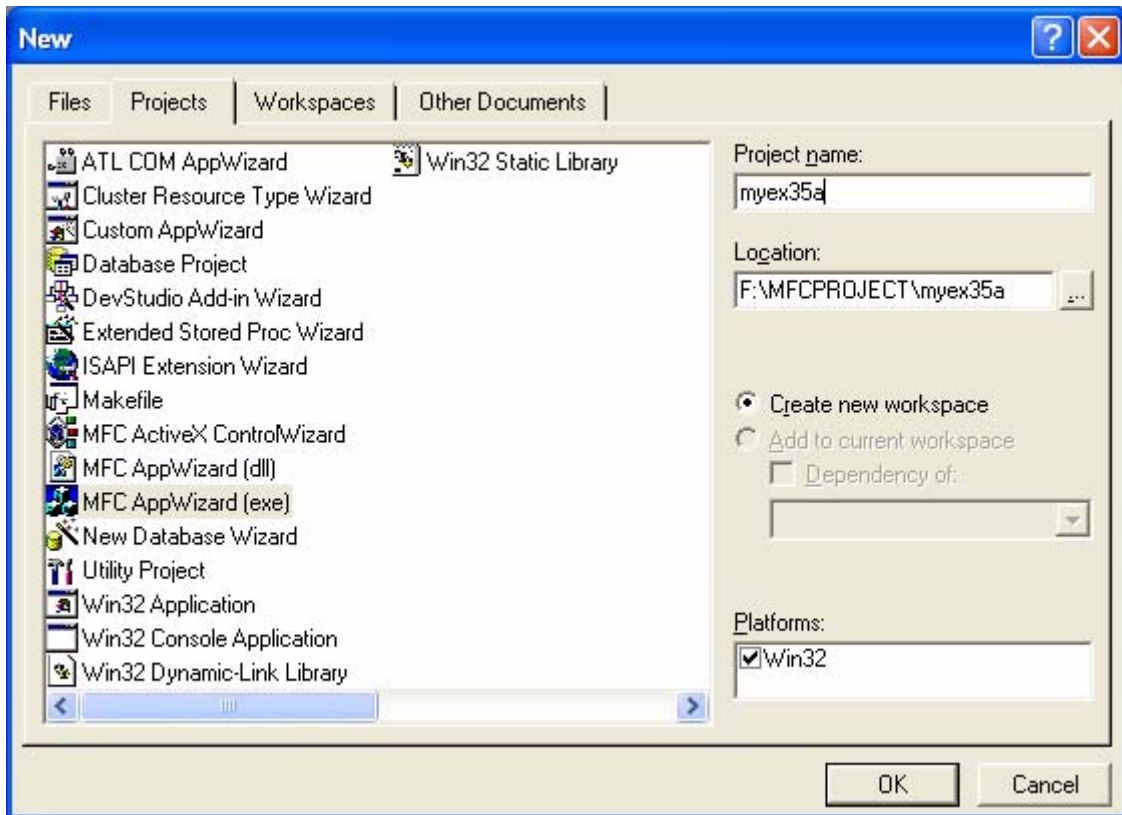


Figure 4: MYEX35A – Visual C++ new project dialog.

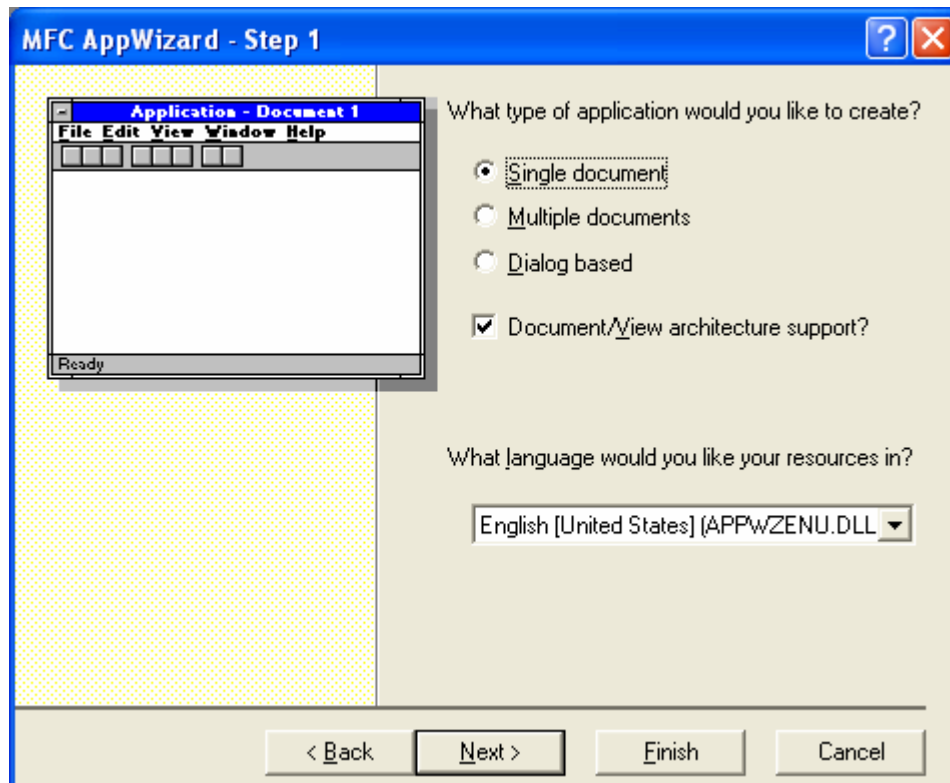


Figure 5: MYEX35A – AppWizard step 1 of 6, an SDI application.

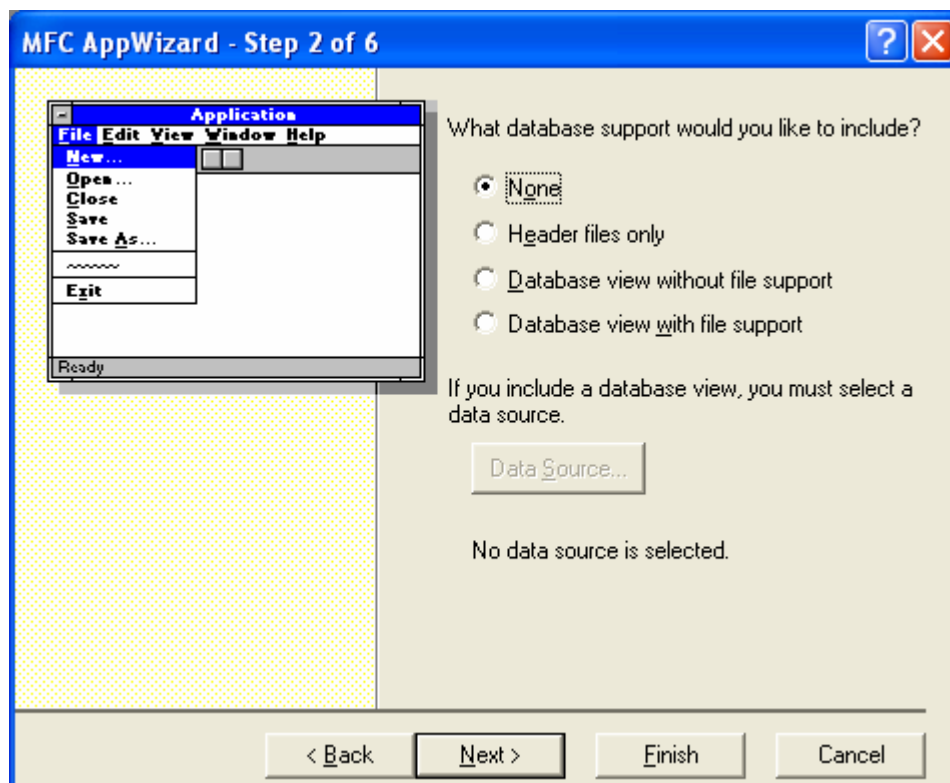


Figure 6: MYEX35A – AppWizard step 2 of 6.

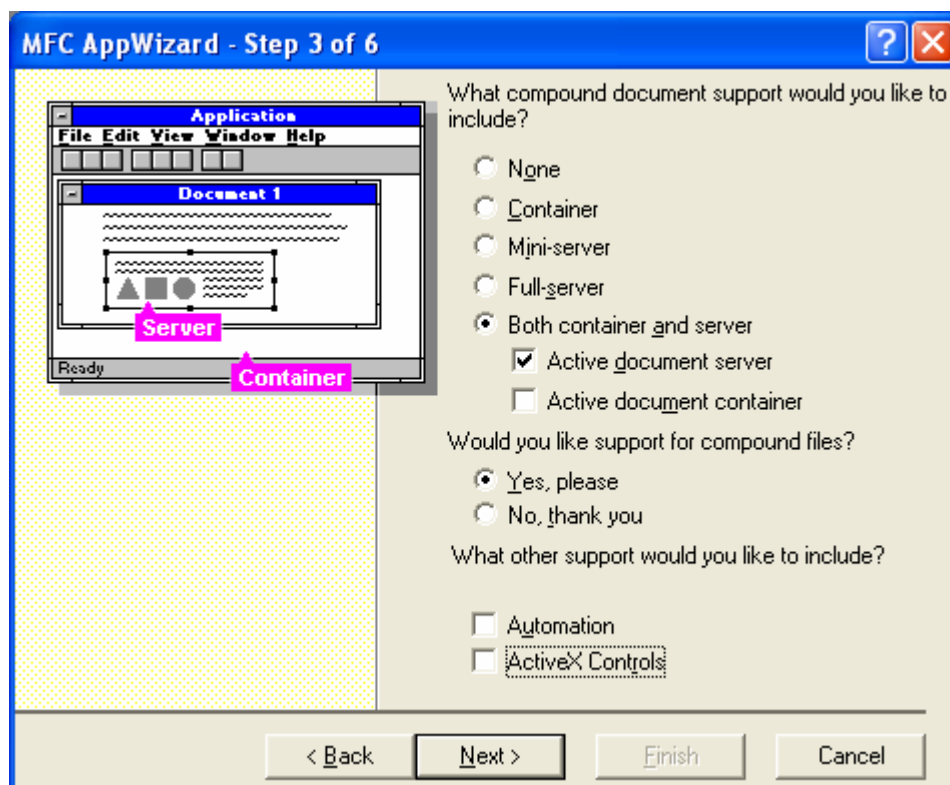


Figure 7: MYEX35A – AppWizard step 3 of 6, selecting the compound document, deselect the Automation and ActiveX Controls.

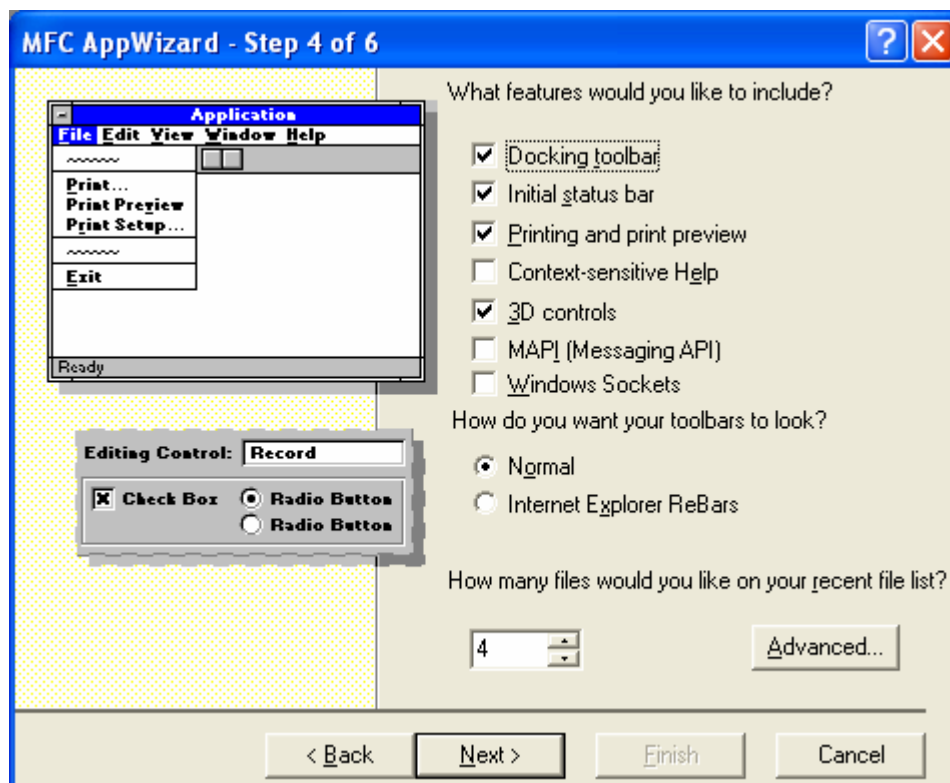


Figure 8: MYEX35A – AppWizard step 4 of 6.

Click the **Advanced** button. Set the **File Extension** as shown below.

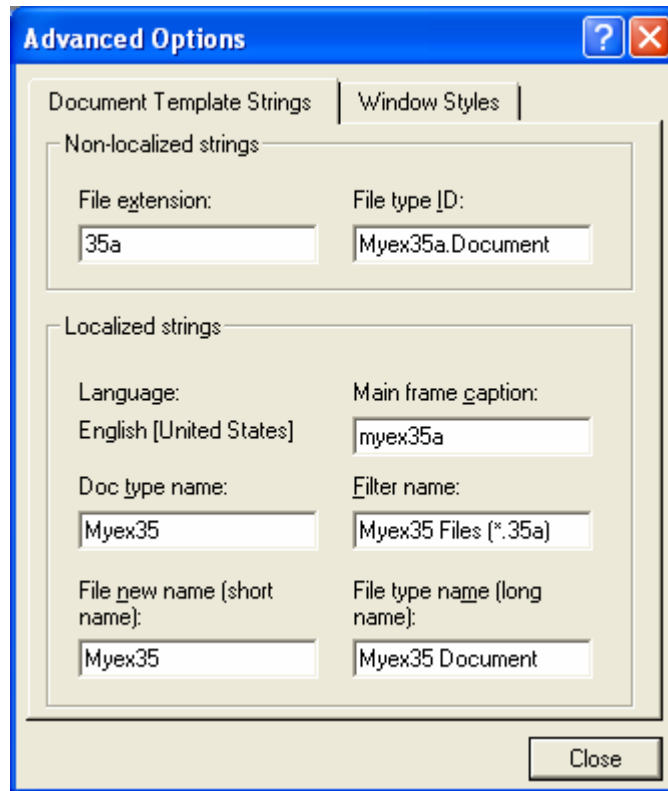


Figure 9: MYEX35A – Setting the file extension.

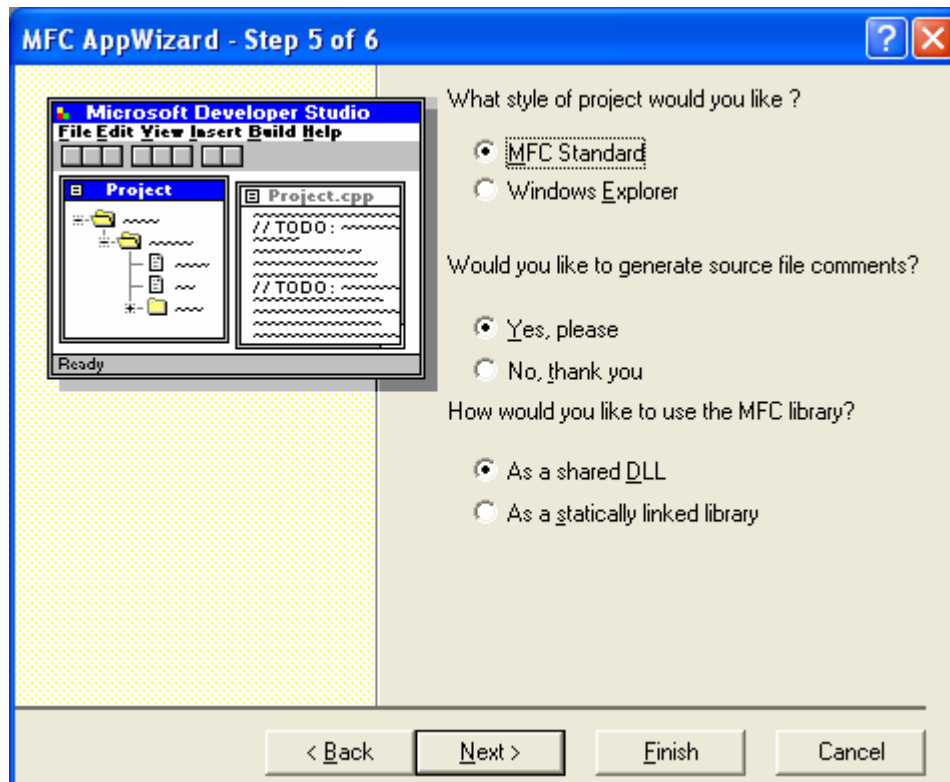


Figure 10: MYEX35A – AppWizard step 5 of 6.

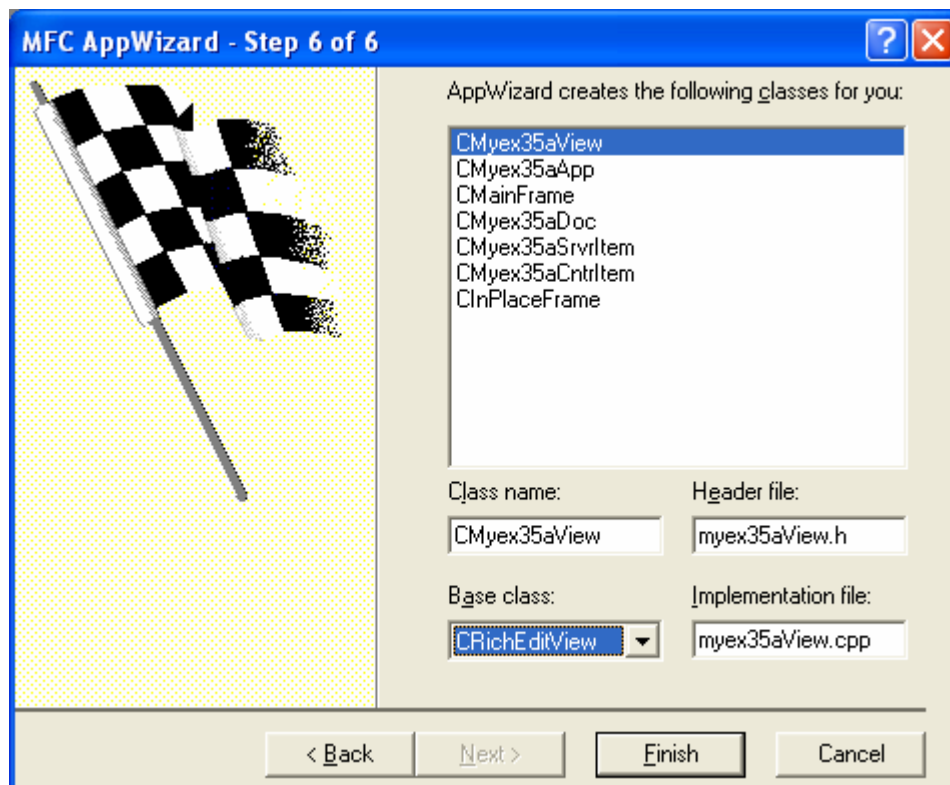


Figure 11: MYEX35A – AppWizard step 6 of 6, selecting CRichEditView as a base class.

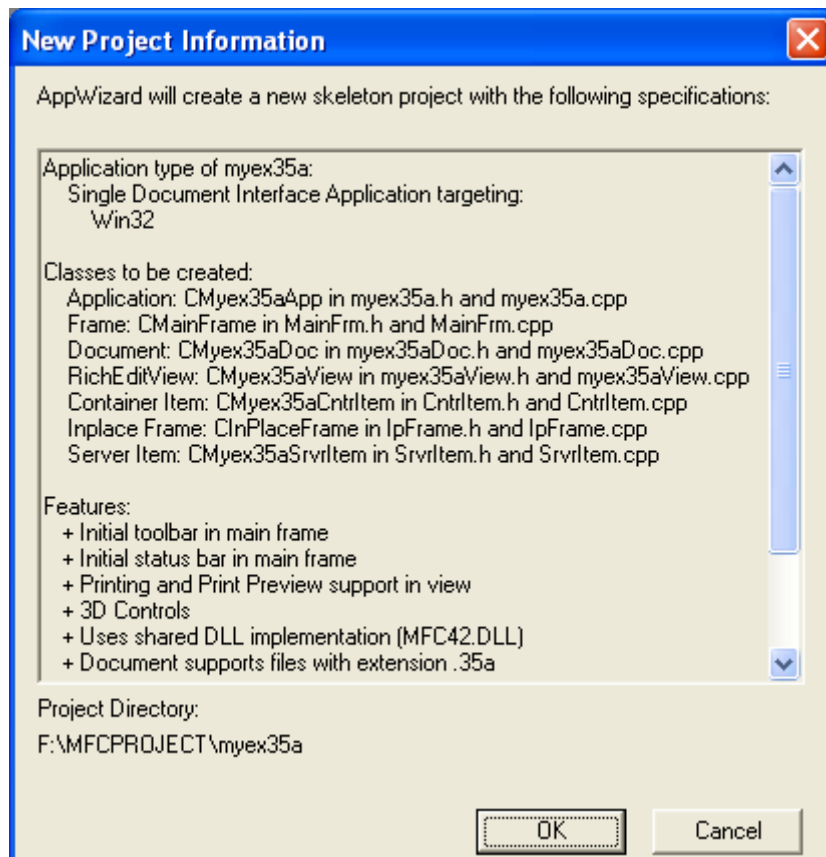


Figure 12: MYEX35A –project summary.

Build and run MYEX35A.

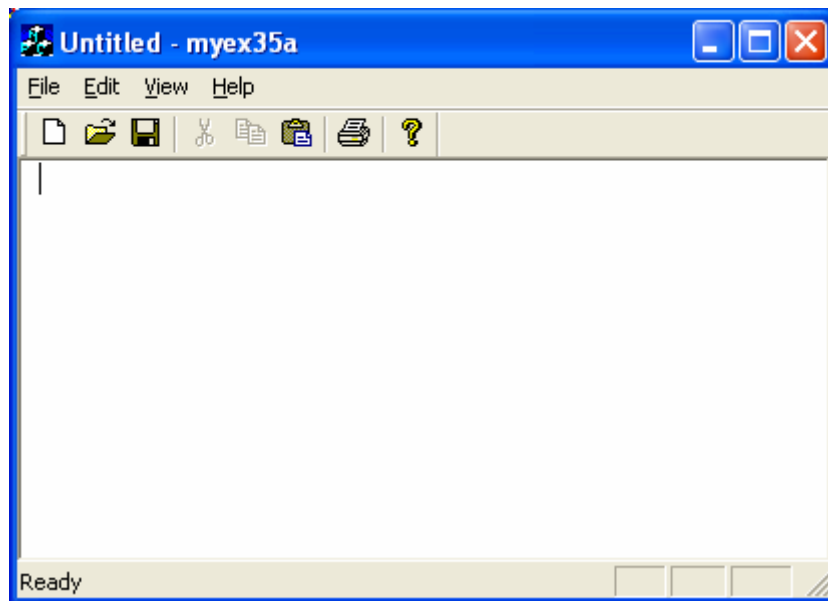


Figure 13: MYEX35A program output.

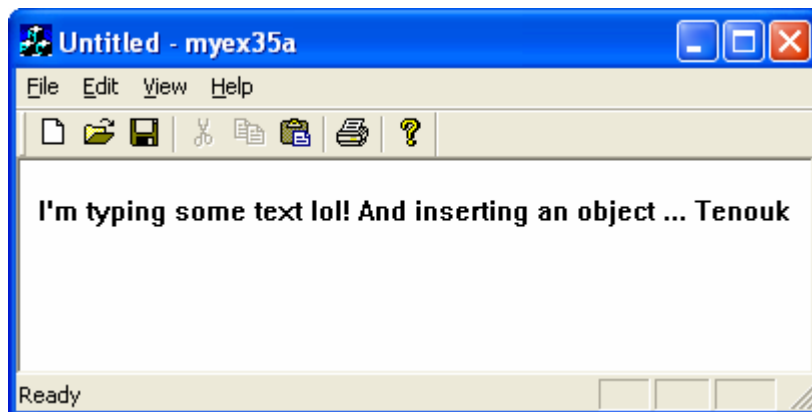


Figure 14: MYEX35A output – typing some texts.

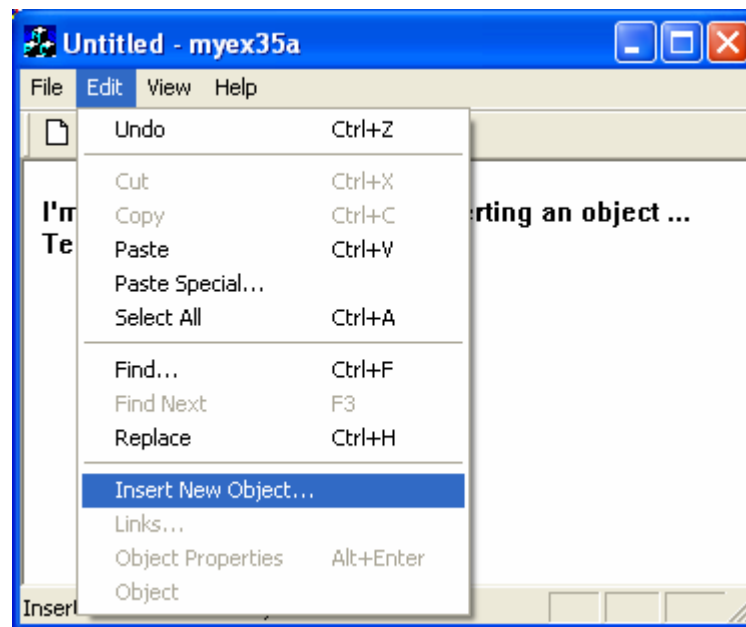


Figure 15: MYEX35A output – inserting an object.

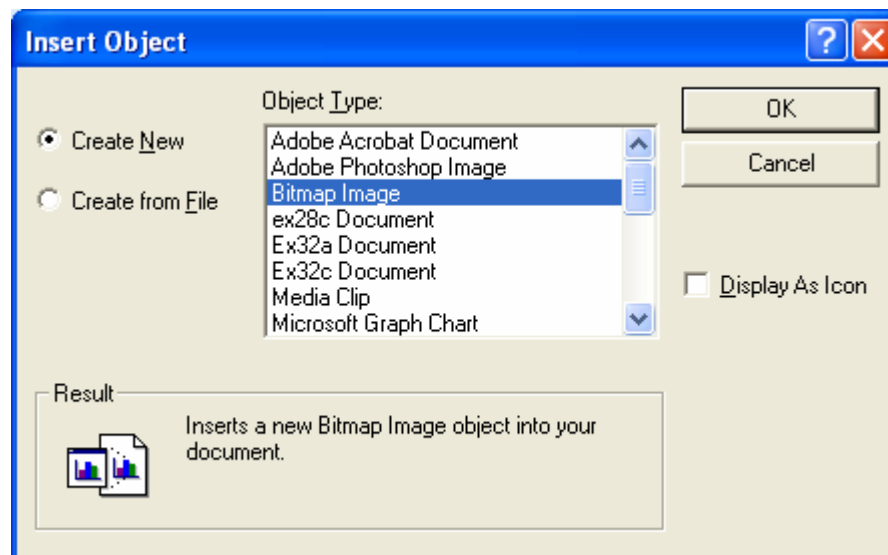


Figure 16: Inserting a bitmap object.

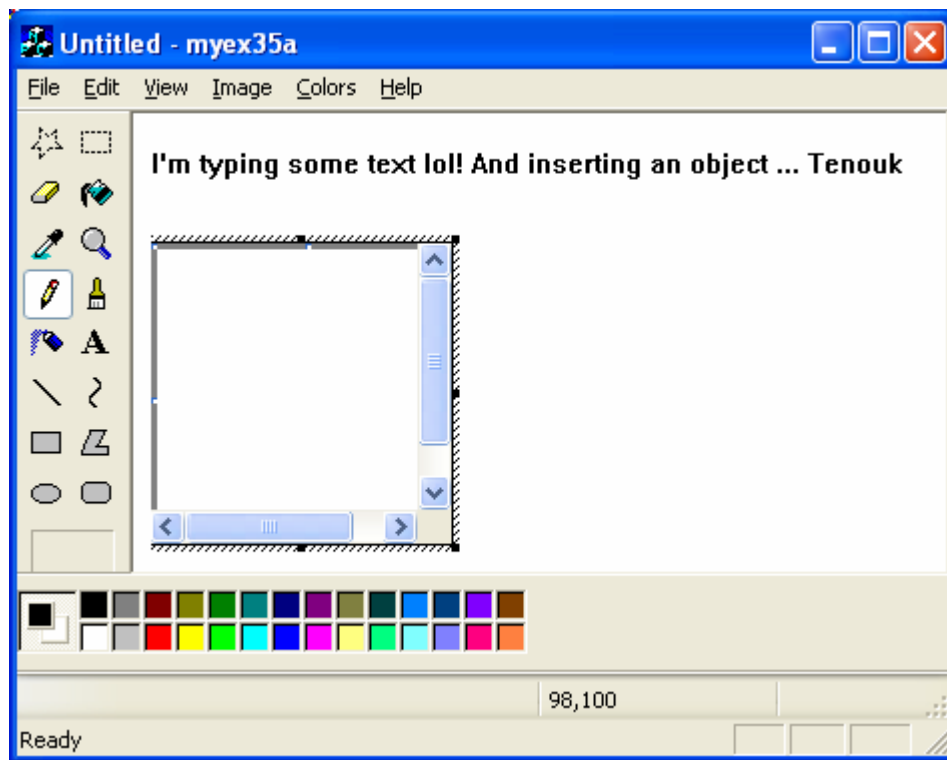


Figure 17: MYEX35A output – text and bitmap editor.

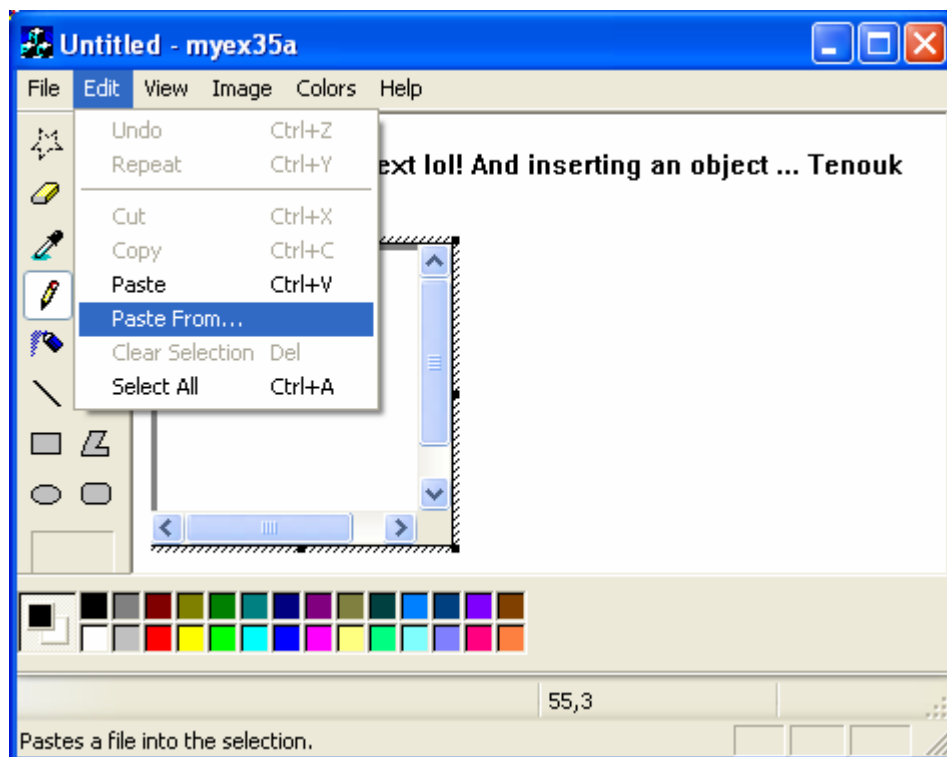


Figure 18: Inserting a bitmap to MYEX35A output.

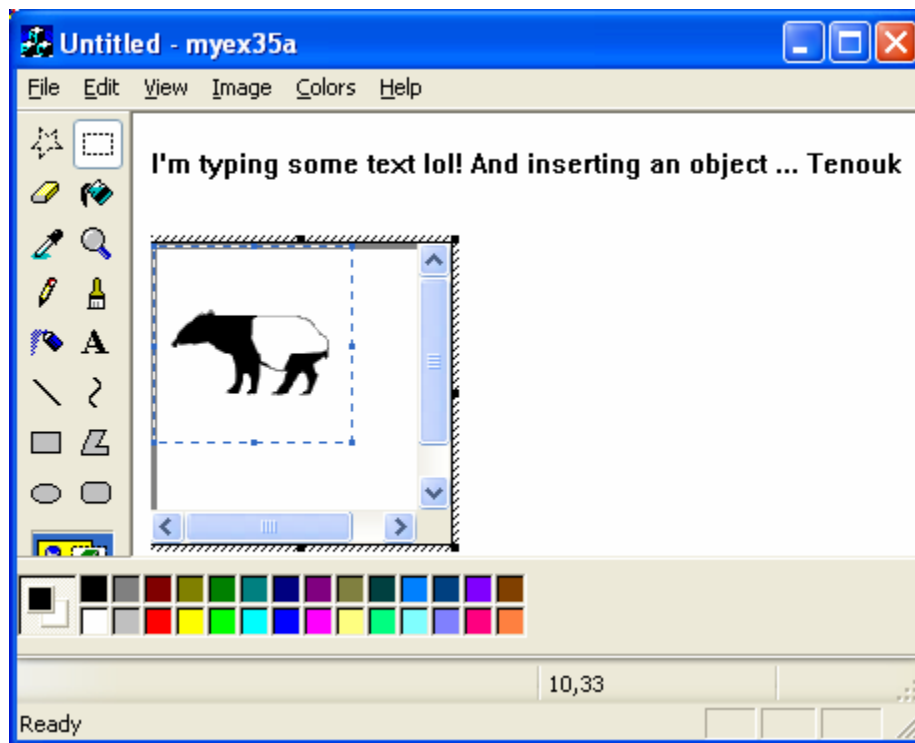


Figure 19: MYEX35A output with bitmap and text.

Then, save MYEX35A output. Click **File Save As** and save as **test.35a**.

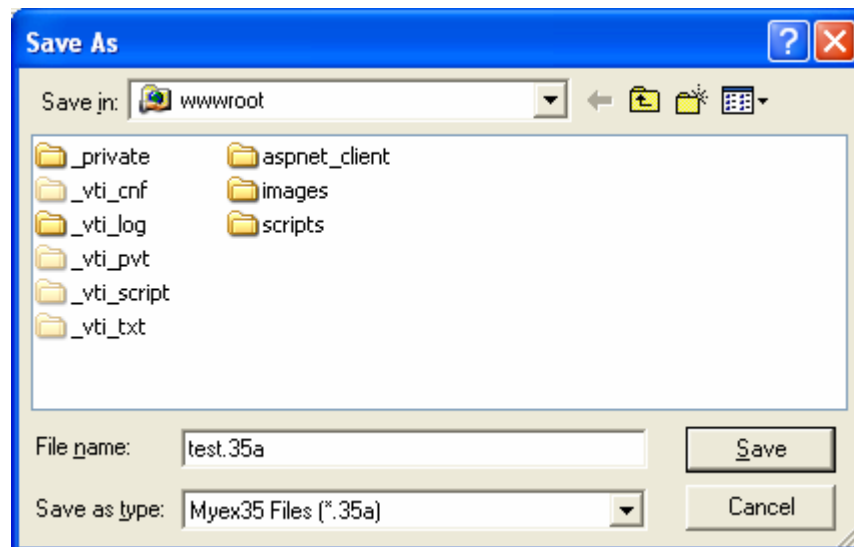


Figure 20: Saving MYEX35A output.

Try opening the saved file in Internet Explorer browser. You can also use **<http://localhost/test.35a>** to open the file.

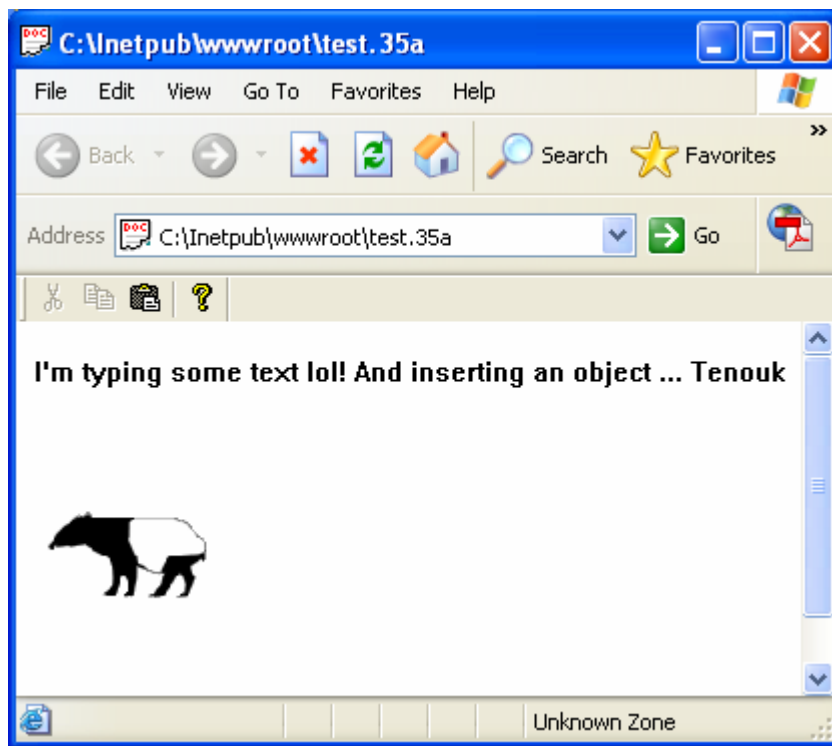


Figure 21: Opening MYEX35A output in browser.

## Debugging an ActiveX Document Server

If you want to debug your program in ActiveX document server mode, click on the **Debug** tab in the **Project Settings** dialog. Set **Program Arguments** to **/Embedding**, and then start the program. Now start the container program and use it to "start" the server, which has in fact already started in the debugger and is waiting for the container.

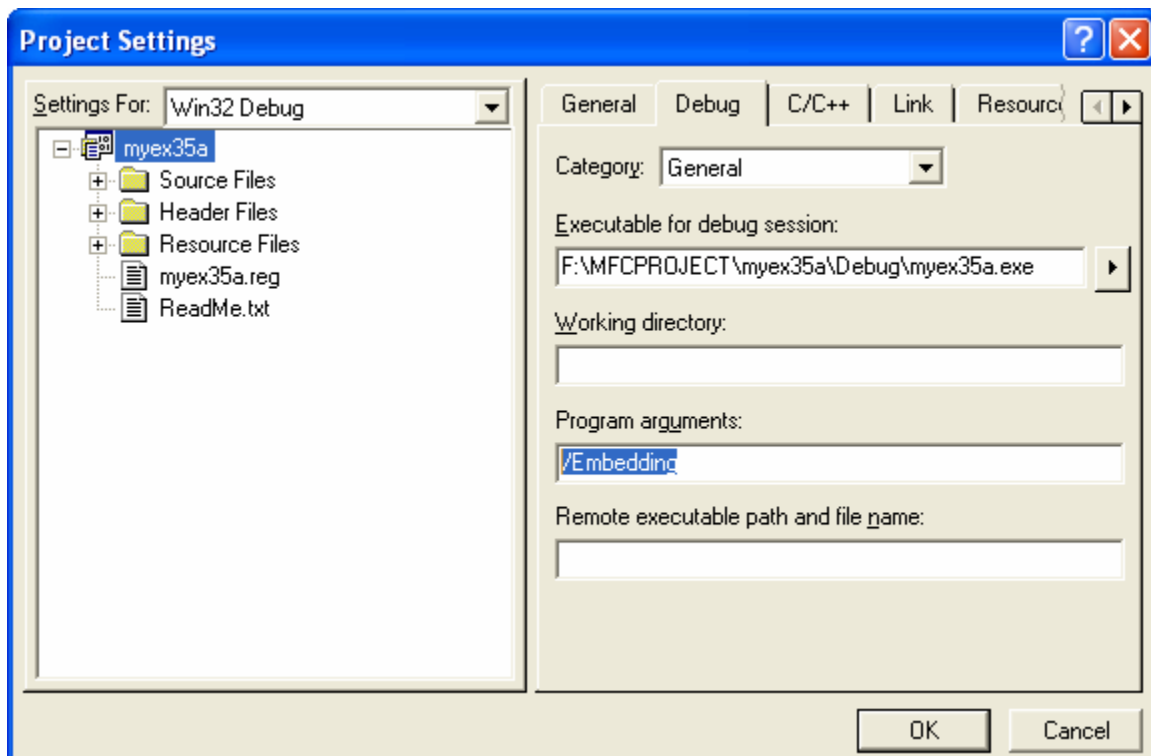


Figure 22: Changing the Visual C++ project setting for debugging process.

## MYEX35A: Phase 2 - Adding More Functionality

Let continue our task by adding more functionalities to MYEX35A. Add two bitmaps, with ID IDB\_GREEN and IDB\_RED for the start and stop button. Select and right click the root folder in ResourceView and choose **Insert** context menu. Select **Bitmap** and click **New**.

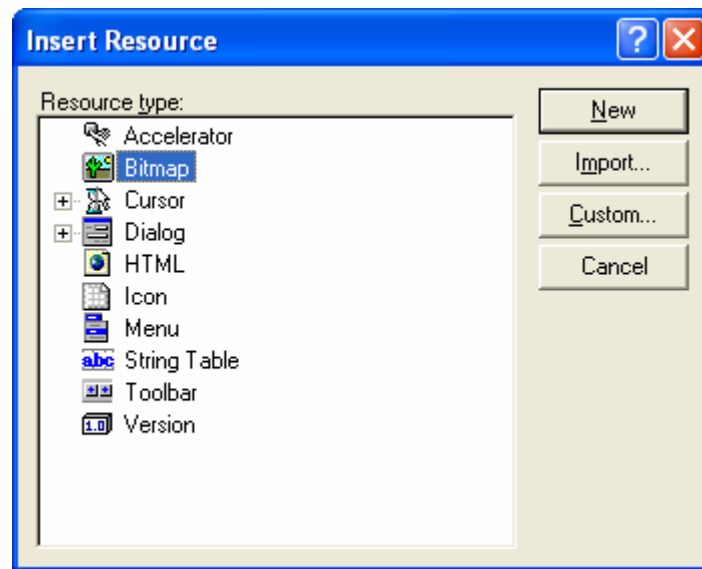


Figure 23: Inserting new bitmap.

Create two bitmaps as shown below.

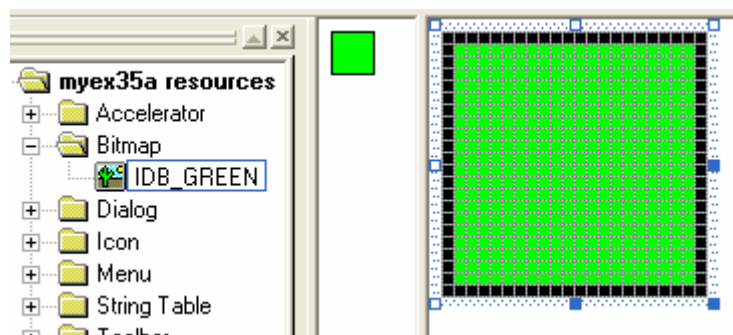


Figure 24: IDB\_GREEN bitmap for **Start** button.

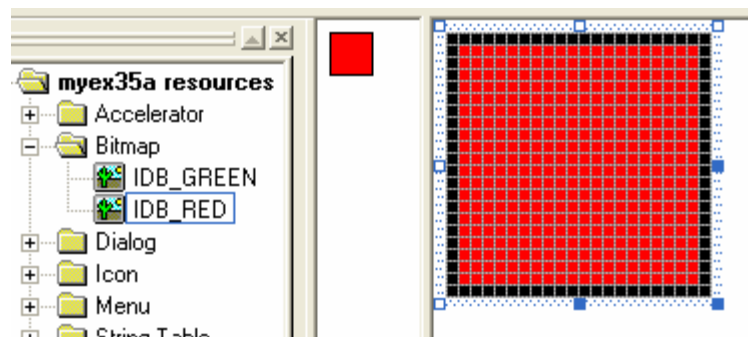


Figure 25: IDB\_RED bitmap for **Stop** button.

Add new dialog, IDD\_DIALOGBAR for dialog bar. Deselect the **Title bar**; choose **Child** for **Styles** and **None** for **Border** in **Styles** page as shown below.

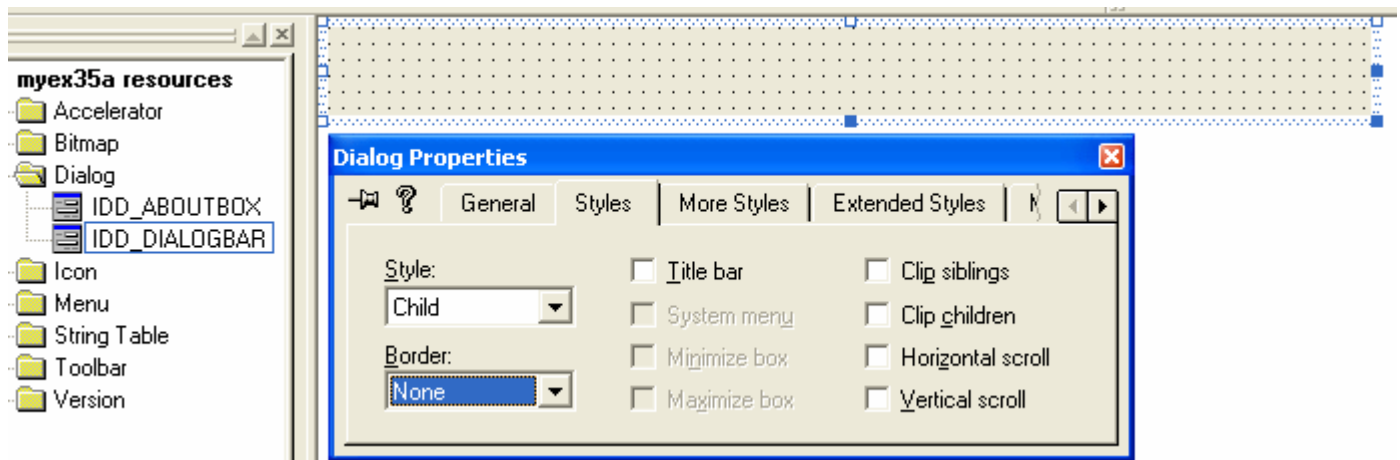


Figure 26: IDD\_DIALOGBAR property page.

Add Static text, Edit control and two buttons with the following properties. It is better for you to adjust the button size to fit the bitmap size later.

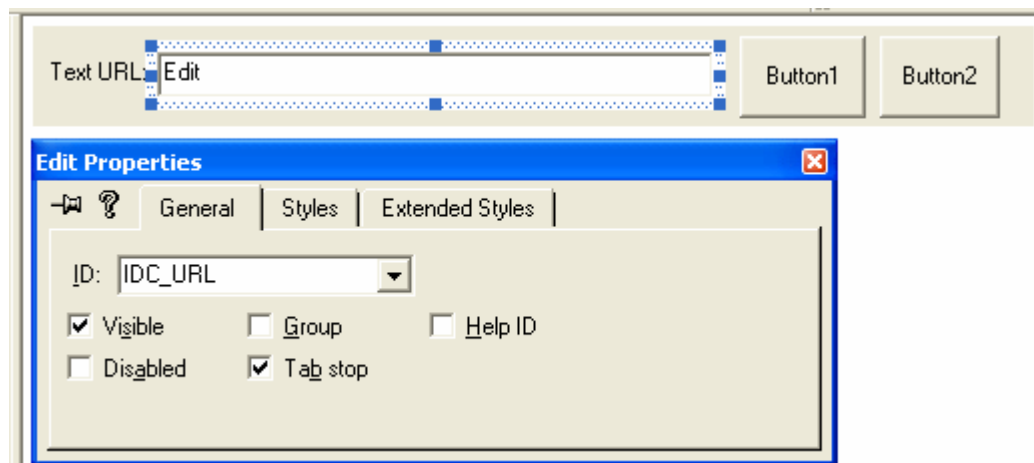


Figure 27: IDC\_URL Edit control property page.

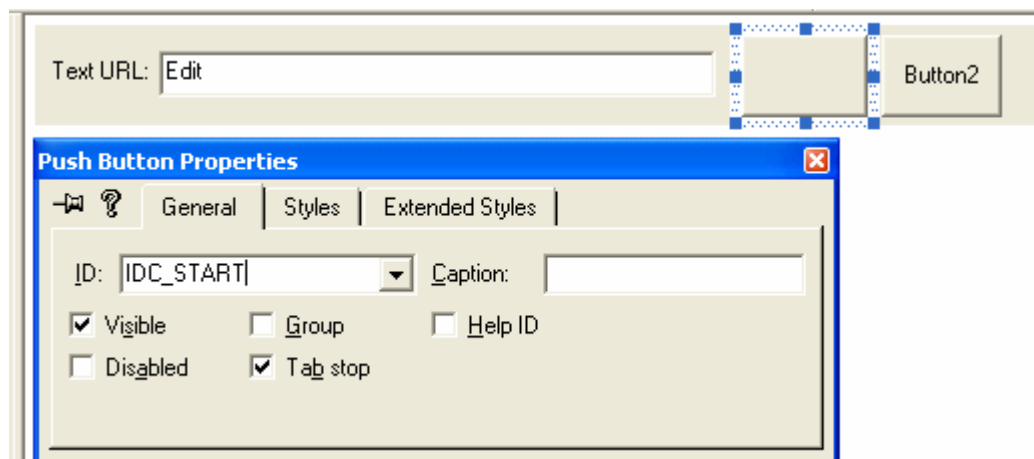


Figure 28: IDC\_START button property page.

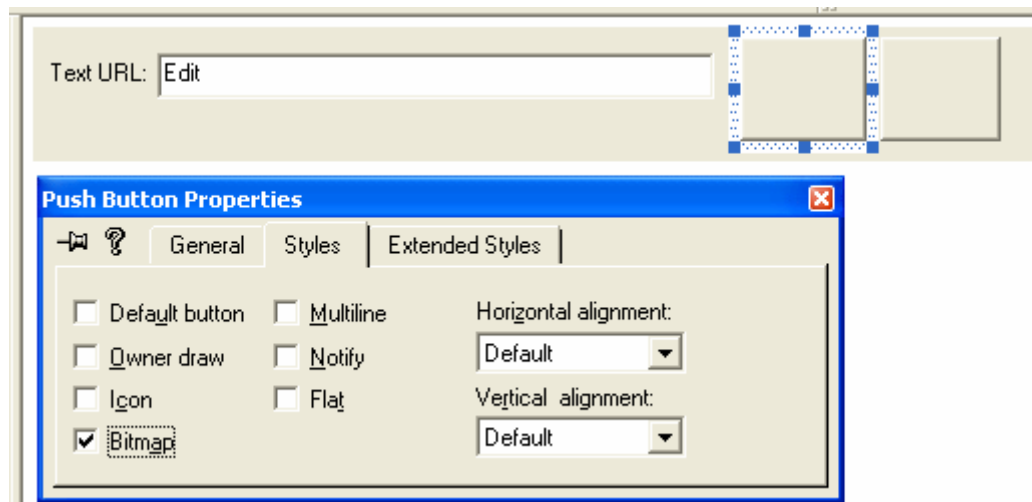


Figure 29: IDC\_START button, Styles property page.

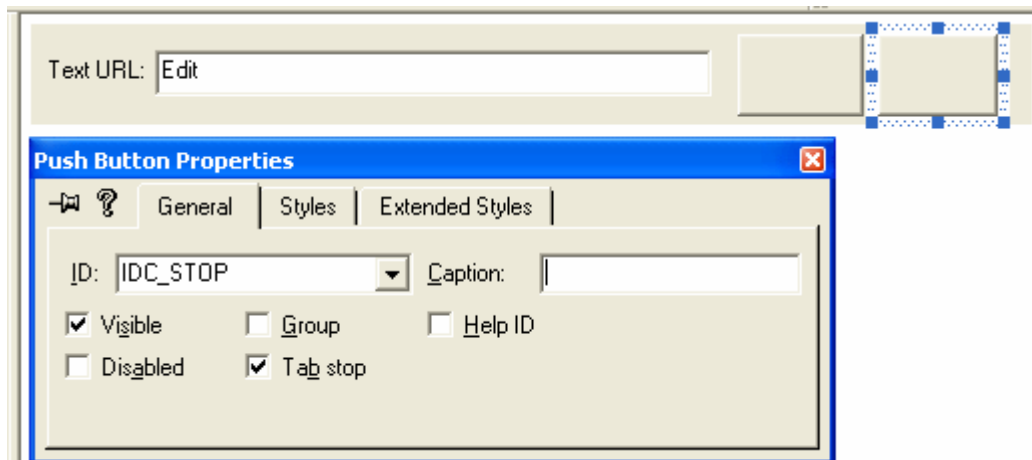


Figure 30: IDC\_STOP button property page.

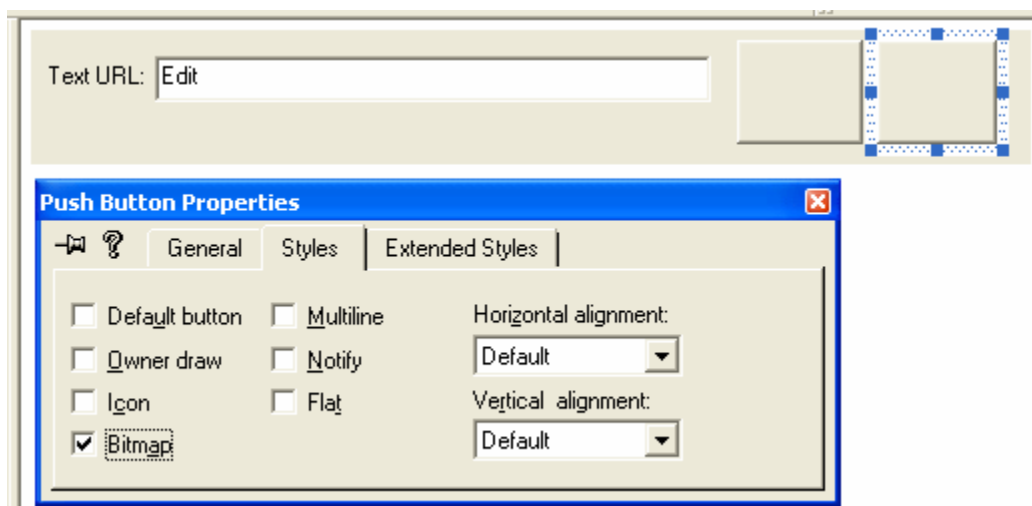


Figure 31: IDC\_STOP button, Styles property page.

Add context menu, IDR\_MENUCONTEXT as shown below.

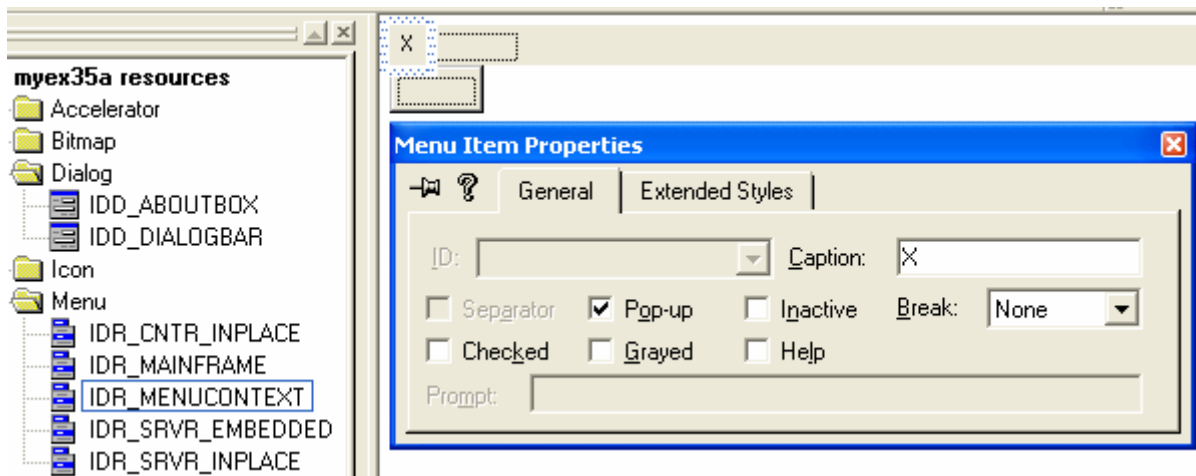


Figure 32: IDR\_MENUCONTEXT, new context menu.

Then add the menu item, **Clear All** with ID ID\_EDIT\_CLEAR\_ALL.

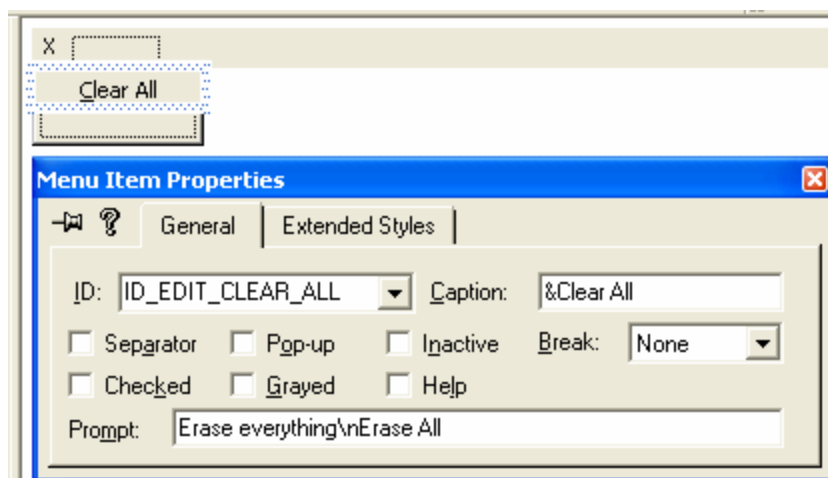


Figure 33: Adding menu item **Clear All** to the context menu.

## The Coding Part

Add new header and source files named **Urlthread.h** and **Urlthread.cpp** respectively.

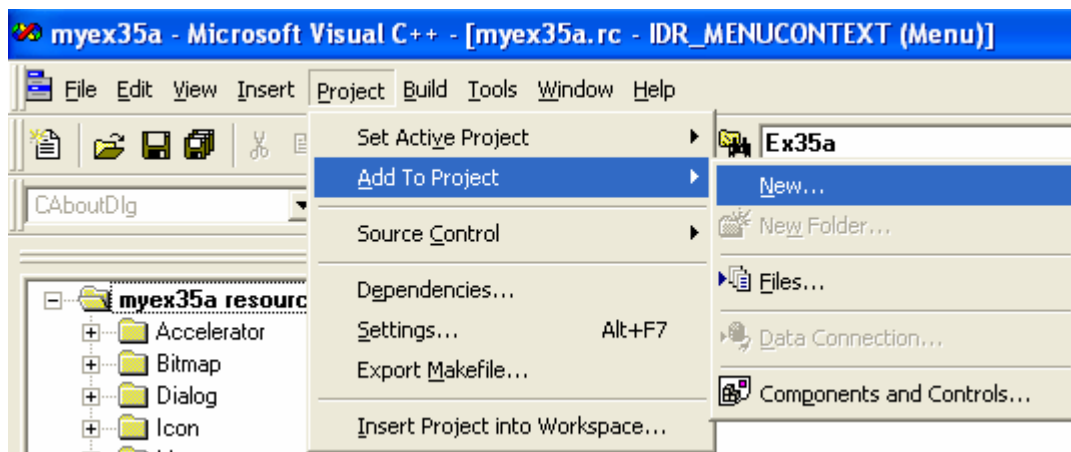


Figure 34: Adding new files to project.

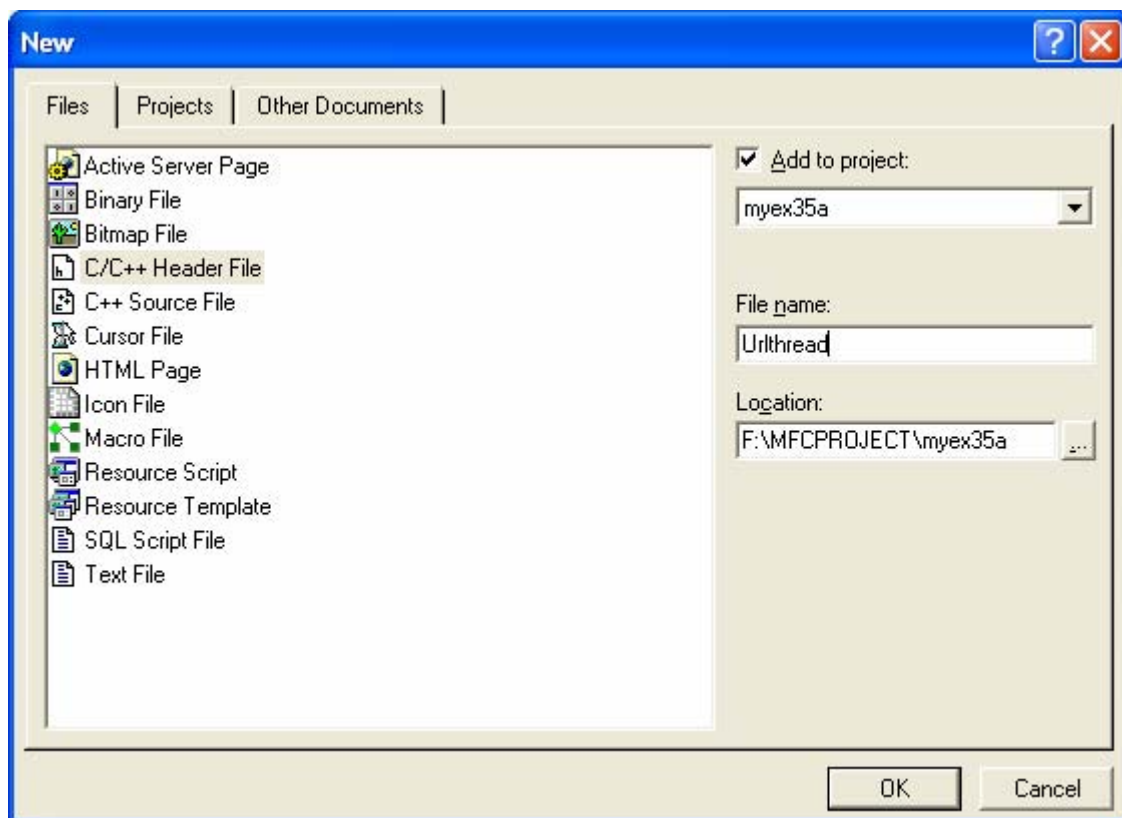


Figure 35: Selecting and entering the new file name.

Copy the codes as shown in the following Listing.

#### URLTHREAD.H

```
// urlthread.h

extern CString g_strURL;
extern volatile BOOL g_bThreadStarted;
extern CEvent g_eKill;

extern UINT UrlThreadProc(LPVOID pParam);
extern void LogInternetException(LPVOID pParam, CInternetException* pe);
```

## URLTHREAD.CPP

```
// urlthread.cpp

#include <stdafx.h>
#include "urlthread.h"
#define MAXBUF 100

CString g_strURL = "http://";
volatile BOOL g_bThreadStarted = FALSE;
CEvent g_eKill;

UINT UrlThreadProc(LPVOID pParam)
{
    g_bThreadStarted = TRUE;
    CString strLine;
    CInternetSession session;
    CStdioFile* pFile1 = NULL;

    Try
    {
        pFile1 = session.OpenURL(g_strURL, 0, INTERNET_FLAG_TRANSFER_BINARY |
INTERNET_FLAG_KEEP_CONNECTION); // needed for Windows NT c/r authentication
        // Keep displaying text from the URL until the Kill event is received
        while(::WaitForSingleObject(g_eKill.m_hObject, 0) != WAIT_OBJECT_0) {
            // one line at a time
            if(pFile1->ReadString(strLine) == FALSE) break;
            strLine += '\n';
            ::SendMessage((HWND) pParam, EM_SETSEL, (WPARAM) 999999, 1000000);
            ::SendMessage((HWND) pParam, EM_REPLACESEL, (WPARAM) 0,
                (LPARAM) (const char*) strLine);
            Sleep(250); // Deliberately slow down the transfer
        }
    }
    catch(CInternetException* e) {
        LogInternetException(pParam, e);
        e->Delete();
    }
    if(pFile1 != NULL) delete pFile1; // closes the file -- prints a warning
    g_bThreadStarted = FALSE;
    // Post any message to update the toolbar buttons
    ::PostMessage((HWND) pParam, EM_SETSEL, (WPARAM) 999999, 1000000);
    TRACE("Post thread exiting normally\n");
    return 0;
}

void LogInternetException(LPVOID pParam, CInternetException* pe)
{
    CString strGmt = CTime::GetCurrentTime().FormatGmt("%m/%d/%y %H:%M:% GMT");
    char text1[300], text2[100];
    wsprintf(text1, "\r\nERROR: WinInet error #%d -- %s\r\n",
        pe->m_dwError, (const char*) strGmt);
    pe->GetErrorMessage(text2, 99);
    strcat(text1, text2);
    if(pe->m_dwError == 12152) {
        strcat(text1, " URL not found?\r\n");
    }
    ::SendMessage((HWND) pParam, EM_SETSEL, (WPARAM) 999999, 1000000);
    ::SendMessage((HWND) pParam, EM_REPLACESEL, (WPARAM) 0, (LPARAM) text1);
}
```

Listing 1.

Add the following `#include` in `StdAfx.h`.

```

#include <afxinet.h>
#include <afxmt.h>

#ifdef _AFX_NO_AFXCMN_SUPPORT

#include <afxrich.h>          // MFC rich edit classes
#include <afxinet.h>
#include <afxmt.h>
//{{AFX_INSERT_LOCATION}}

```

Listing 1.

## CMainFrame Class

Add the following public member variables.

```

CBitmap      m_bitmapGreen;
CBitmap      m_bitmapRed;

```

Then, change the:

```

CToolBar      m_wndToolBar;

```

variable to

```

CDialogBar    m_wndDialogBar;

```

And change the class access type as shown below. We need to access the public variable later.

```

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
public:
    CDialogBar m_wndDialogBar;
    CBitmap    m_bitmapGreen;
    CBitmap    m_bitmapRed;

    // Generated message map functions

```

Add the following `#include` directive to **MainFrm.cpp**.

```

#include "urlthread.h"

#include "stdafx.h"
#include "myex35a.h"

#include "urlthread.h"
#include "MainFrm.h"

#ifdef _DEBUG

```

Listing 2.

Change the `OnCreate()` as shown below.

```

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

```

```

if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return -1;        // fail to create
}

if (!m_wndDialogBar.Create(this, IDD_DIALOGBAR, CBRS_TOP, 0xE810))
{
    TRACE0("Failed to create dialog bar\n");
    return -1;        // fail to create
}
m_wndDialogBar.SetDlgItemText(IDC_URL, g_strURL);
// attach the color bitmaps to the dialog bar buttons
m_bitmapGreen.LoadBitmap(IDB_GREEN);
HBITMAP hBitmap = (HBITMAP) m_bitmapGreen.GetSafeHandle();
((CButton*) m_wndDialogBar.GetDlgItem(IDC_START))->SetBitmap(hBitmap);
m_bitmapRed.LoadBitmap(IDB_RED);
hBitmap = (HBITMAP) m_bitmapRed.GetSafeHandle();
((CButton*) m_wndDialogBar.GetDlgItem(IDC_STOP))->SetBitmap(hBitmap);

return 0;
}

```

And PreCreateWindow() as shown below.

```

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    return CFrameWnd::PreCreateWindow(cs);
}

```

### **CMyex35aApp Class**

Using ClassWizard, add ExitInstance() virtual function to CMyex35aApp class.

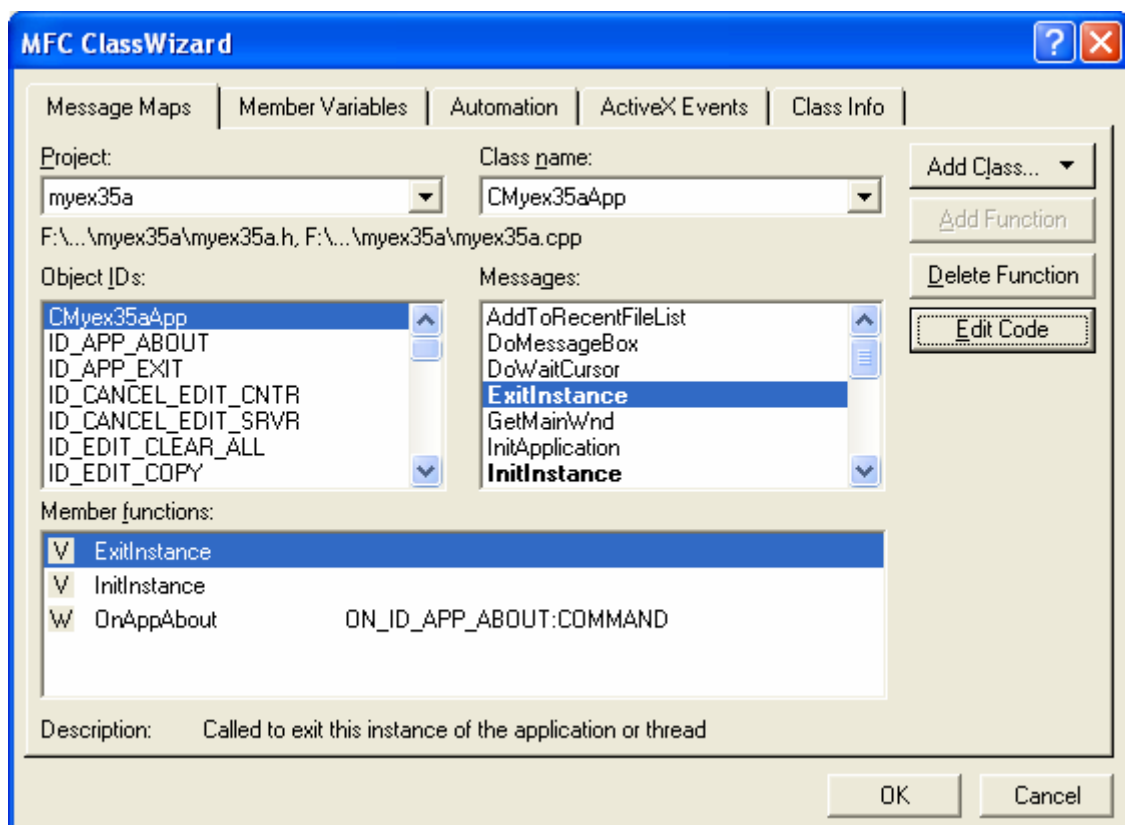


Figure 36: Adding `ExitInstance()` virtual function.

Click the **Edit Code** button and modify the code as shown below.

```
int CMyex35aApp::ExitInstance()
{
    // TODO: Add your specialized code here and/or call the base class
    g_eKill.SetEvent();
    Sleep(500);
    return CWinApp::ExitInstance();
}

// CMyex35aApp message handlers
int CMyex35aApp::ExitInstance()
{
    // TODO: Add your specialized code here and/or call the base class
    g_eKill.SetEvent();
    Sleep(500);
    return CWinApp::ExitInstance();
}
```

Listing 3.

Add the following `#include` directive to **myex35a.cpp**.

```
#include "urlthread.h"
```

```
#include "stdafx.h"
#include "myex35a.h"

#include "urlthread.h"
#include "MainFrm.h"
```

Add the following code in `InitInstance()`.

```
AfxEnableControlContainer();
```

```
BOOL CMyex35aApp::InitInstance()
{
    // Initialize OLE libraries
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    int __cdecl AfxMessageBox(unsigned short *, unsigned int, unsigned int)
    AfxEnableControlContainer();
    // Standard initialization
```

Listing 4.

Delete the following lines of code in `InitInstance()`.

```
// The one and only window has been initialized, so show and update it.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
```

Remove the following code of the `OnDraw()` in **SrvrItem.cpp**.

```
// Remove this if you use rSize
UNREFERENCED_PARAMETER(rSize);
```

Add variables, delete and change the class access type as shown below in **lpFrame.h**.

```
protected:
    COleResizeBar    m_wndResizeBar;
    COleDropTarget   m_dropTarget;
public:
    CDialogBar       m_wndDialogBar;
    CBitmap          m_bitmapGreen;
    CBitmap          m_bitmapRed;
    // Generated message map functions
```

Listing 5.

Add the `#include "urlthread.h"` directive and modify the `OnCreateControlBars()` in **lpFrame.cpp** as shown below.

```
BOOL CInPlaceFrame::OnCreateControlBars(CFrameWnd* pWndFrame, CFrameWnd*
pWndDoc)
{
    m_wndDialogBar.SetOwner(this); // for commands
    if (!m_wndDialogBar.Create(pWndFrame, IDD_DIALOGBAR, CBRS_TOP, 0xE810))
    {
        TRACE0("Failed to create dialog bar\n");
    }
```

```

        return -1;        // fail to create
    }
    m_wndDialogBar.SetDlgItemText(IDC_URL, g_strURL);
    // attach the color bitmaps to the dialog bar buttons
    m_bitmapGreen.LoadBitmap(IDB_GREEN);
    HBITMAP hBitmap = (HBITMAP) m_bitmapGreen.GetSafeHandle();
    ((CButton*) m_wndDialogBar.GetDlgItem(IDC_START))->SetBitmap(hBitmap);
    m_bitmapRed.LoadBitmap(IDB_RED);
    hBitmap = (HBITMAP) m_bitmapRed.GetSafeHandle();
    ((CButton*) m_wndDialogBar.GetDlgItem(IDC_STOP))->SetBitmap(hBitmap);

    return TRUE;
}

// application can place MFC control bars on either window.
BOOL CInPlaceFrame::OnCreateControlBars(CFrameWnd* pWndFrame,
                                         CFrameWnd* pWndDoc)
{
    m_wndDialogBar.SetOwner(this); // for commands
    if (!m_wndDialogBar.Create(pWndFrame, IDD_DIALOGBAR, CBRS_TOP, 0xE810))
    {
        TRACE0("Failed to create dialog bar\n");
        return -1;        // fail to create
    }
    m_wndDialogBar.SetDlgItemText(IDC_URL, g_strURL);
    // attach the color bitmaps to the dialog bar buttons
    m_bitmapGreen.LoadBitmap(IDB_GREEN);
    HBITMAP hBitmap = (HBITMAP) m_bitmapGreen.GetSafeHandle();
    ((CButton*) m_wndDialogBar.GetDlgItem(IDC_START))->SetBitmap(hBitmap);
    m_bitmapRed.LoadBitmap(IDB_RED);
    hBitmap = (HBITMAP) m_bitmapRed.GetSafeHandle();
    ((CButton*) m_wndDialogBar.GetDlgItem(IDC_STOP))->SetBitmap(hBitmap);

    return TRUE;
}

```

Listing 6.

### CMyex35aDoc Class

Change the GetDocObjectServer() virtual function to public in **myex35aDoc.h**.

```

#endif
public:
    virtual CDocObjectServer* GetDocObjectServer(LPOLEDOCUMENTSITE pDocSite);
    // Generated message map functions

```

Listing 7.

### CMyex35aView Class

Using ClassWizard delete OnDestroy(). Go to ResourceView, select the IDD\_DIALOGBAR, right click and select the ClassWizard context menu.

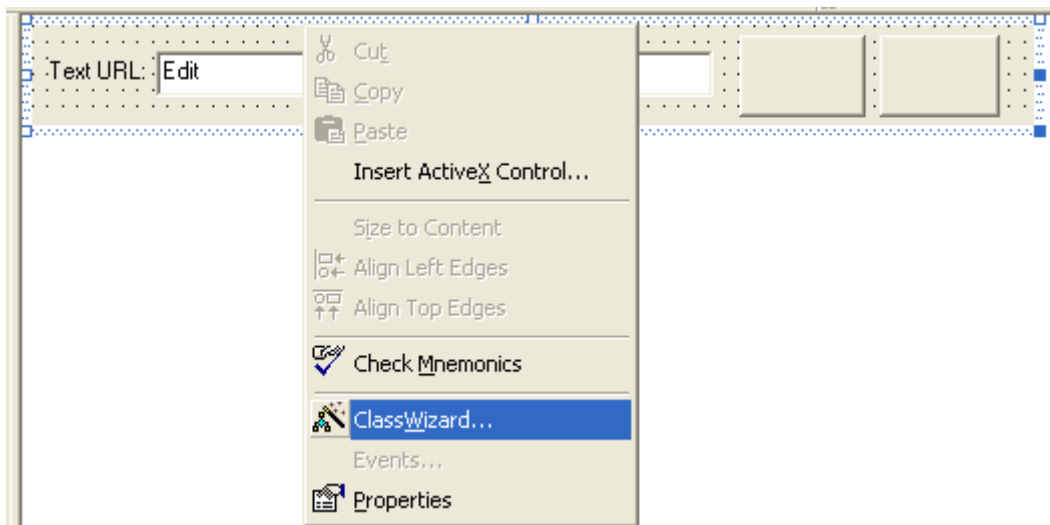


Figure 37: Invoking ClassWizard from dialog editor.

For the **Adding a Class** dialog prompt, choose **Select an existing class**.

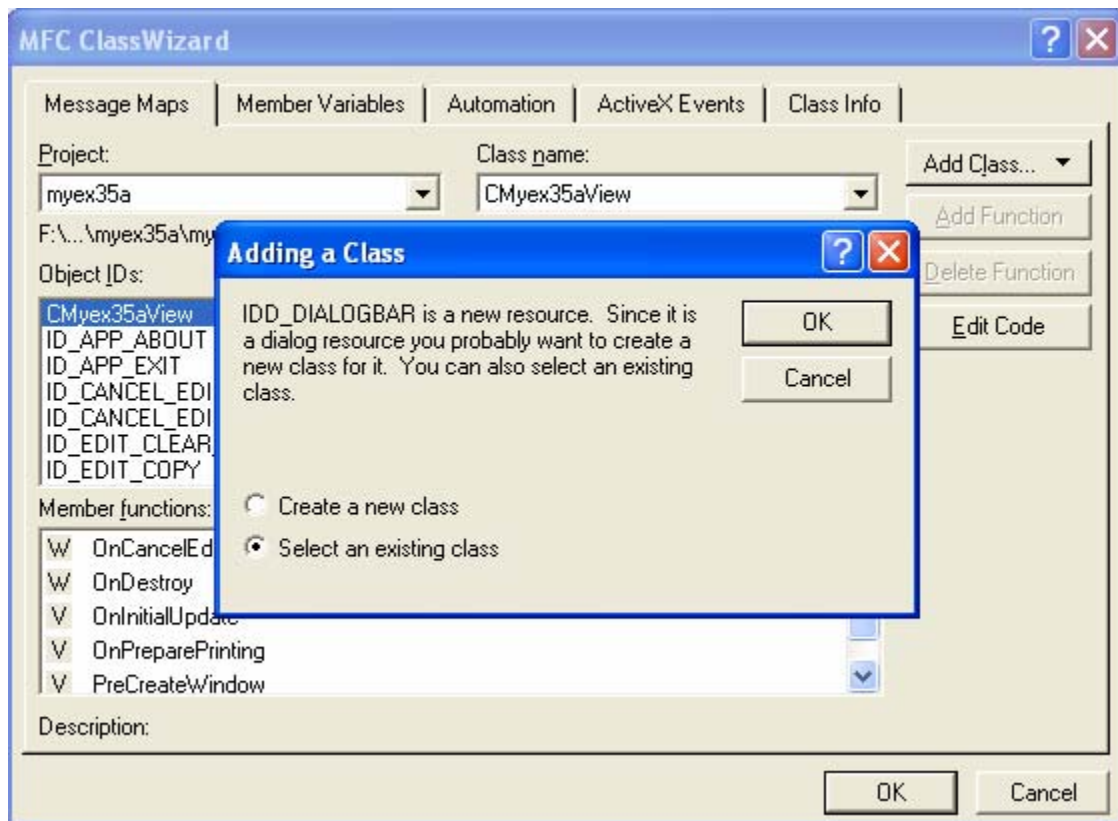


Figure 38: Adding dialog to an existing class.

Select the CMyx35aView class.

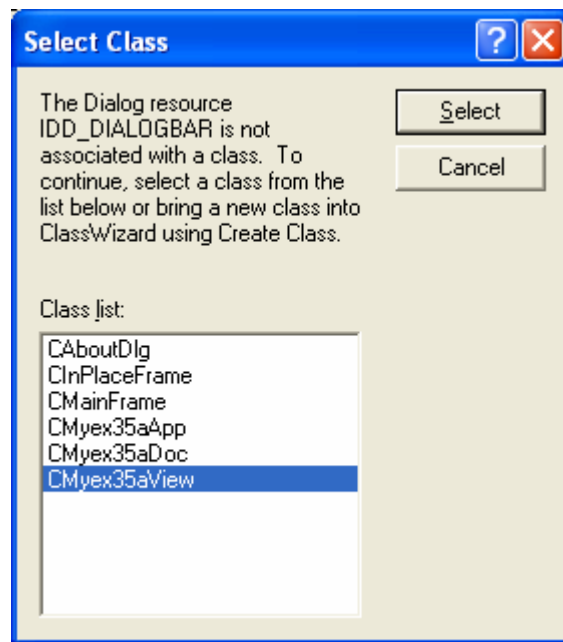


Figure 39: Selecting an existing class.

Just click **Yes** for the following dialog prompt.



Figure 40: Dialog prompt for the class to dialog linking.

Add the following message map command and update command handler functions.

ID	Command
ID_EDIT_CLEAR_ALL	Command
IDC_START	Command and update command
IDC_STOP	Command and update command

Table 2.

You may add those commands and update commands manually if needed.

```
// Generated message map functions
protected:
   //{{AFX_MSG(CMyex35aView)
    afx_msg void OnCancelEditSrvr();
    afx_msg void OnEditClearAll();
    afx_msg void OnStart();
    afx_msg void OnStop();
   //}}AFX_MSG
    afx_msg void OnUpdateStop(CCmdUI* pCmdUI);
    afx_msg void OnUpdateStart(CCmdUI* pCmdUI);
    DECLARE_MESSAGE_MAP()
};
```

Listing 8.

Add the following #include directives to myex35aView.cpp.

```
#include "Urlthread.h"
#include "MainFrm.h"
#include "IpFrame.h"

#include "myex35aDoc.h"
#include "CntrlItem.h"
#include "myex35aView.h"
#include "Urlthread.h"
#include "MainFrm.h"
#include "IpFrame.h"

#ifdef _DEBUG
```

Listing 9.

If you add some of the previous handlers manually make sure message map in **myex35aView.cpp** as shown below.

```
BEGIN_MESSAGE_MAP(CMyex35aView, CRichEditView)
//{{AFX_MSG_MAP(CMyex35aView)
ON_WM_DESTROY()
ON_COMMAND(ID_CANCEL_EDIT_SRVR, OnCancelEditSrvr)
ON_COMMAND(ID_EDIT_CLEAR_ALL, OnEditClearAll)
ON_COMMAND(IDC_START, OnStart)
ON_UPDATE_COMMAND_UI(IDC_START, OnUpdateStart)
ON_COMMAND(IDC_STOP, OnStop)
ON_UPDATE_COMMAND_UI(IDC_STOP, OnUpdateStop)
//}}AFX_MSG_MAP
// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CRichEditView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, CRichEditView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CRichEditView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

Listing 10.

Don't forget to delete the OnDestroy() implementation and complete the message handler implementations as shown below.

```
void CMyex35aView::OnEditClearAll()
{
    // Not so useable here, coz of the save as...
    SetWindowText("");
}

void CMyex35aView::OnStart()
{
    CWnd* pFrm = GetParent(); // SDI only
    CWnd* pBar;
    if(pFrm->IsKindOf(RUNTIME_CLASS(CMainFrame))) {
        pBar = &((CMainFrame*) pFrm)->m_wndDialogBar;
    }
    else {
        pBar = &((CInPlaceFrame*) pFrm)->m_wndDialogBar;
    }
}
```

```

        // g_strURL: thread sync?
        pBar->GetDlgItemText(IDC_URL, g_strURL);
        TRACE("CMyex35aView::OnRequest -- URL = %s\n", g_strURL);
        AfxBeginThread(UrlThreadProc, GetSafeHwnd(), THREAD_PRIORITY_NORMAL);
    }

void CMyex35aView::OnStop()
{
    g_eKill.SetEvent();
}

void CMyex35aView::OnUpdateStop(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(g_bThreadStarted);
}

void CMyex35aView::OnUpdateStart(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(!g_bThreadStarted);
}

```

Build and run. Type the localhost address or any other address if you are online and click the green button. You can stop the progress anytime by pressing the red button that only visible when the program is in progress.

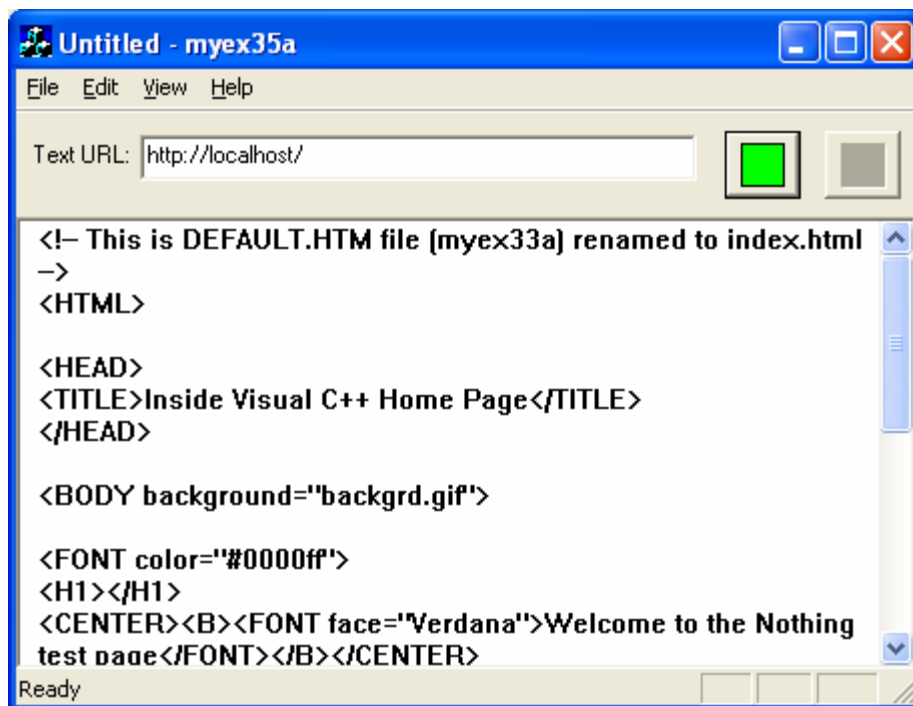


Figure 41: MYEX35A phase 2 program output in action, testing the localhost.

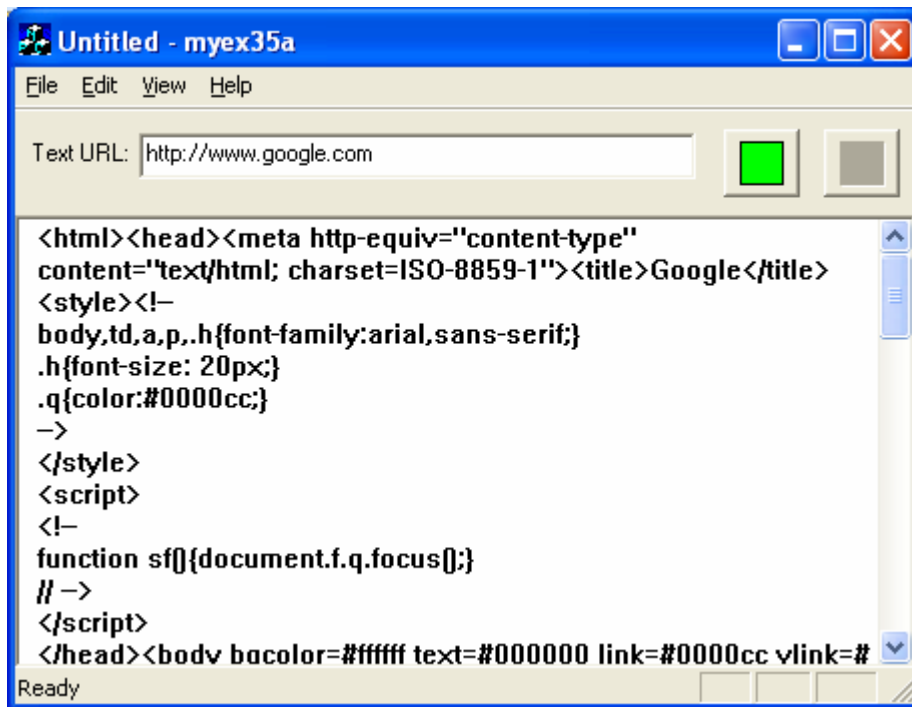


Figure 42: MYEX35A phase 2 program output in action, testing the google.com.

Use the **File Open** menu to open the previous **test.35a** file.



Figure 43: MYEX35A phase 2 program output in action, opening **test.35a** file.

Create a text file, let name it **test.txt**. Write some content and put it under the wwwroot directory and type the address of the text file as shown below. Press the green button.

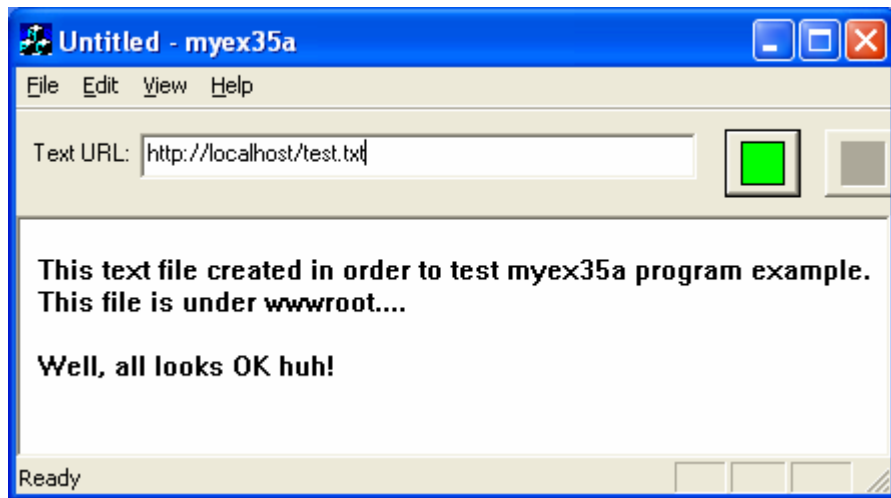


Figure 44: MYEX35A phase 2 program output in action, opening text file.

It seems very useful and fun isn't it? Let have some story for what we have done.

## The Story of MYEX35A Phase 2: Adding WinInet Calls

The completed MYEX35A example includes two dialog bar objects, one for the main frame window and another for the in-place frame window. Both are attached to the same resource template, IDD\_DIALOGBAR, which contains an edit control that accepts a text file URL plus start and stop buttons that display green and red bitmaps. If you click the green button (handled by the `OnStart()` member function of the `CMyex35aView` class), you'll start a thread that reads the text file one line at a time. The thread code from the file **UrlThread.cpp** is shown here:

```
CString g_strURL = "http:// ";
volatile BOOL g_bThreadStarted = FALSE;
CEvent g_eKill;

UINT UrlThreadProc(LPVOID pParam)
{
    g_bThreadStarted = TRUE;
    CString strLine;
    CInternetSession session;
    CStdioFile* pFile1 = NULL;

    try {
        pFile1 = session.OpenURL(g_strURL, 0,
INTERNET_FLAG_TRANSFER_BINARY | INTERNET_FLAG_KEEP_CONNECTION); // needed
// for Windows NT c/r authentication
        // Keep displaying text from the URL until the Kill event is
        // received
        while(::WaitForSingleObject(g_eKill.m_hObject, 0) !=
WAIT_OBJECT_0) {
            // one line at a time
            if(pFile1->ReadString(strLine) == FALSE) break;
            strLine += '\n';
            ::SendMessage((HWND) pParam, EM_SETSEL, (WPARAM) 999999,
1000000);
            ::SendMessage((HWND) pParam, EM_REPLACESEL, (WPARAM) 0,
(LPARAM) (const char*) strLine);
            Sleep(250); // Deliberately slow down the transfer
        }
    }
    catch(CInternetException* e) {
```

```

        LogInternetException(pParam, e);
        e->Delete();
    }
    if(pFile1 != NULL) delete pFile1; // closes the file-prints a warning
    g_bThreadStarted = FALSE;
    // Post any message to update the toolbar buttons
    ::PostMessage((HWND) pParam, EM_SETSEL, (WPARAM) 999999, 1000000);
    TRACE("Post thread exiting normally\n");
    return 0;
}

```

This code uses the CStdioFile pointer to pFile1 returned from OpenURL( ). The ReadString member function reads one line at a time, and each line is sent to the rich edit view window. When the main thread sets the "kill" event (the red button), the URL thread exits.

Before you test MYEX35A, make sure that the server (MYEX34A or IIS) is running and that you have a **text file** in the server's home directory. Test the MYEX35A program first in stand-alone mode by entering the **text file URL** in the dialog bar. Next try running the program in server mode from Internet Explorer. Enter **test.35a** (the document you created when you ran MYEX35A in stand-alone mode) in Internet Explorer's Address field to load the server.

We considered using the CAsyncMonikerFile class instead of the MFC WinInet classes to read the text file. We stuck with WinInet, however, because the program could use the CStdioFile class ReadString( ) member function to "pull" individual text lines from the server when it wanted them. The CAsyncMonikerFile class would have "pushed" arbitrary blocks of characters into the program (by calling the overridden OnDataAvailable( ) function) as soon as the characters had been received.

## Displaying Bitmaps on Buttons

[Module 21](#) describes the CBitmapButton class for associating a group of bitmaps with a pushbutton. Microsoft Windows 95, Microsoft Windows 98, and Microsoft Windows NT 4.0 above support an alternative technique that associates a single bitmap with a button. First you apply the Bitmap style (on the button's property sheet) to the button, and then you declare a variable of class CBitmap that will last at least as long as the button is enabled. Then you make sure that the CButton::SetBitmap function is called just after the button is created. Here is the code for associating a bitmap with a button, from the MYEX35A **MainFrm.cpp** and **IpFrame.cpp** files:

```

m_bitmapGreen.LoadBitmap(IDB_GREEN);
HBITMAP hBitmap = (HBITMAP) m_bitmapGreen.GetSafeHandle();
((CButton*) m_wndDialogBar.GetDlgItem(IDC_START))
->SetBitmap(hBitmap);

```

If your button was in a dialog, you could put similar code in the OnInitDialog( ) member function and declares a CBitmap member in the class derived from CDialog.

## ActiveX Document Server Example MYEX35B

Look at the pizza form example from [Module 33](#) (MYEX34A). Note that the server (the ISAPI DLL) processes the order only when the customer clicks the **Submit Order Now** button. This is okay for ordering pizzas because you're probably happy to accept money from anyone, no matter what kind of browser is used.

For a form-based intranet application, however, you can be more selective. You can dictate what browser your clients have, and you can distribute your own client software on the net. In that environment, you can make data entry more sophisticated, allowing, for example, the client computer to validate each entry as the user types it. That's exactly what's happening in MYEX35B, which is another **ActiveX document server**, of course. MYEX35B is a form-based employee time-sheet entry program that works inside Internet Explorer (as shown in Figure 45) or works as a stand-alone application. Looks like a regular HTML form, doesn't it? It's actually an MFC form view, but the average user probably won't know the difference. The **Name** field is a drop-down combo box, however, which is different from the select field you would see in an HTML form, because the user can type in a value if necessary. The **Job Number** field has a spin button control that helps the user select the value. These aren't necessarily the ideal controls for time-sheet entry, but the point here is that you can use any Windows controls you want, including tree controls, list controls, trackbars, and ActiveX controls, and you can make them interact any way you want.

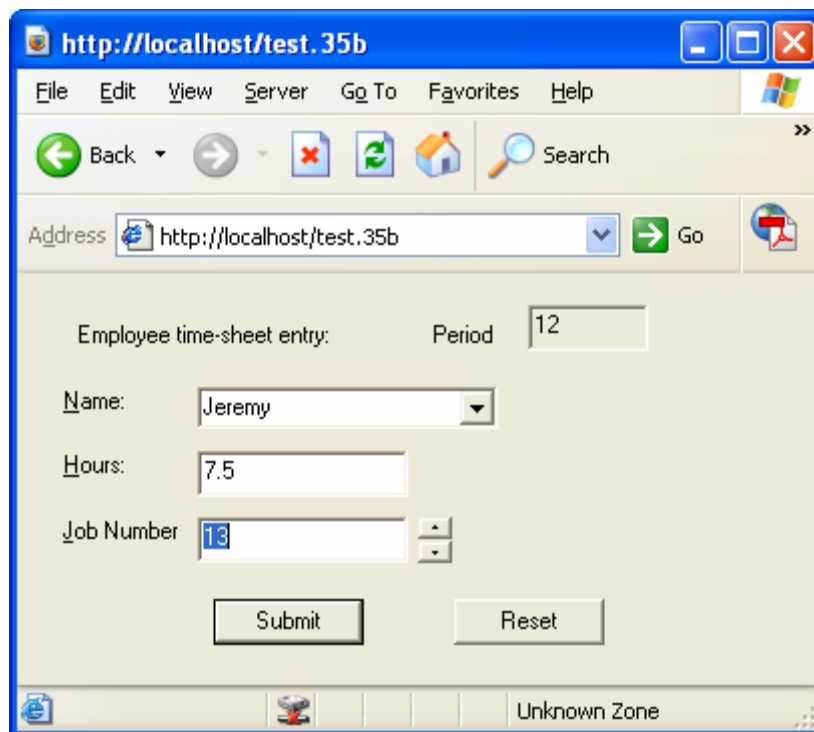


Figure 45: Employee time-sheet entry form.

### MYEX35B From Scratch

Let build MYEX35B from scratch. This is an SDI application program. Follow the shown steps.

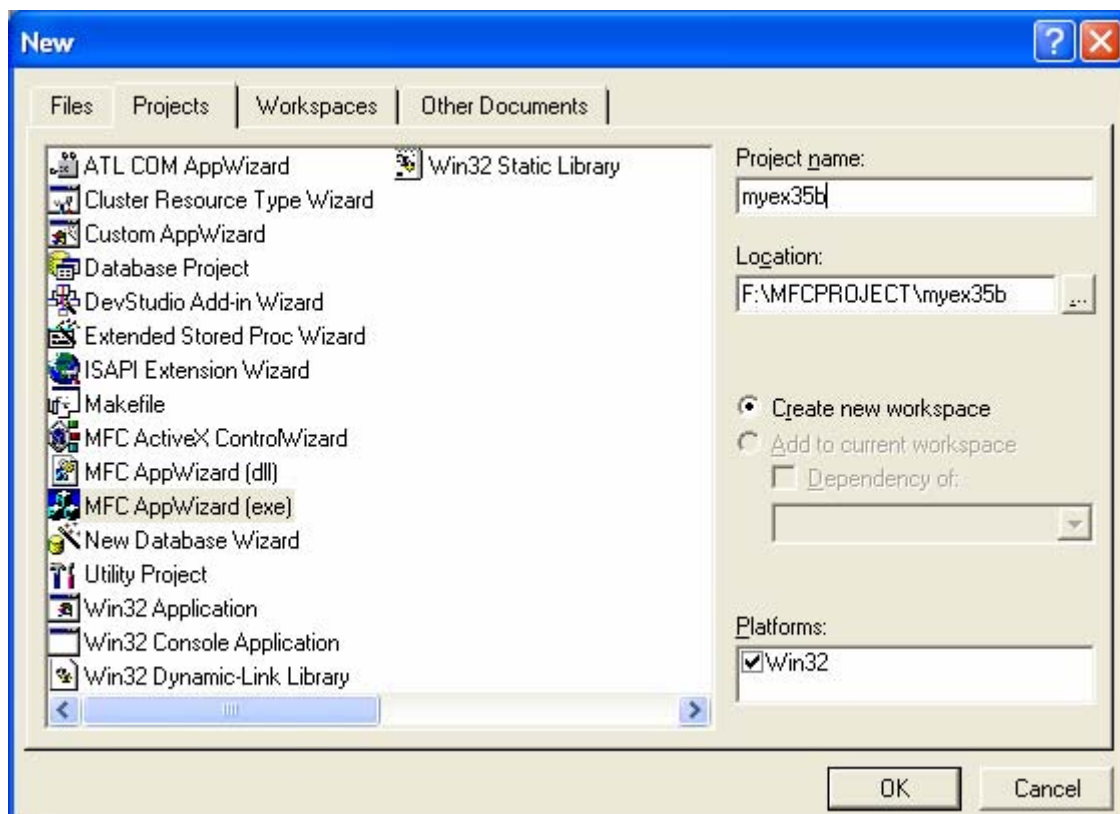


Figure 46: MYEX35B – MFC AppWizard new project dialog.

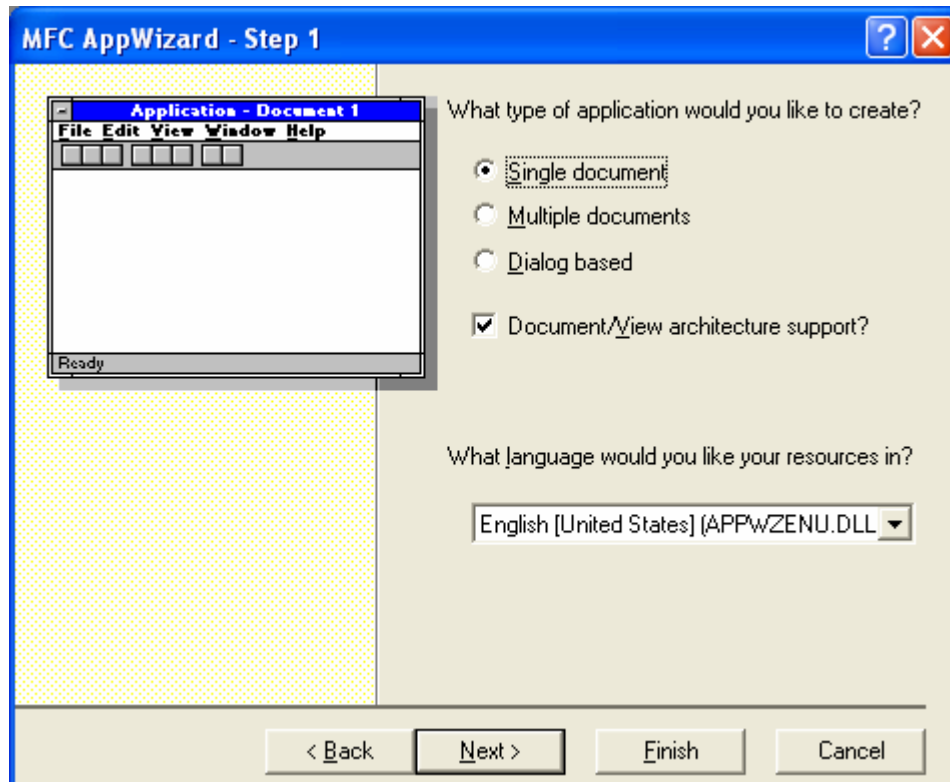


Figure 47: MYEX35B – MFC AppWizard step 1 of 6.

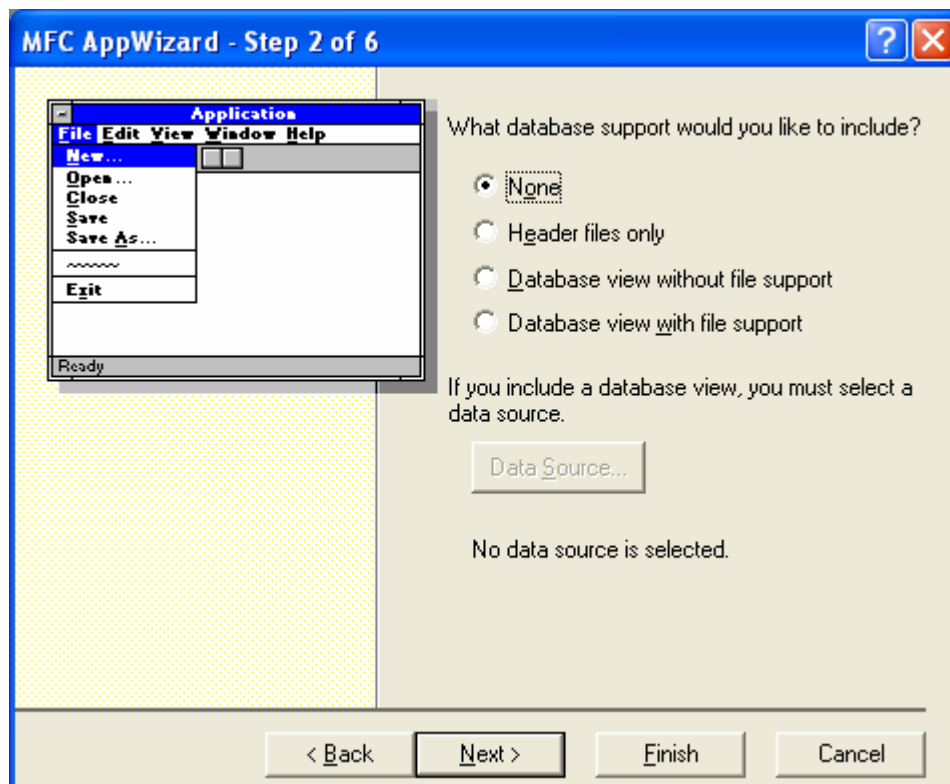


Figure 48: MYEX35B – MFC AppWizard step 2 of 6.

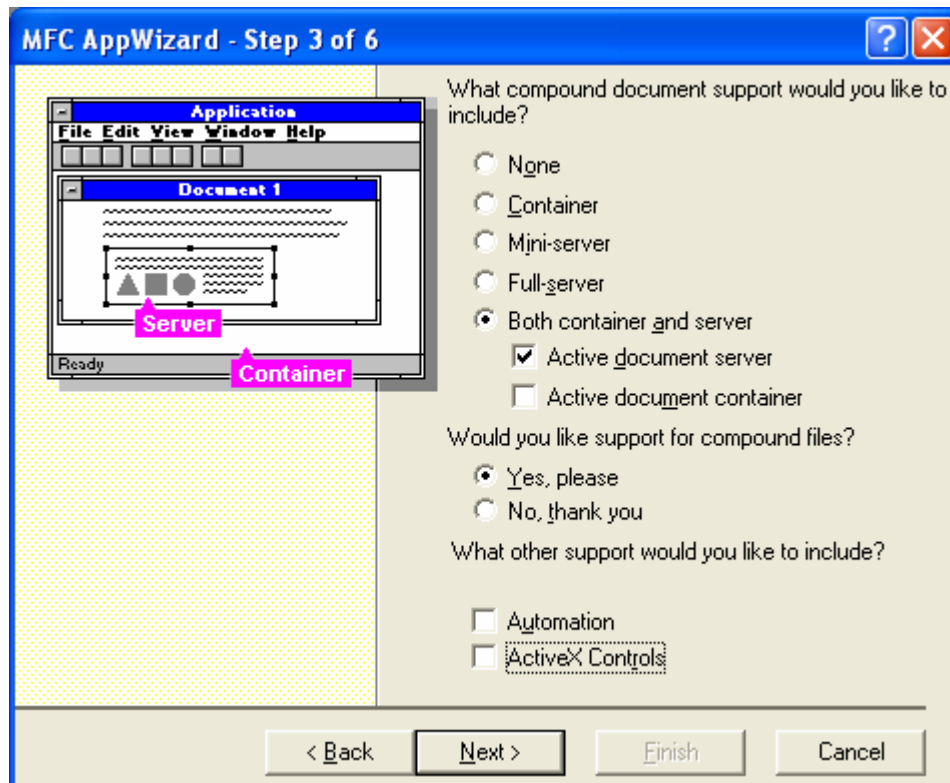


Figure 49: MYEX35B – MFC AppWizard step 3 of 6, selecting the compound document and deselecting the Automation and ActiveX Controls.

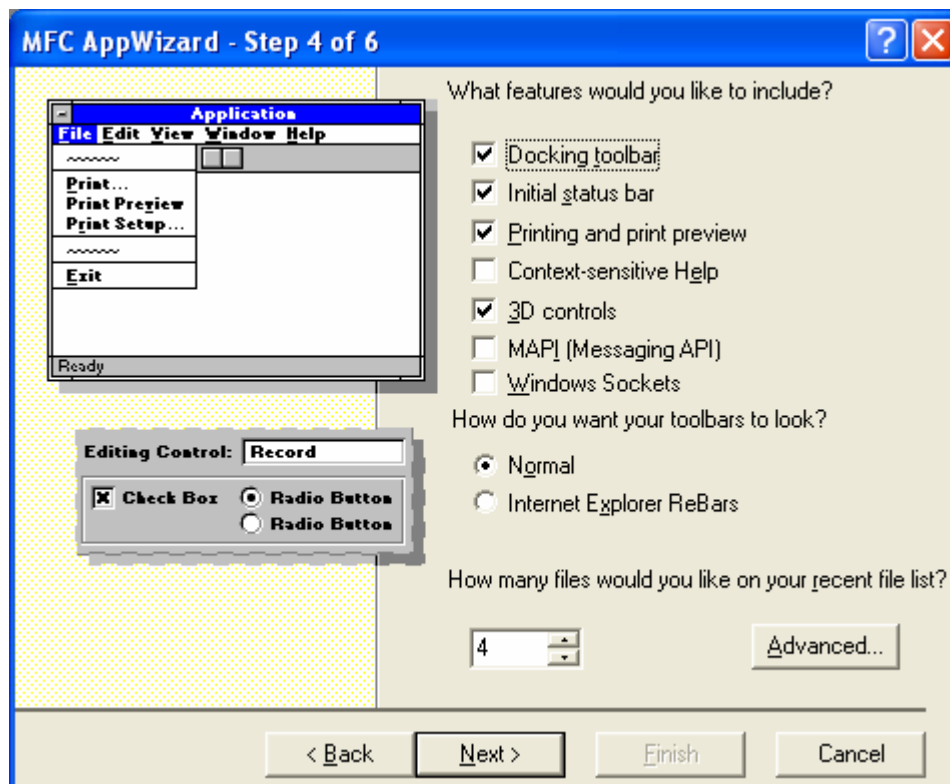


Figure 50: MYEX35B – MFC AppWizard step 4 of 6.

Click the **Advanced** button. Enter the **File Extension** as shown below.

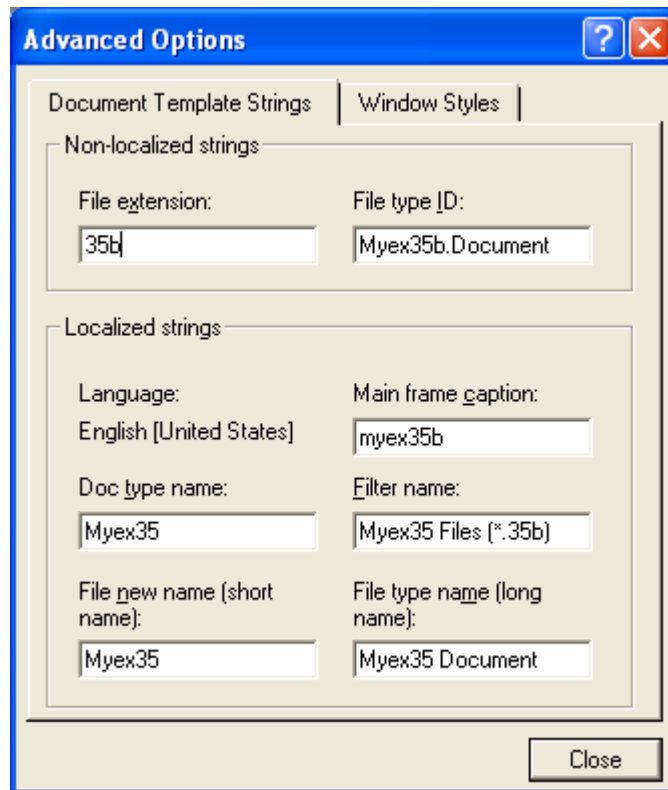


Figure 51: MYEX35B – entering a file extension.

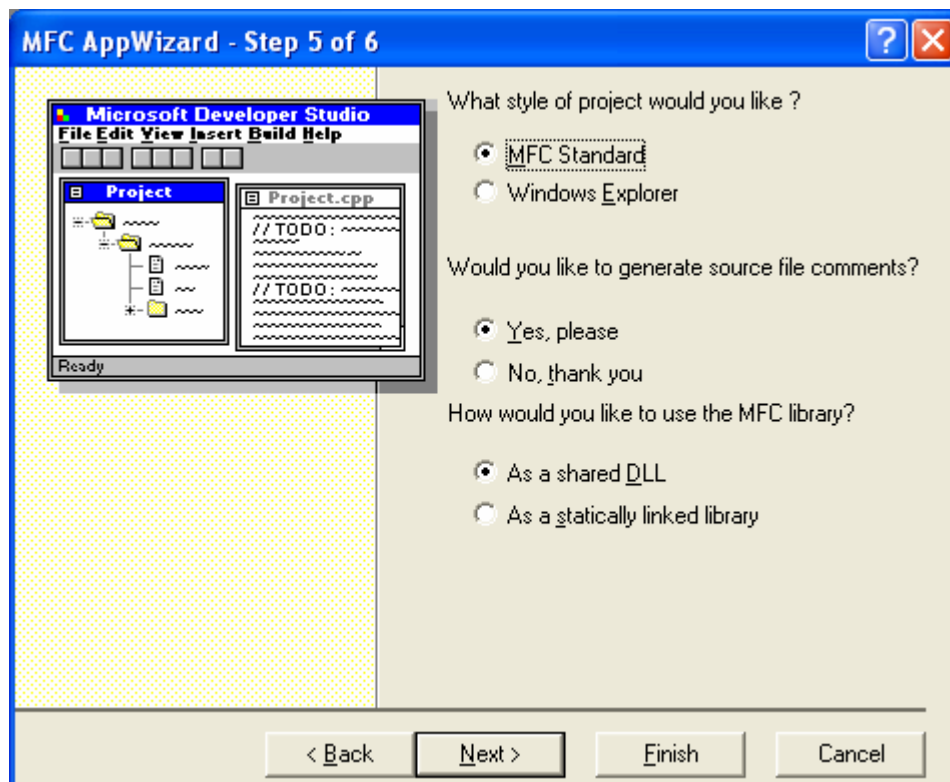


Figure 52: MYEX35B – MFC AppWizard step 5 of 6.

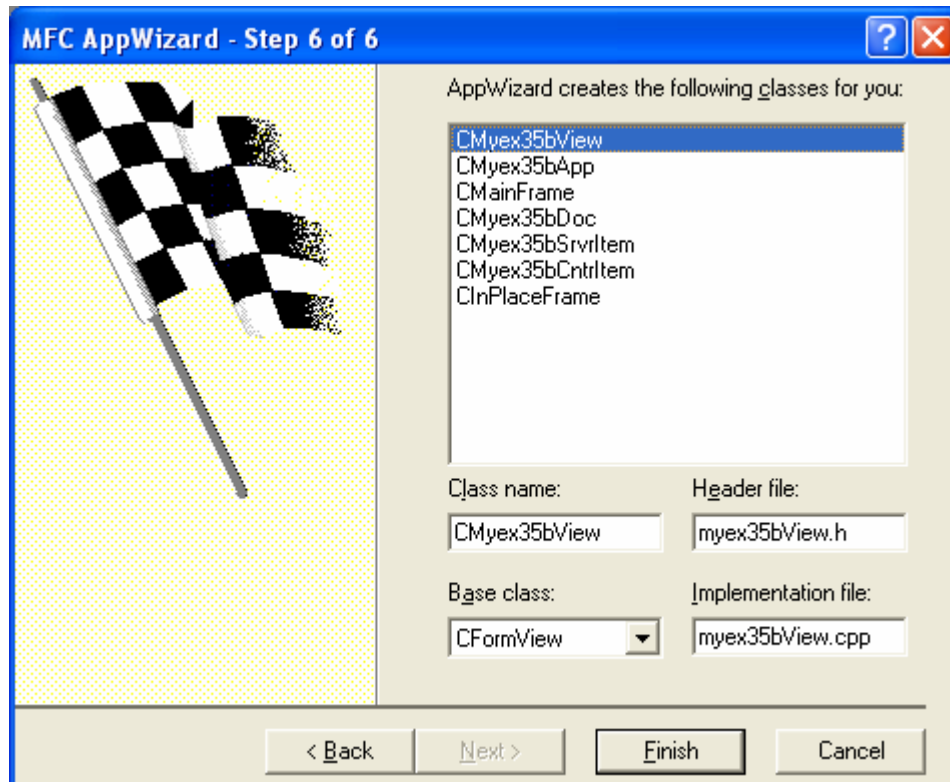


Figure 53: MYEX35B – MFC AppWizard step 6 of 6, Selecting CFormView as the base class.

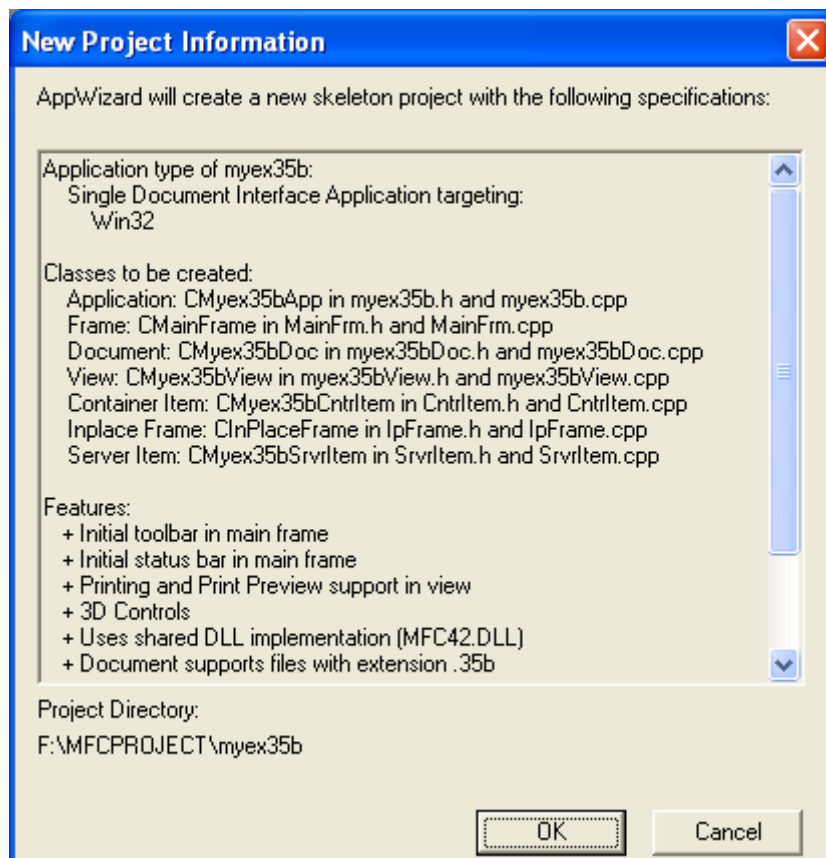


Figure 54: MYEX35B project summary.

Insert a dialog and controls. Use the following information and Figures for the dialog and controls.

Resource	ID	Caption/text
Dialog	IDD_ADDRDIALOG	Internet Address Dialog
Static text	-	Server Name:
Static text	-	File:
Edit control	IDC_SERVERNAME	-
Edit control	IDC_FILE	-

Table 3.

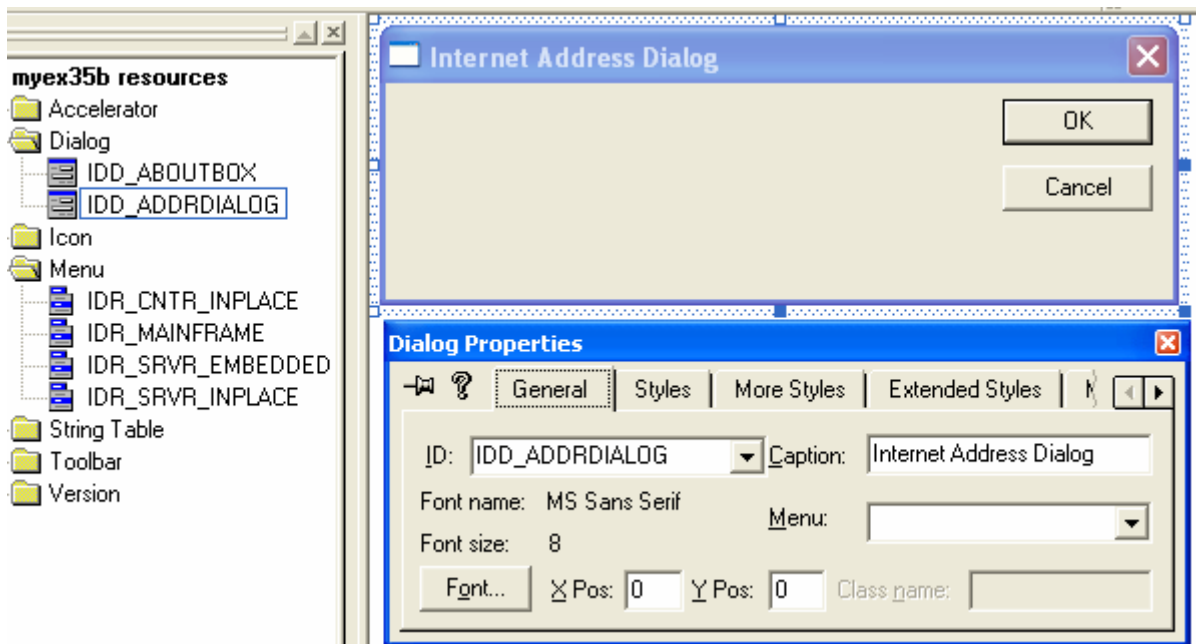


Figure 55: IDD\_ADDRDIALOG property page.

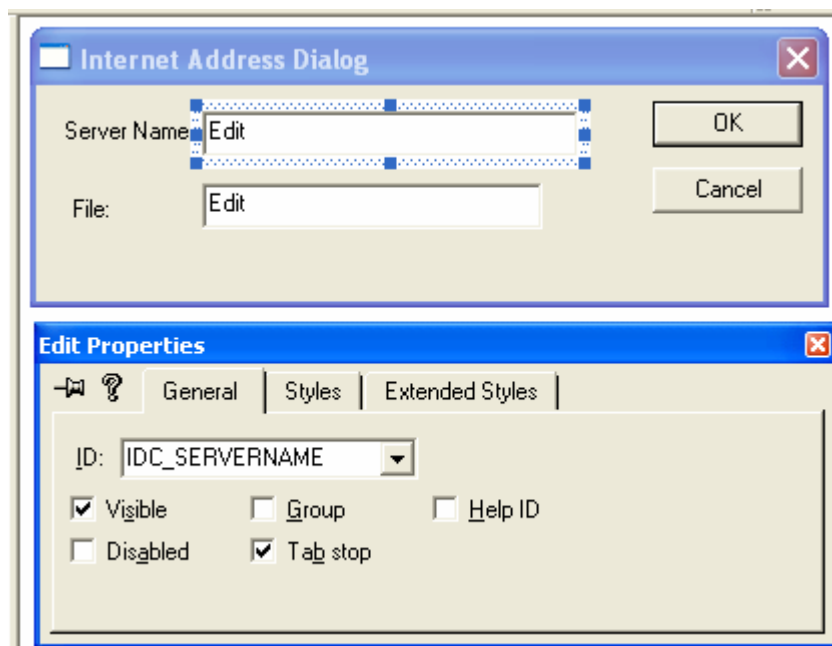


Figure 56: IDC\_SERVERNAME, an edit control property page.

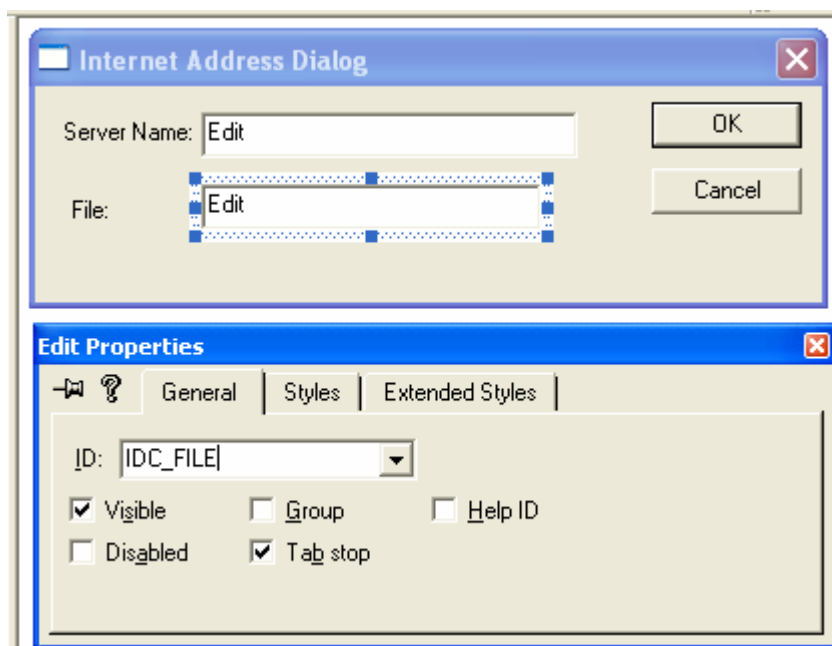


Figure 57: IDC\_FILE property page.

Insert another dialog and use the following information.

Resource	ID	Caption/text/other
Dialog	IDD_MYEX35B_FORM	Styles – Style: Child, Border: None, deselect Title bar

Table 4.

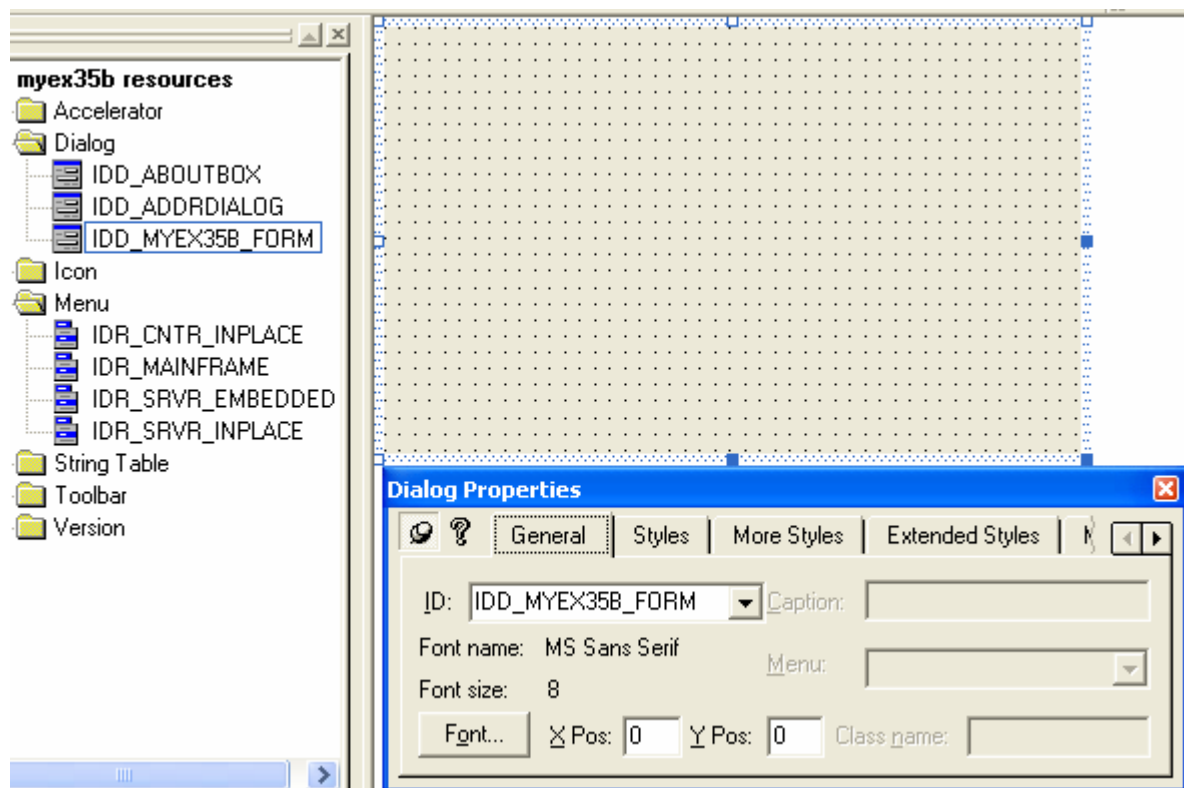


Figure 58: IDD\_MYEX35B\_FORM dialog property page.

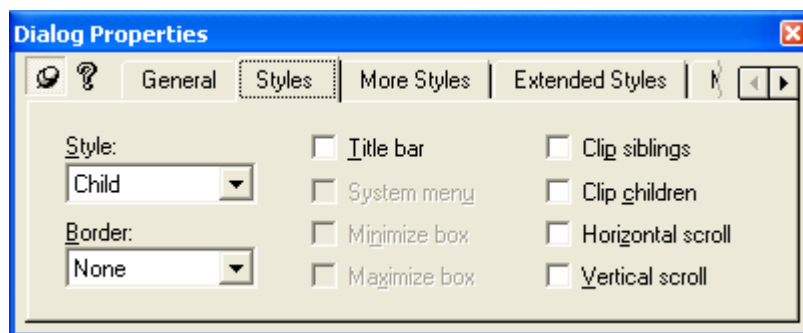


Figure 59: Styles dialog property page.

Resource	ID	Caption/text/other
Static texts	-	Employee time-sheet entry: Period
Static texts	-	Name:
Static texts	-	Hours:
Static texts	-	Job Number:
Edit control	IDC_PERIOD	Styles - Left, Number, Auto HScroll, Border, Read-only.
Combo box	IDC_EMPLOYEE	Data – Kurt, Cobain, Jeremy, Matt Cutt, William, Dunkel, Diana, Major, Masterjee, Stallman (separate the name with <b>Ctrl + Enter</b> )
Edit control	IDC_HOURS	-
Edit control	IDC_JOB	Styles - Number
Spin control	IDC_SPIN1	Styles - Vertical, Unattached, Auto buddy, Set buddy integer, Arrow keys.
Button control	IDC_SUBMIT	Submit
Button control	IDCANCEL	Reset

Table 5.

The screenshot displays a form titled "Employee time-sheet entry:" with a "Period" label and an "Edit" button. Below this, there are three input fields: "Name:" (a dropdown menu), "Hours:" (a text box with "Edit" inside), and "Job Number" (a text box with "Edit" inside). At the bottom of the form are "Submit" and "Reset" buttons. Below the form is a dialog box titled "Edit Properties" with a blue header and a close button. The dialog has three tabs: "General", "Styles", and "Extended Styles". The "General" tab is selected, showing an "ID:" dropdown menu with "IDC\_PERIOD" selected. Below this are several checkboxes: "Visible" (checked), "Group" (unchecked), "Help ID" (unchecked), "Disabled" (unchecked), and "Tab stop" (checked).

Figure 60: IDC\_PERIOD property page.

The screenshot displays the same form as Figure 60, but the "Edit Properties" dialog box is now on the "Styles" tab. The "General" tab is still selected in the dialog, but the "Styles" tab is active, showing various style options. The "Align text:" dropdown is set to "Left". Other options include "Horizontal scroll" (unchecked), "Password" (unchecked), "Border" (checked), "Auto HScroll" (checked), "No hide selection" (unchecked), "Uppercase" (unchecked), "Multiline" (unchecked), "Vertical scroll" (unchecked), "OEM convert" (unchecked), "Lowercase" (unchecked), "Number" (checked), "Auto VScroll" (unchecked), "Want return" (unchecked), and "Read-only" (checked).

Figure 61: IDC\_PERIOD Styles property page.

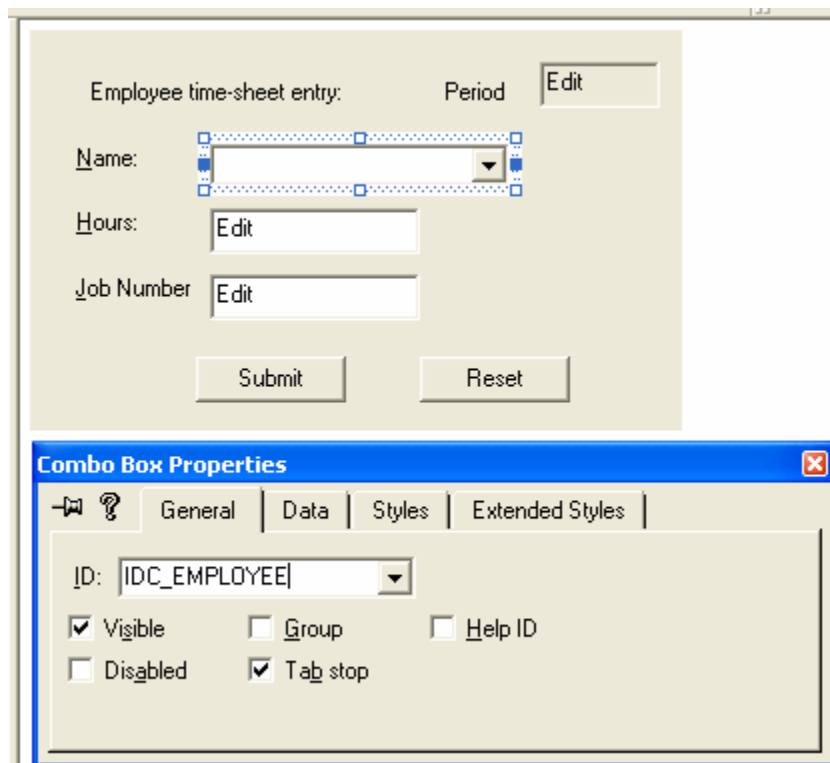


Figure 62: IDC\_EMPLOYEE property page.

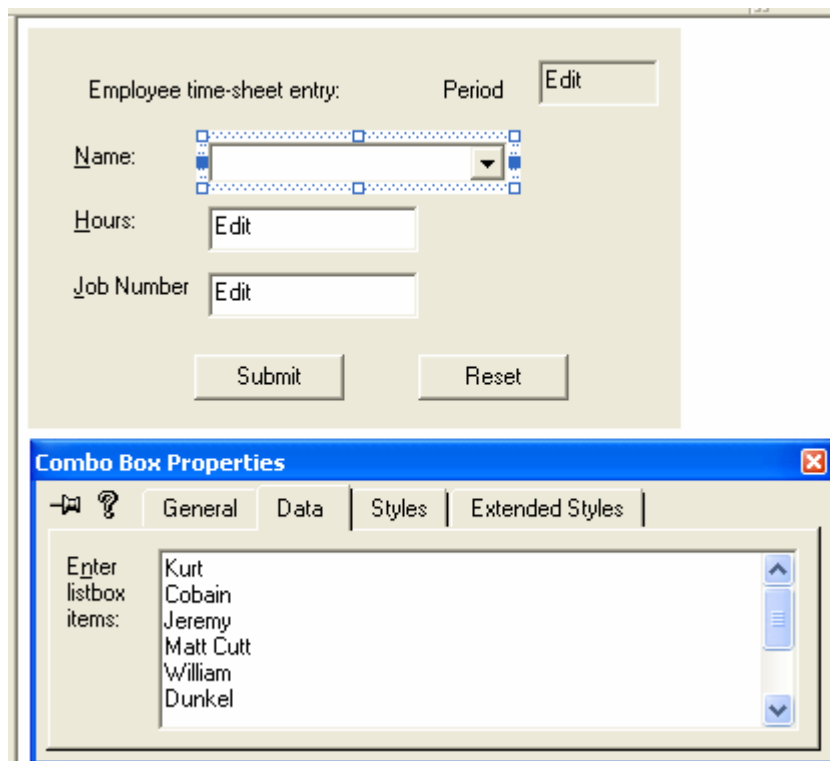


Figure 63: IDC\_EMPLOYEE sample data.

Employee time-sheet entry:      Period

Name:

Hours:

Job Number

---

**Edit Properties** ✕

General   Styles   Extended Styles

ID:

☒ Visible      ☐ Group      ☐ Help ID

☐ Disabled      ☒ Tab stop

Figure 64: IDC\_HOURS property page.

Employee time-sheet entry:      Period

Name:

Hours:

Job Number

---

**Edit Properties** ✕

General   Styles   Extended Styles

ID:

☒ Visible      ☐ Group      ☐ Help ID

☐ Disabled      ☒ Tab stop

Figure 65: IDC\_JOB property page.

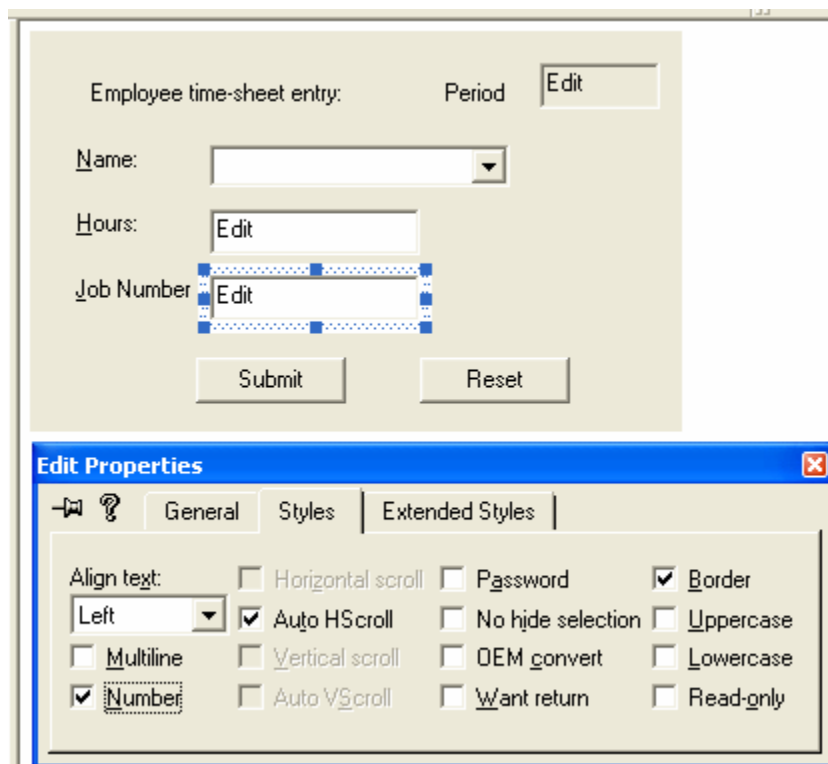


Figure 66: IDC\_JOB Styles property page.

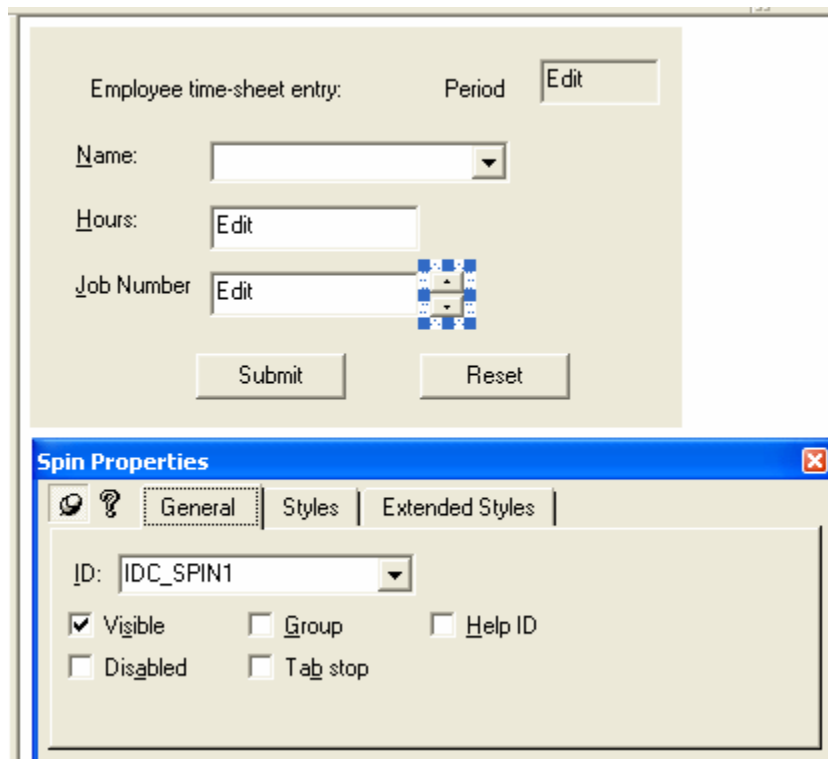


Figure 67: IDC\_SPIN1 property job.

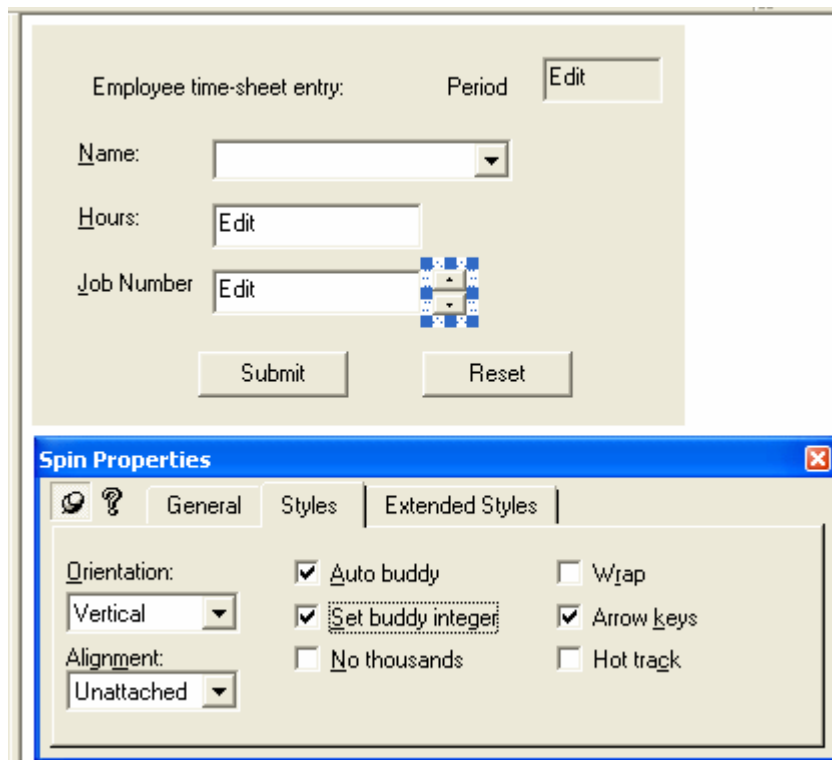


Figure 68: IDC\_SPIN1 Styles property job.

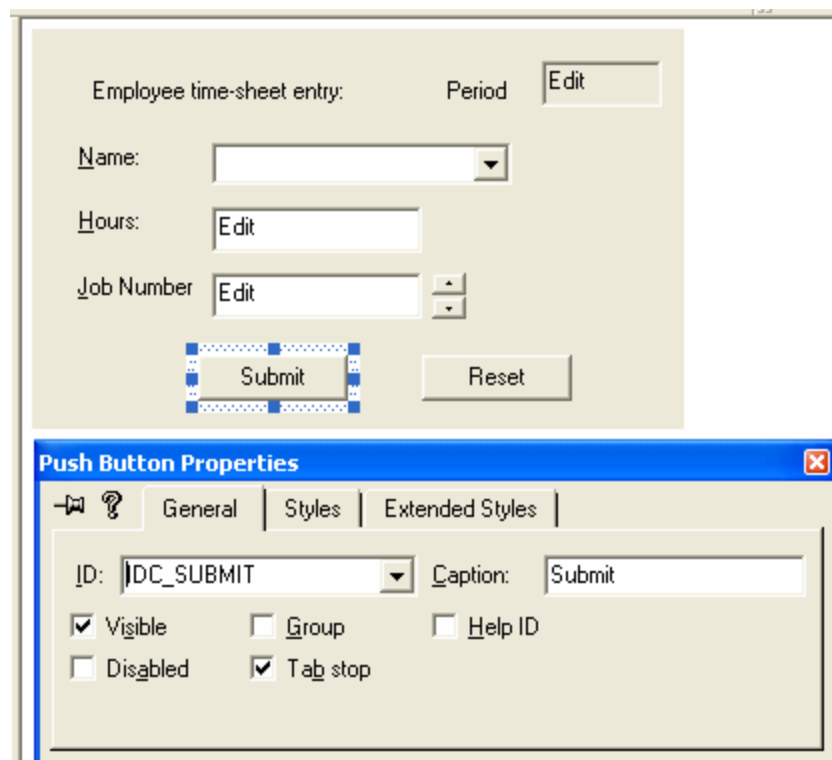


Figure 69: IDC\_SUBMIT property page.

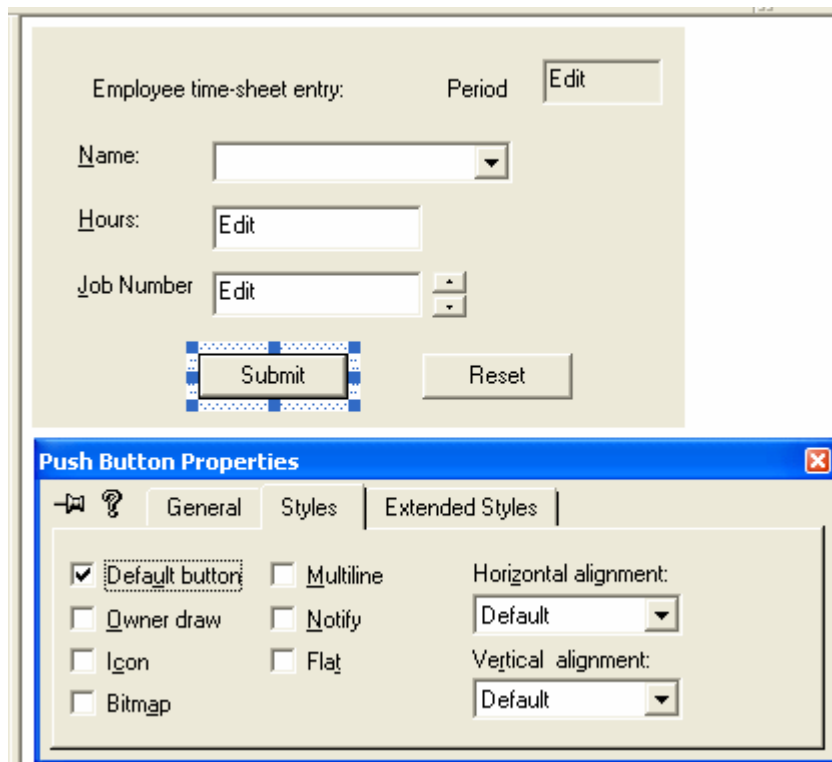


Figure 70: IDC\_SUBMIT Styles property page.

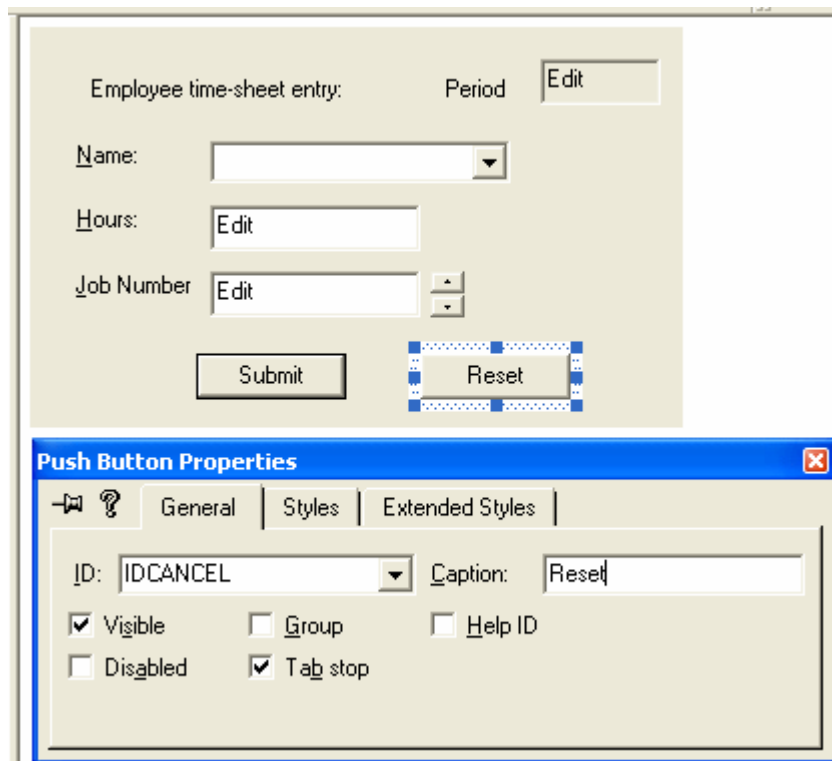


Figure 71: IDCANCEL property page.

Add menu and item in IDR\_MAINFRAME, IDR\_ SRVR\_EMBEDDED and IDR\_SRVR\_INPLACE.

Resource	ID	Caption/text/other
Menu	-	Server
Menu item	ID_SERVER_ADDRESS	Address

Table 6.

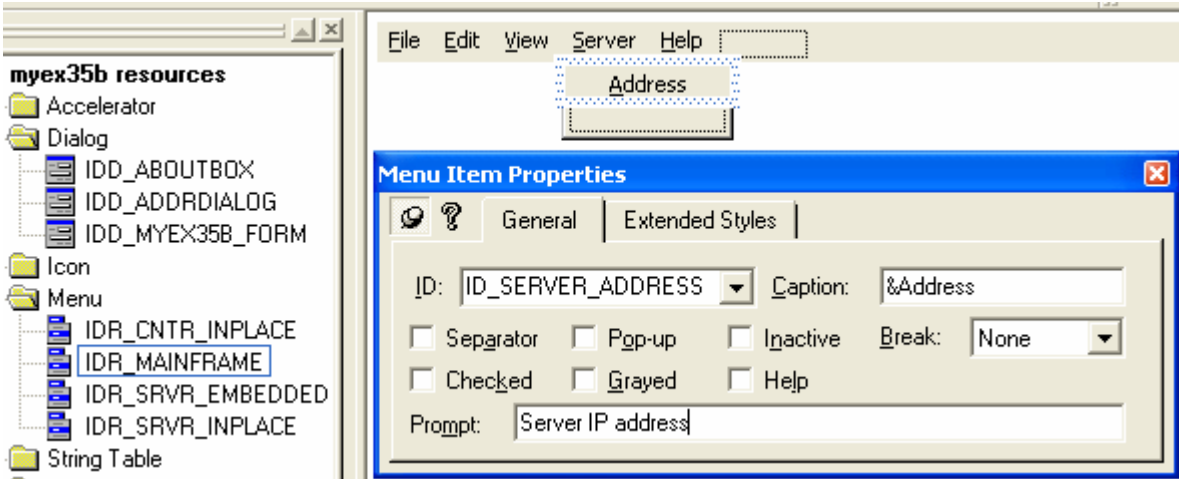


Figure 72: Adding menu and menu item to IDR\_MAINFRAME.

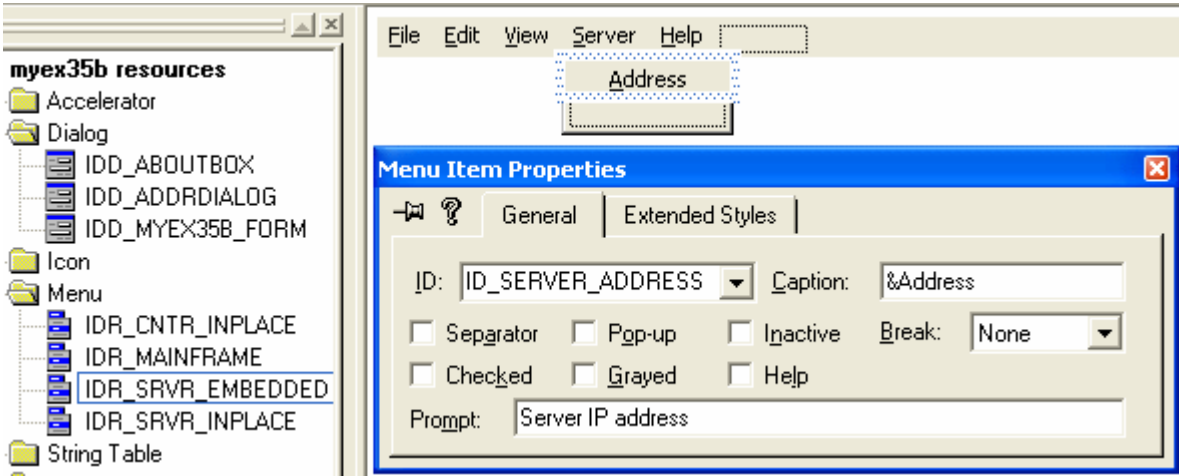


Figure 73: Adding menu and menu item to IDR\_SRVR\_EMBEDDED.

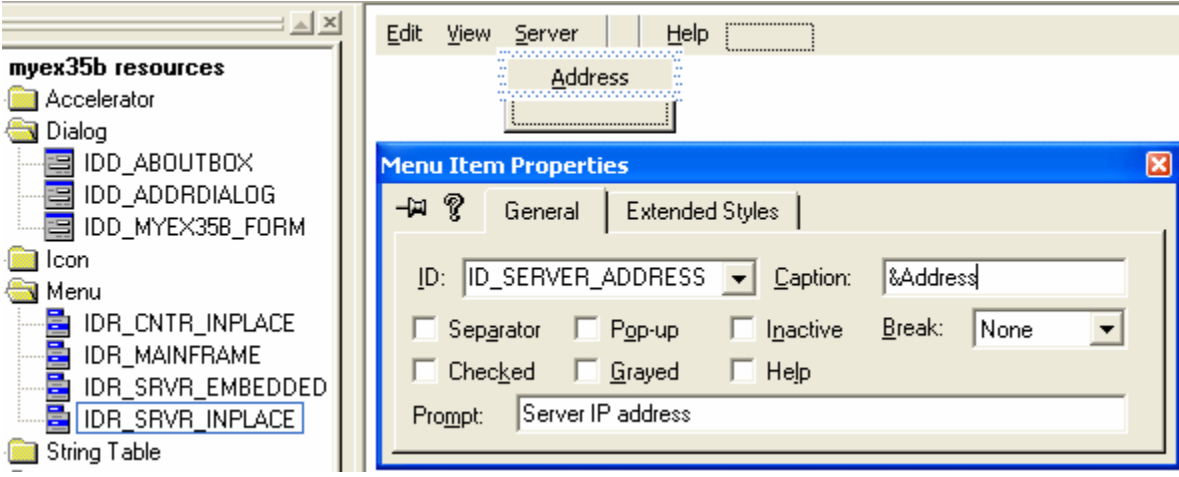


Figure 74: Adding menu and menu item to IDR\_SRVR\_INPLACE.

In ResourceView select IDD\_ADDRDIALOG and launch ClassWizard. Select **Create a new class** and click **OK**.

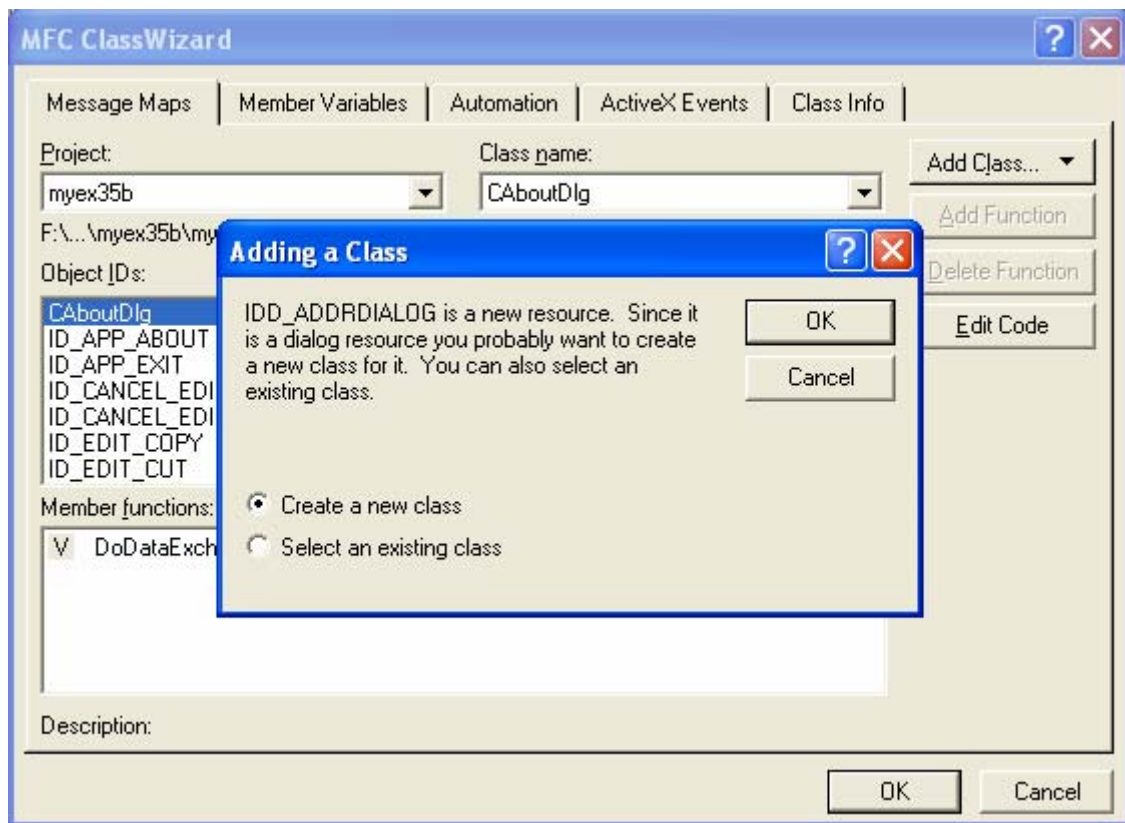


Figure 75: Adding a new class to project.

Fill in as shown below. Click **OK**.

The 'New Class' dialog box is shown with the following fields:

- Class information:**
  - Name:** CAddrDialog
  - File name:** AddrDialog.cpp (with a 'Change...' button)
  - Base class:** CDialog
  - Dialog ID:** IDD\_ADDRDIALOG
- Automation:**
  - ☒ None
  - ☐ Automation
  - ☐ Createable by type ID: myex35b.AddrDialog

Buttons: OK, Cancel.

Figure 76: Entering IDD\_ADDRDIALOG class information.

Add the following member variables to CAddrDialog class.

ID	Type	Variable name
IDC_FILE	CString	m_strFile
IDC_SERVERNAME	CString	m_strServerName

Table 7.

The 'Add Member Variable' dialog box is shown with the following fields:

- Member variable name:** m\_strServerName
- Category:** Value
- Variable type:** CString
- Description:** CString with length validation

Buttons: OK, Cancel.

Figure 77: Adding member variable to class.

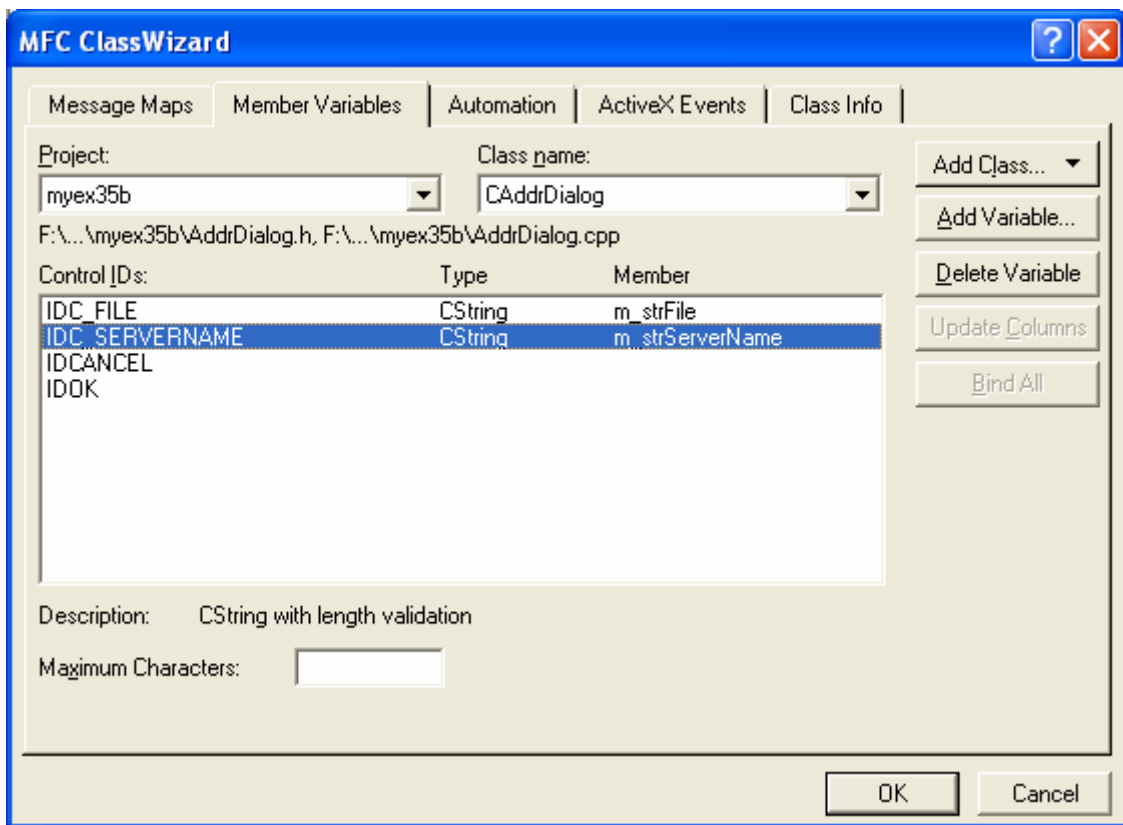


Figure 78: The added member variables.

Add a new class. Click the **Add Class** button and select **New**. Add CValidForm class information as shown below.

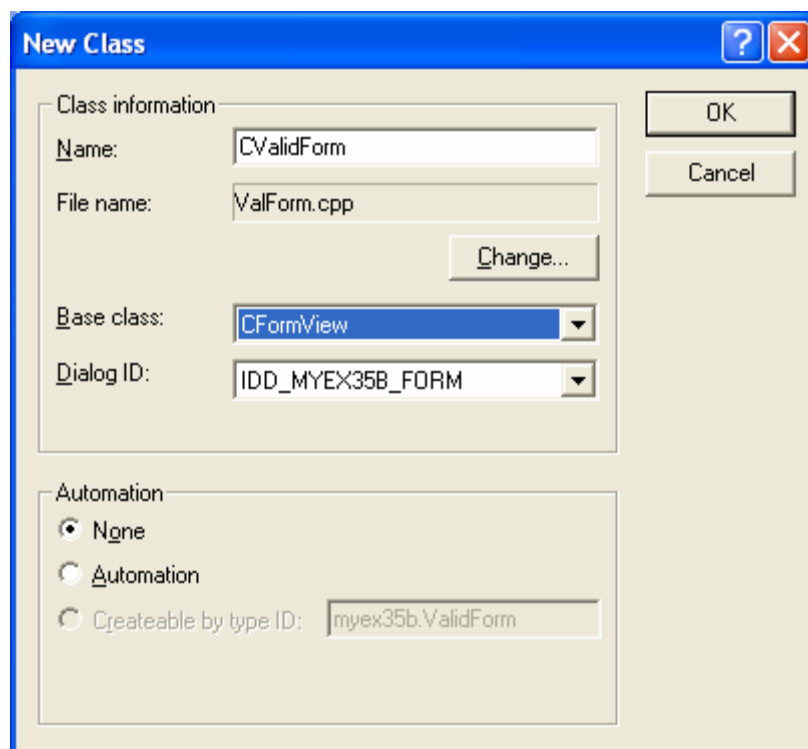


Figure 79: Adding CValidForm class information.

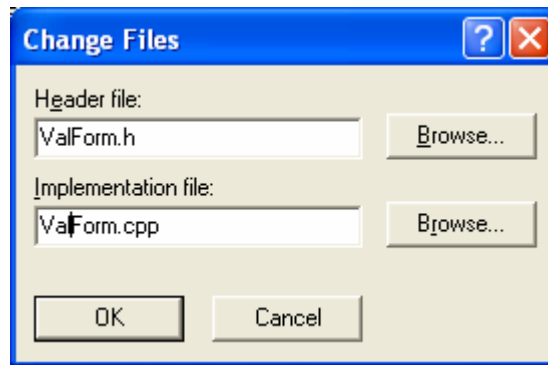


Figure 80: Modifying the source and header files for the class.

Add new header and source files, **PostThread.h** and **PostThread.cpp**.

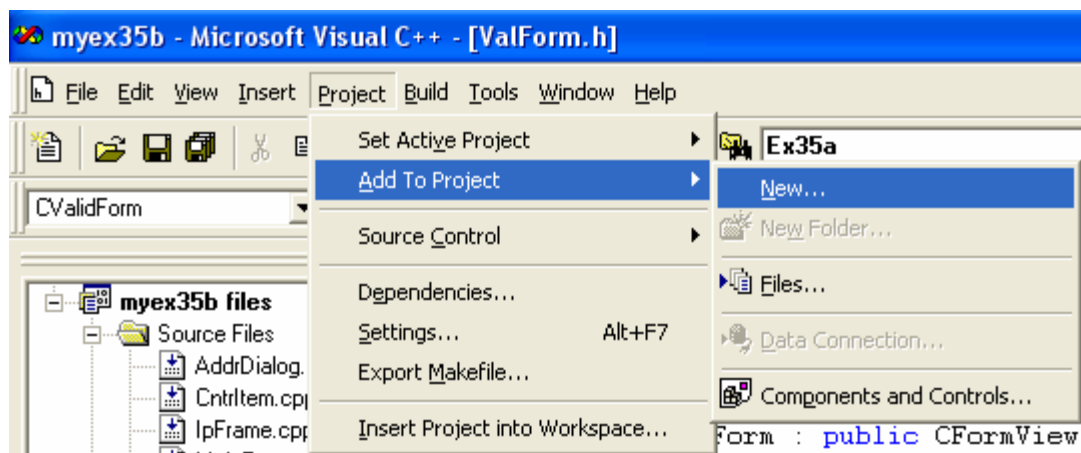


Figure 81: Adding new files to a class.

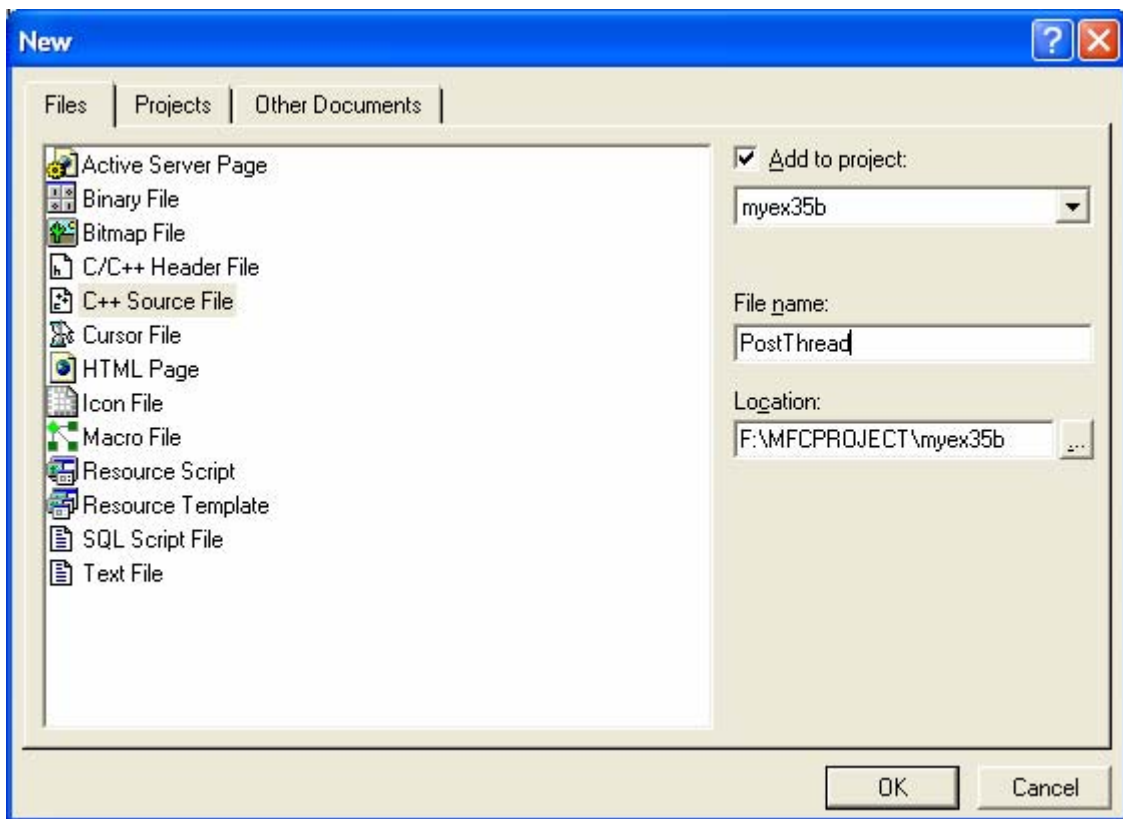


Figure 82: Adding new header and source files, PostThread.h and PostThread.cpp.

Copy the following codes to **PostThread.h** and **PostThread.cpp** file respectively.

```
// PostThread.h

#define WM_POSTCOMPLETE WM_USER + 6

extern CString g_strFile;
extern CString g_strServerName;
extern CString g_strParameters;

UINT PostThreadProc(LPVOID pParam);

// PostThread.cpp (uses MFC Wininet calls)

#include <stdafx.h>
#include "PostThread.h"

#define MAXBUF 50000

CString g_strFile = "/scripts/myex34a.dll";
CString g_strServerName = "localhost";
CString g_strParameters;

UINT PostThreadProc(LPVOID pParam)
{
    CInternetSession session;
    CHttpConnection* pConnection = NULL;
    CHttpFile* pFile1 = NULL;
    char* buffer = new char[MAXBUF];
    UINT nBytesRead = 0;
    DWORD dwStatusCode;
    BOOL bOkStatus = FALSE;
```

```

try
{
    pConnection = session.GetHttpConnection(g_strServerName, (INTERNET_PORT) 80);
    pFile1 = pConnection->OpenRequest(0, g_strFile + "?ProcessTimesheet", // POST
                                     // request
                                     NULL, 1, NULL, NULL, INTERNET_FLAG_KEEP_CONNECTION |
INTERNET_FLAG_RELOAD); // no cache
    pFile1->SendRequest(NULL, 0, (LPVOID) (const char*) g_strParameters,
                       g_strParameters.GetLength());
    pFile1->QueryInfoStatusCode(dwStatusCode);
    if(dwStatusCode == 200) { // OK status
        // doesn't matter what came back from server
        // -- we're looking for OK status
        bOkStatus = TRUE;
        nBytesRead = pFile1->Read(buffer, MAXBUF - 1);
        buffer[nBytesRead] = '\0'; // necessary for TRACE
        TRACE(buffer);
        TRACE("\n");
    }
}
catch(CInternetException* pe) {
    char text[100];
    pe->GetErrorMessage(text, 99);
    TRACE("WinInet exception %s\n", text);
    pe->Delete();
}
if(pFile1) delete pFile1; // does the close -- prints a warning
if(pConnection) delete pConnection; // why does it print a warning?
delete [] buffer;
::PostMessage((HWND) pParam, WM_POSTCOMPLETE, (WPARAM) bOkStatus, 0);
return 0;
}

```

Listing 11.

## The Coding Part

Add the following `#include` directives in **StdAfx.h**.

```

#include <afxinet.h>
#include <afxmt.h>

#ifdef _AFX_NO_AFXCMN_ST
#include <afxinet.h>
#include <afxmt.h>
|
//{{AFX_INSERT_LOCATION}}

```

Listing 12.

## CMainFrame Class

Add the following codes in **MainFrm.cpp**.

```

// TODO: Remove this if you don't want tool tips or a resizable toolbar
m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

```

```

// TODO: Remove this if you don't want tool tips or a
// resizable toolbar
m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

// TODO: Delete these three lines if you don't want the
// , , ,

```

Listing 13.

Edit the `PreCreateWindow()` as shown below.

```

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    return CFrameWnd::PreCreateWindow(cs);
}

```

Listing 14.

Add the following code in `InitInstance()`.

```

    AfxEnableControlContainer();

BOOL CMyex35bApp::InitInstance()
{
    // Initialize OLE libraries
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }

    AfxEnableControlContainer();

    // Standard initialization

```

Listing 15.

And delete the following code at the end of the `InitInstance()`.

```

// The one and only window has been initialized, so show and update it.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

```

## CInPlaceFrame Class

Using `ClassView`, delete `OnCreateControlBars()` virtual function and delete the implementation in `IpFrame.cpp`.

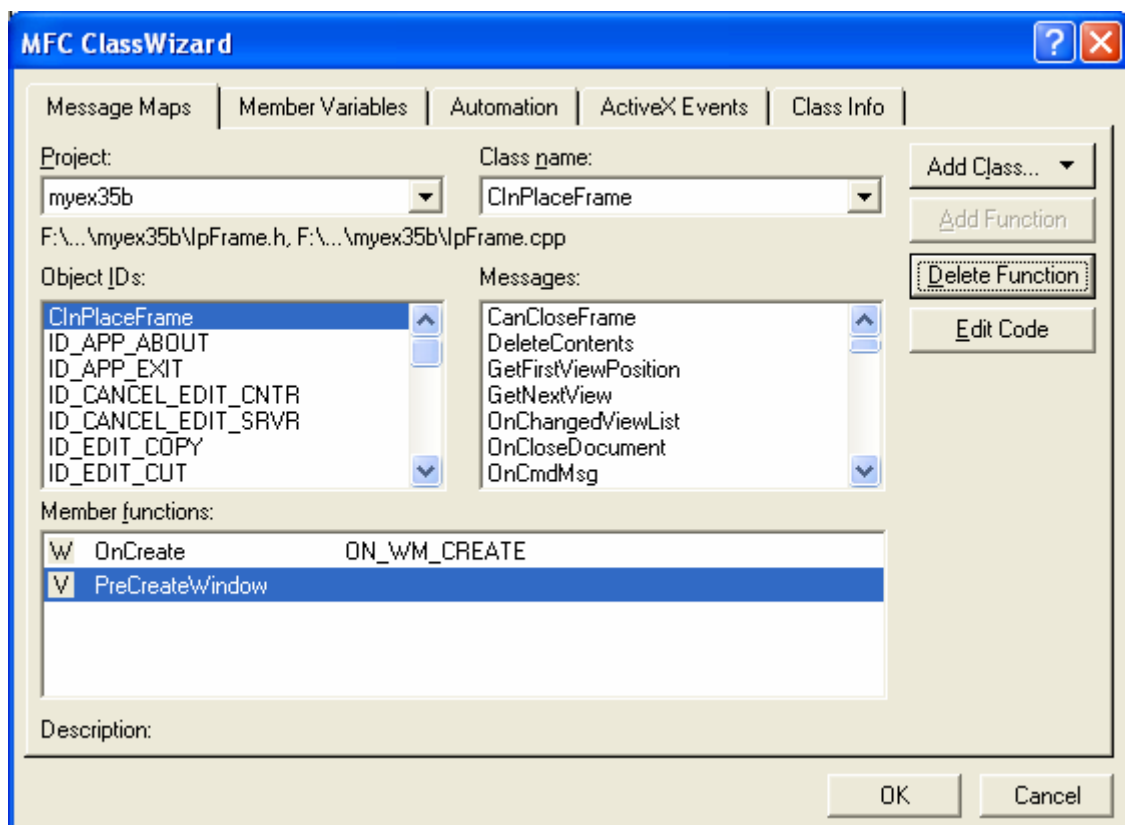


Figure 83: Deleting a message handler.

Delete also the following variable.

```
CToolBar    m_wndToolBar;

protected:
    COleDropTarget    m_dropTarget;
    COleResizeBar     m_wndResizeBar;

// Generated message map functions
protected:
```

Listing 16.

Edit/change the **ValForm.h** and **ValForm.cpp** code as shown below.

```
// ValForm.h

#ifndef _VALIDFORM
#define _VALIDFORM

#define WM_VALIDATE WM_USER + 5

class CValidForm : public CFormView
{
    DECLARE_DYNAMIC(CValidForm)
private:
    BOOL m_bValidationOn;
public:
    CValidForm(UINT ID);
    // override in derived dlg to perform validation
    virtual void ValidateDlgItem(CDataExchange* pDX, UINT ID);
```

```

    //{AFX_VIRTUAL(CValidForm)
protected:
virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
//}}AFX_VIRTUAL

    //{AFX_MSG(CValidForm)
afx_msg LONG OnValidate(UINT wParam, LONG lParam);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

#endif // _VALIDFORM

// ValForm.cpp : implementation file
//

#include "stdafx.h"
#include "myex35b.h"
#include "ValForm.h"

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

IMPLEMENT_DYNAMIC(CValidForm, CFormView)

BEGIN_MESSAGE_MAP(CValidForm, CFormView)
    //{AFX_MSG_MAP(CValidForm)
    ON_MESSAGE(WM_VALIDATE, OnValidate)
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

CValidForm::CValidForm(UINT ID) : CFormView(ID)
{
    TRACE("CValidForm ctor\n");
    m_bValidationOn = TRUE; // turn validation on
}

void CValidForm::ValidateDlgItem(CDataExchange* pDX, UINT ID)
{
    // return valid unless overridden in the form
    ASSERT(this);
    TRACE("CValidForm::ValidateDlgItem (should be overridden)\n");
    return;
}

BOOL CValidForm::OnCommand(WPARAM wParam, LPARAM lParam)
{
    // specific for WIN32 -- wParam/lParam processing different for WIN16
    TRACE("CValidForm::OnCommand, wParam = %x, lParam = %x\n", wParam, lParam);
    TRACE("m_bValidationOn = %d\n", m_bValidationOn);
    if(m_bValidationOn) { // might be a killfocus
        UINT notificationCode = (UINT) HIWORD( wParam );
        if((notificationCode == EN_KILLFOCUS) ||
           (notificationCode == LBN_KILLFOCUS) ||
           (notificationCode == CBN_KILLFOCUS) ) {
            CWnd* pFocus = CWnd::GetFocus(); // static function call
            // if we're changing focus to another control in the same form
            if( pFocus && (pFocus->GetParent() == this) ){
                if(pFocus->GetDlgCtrlID() != IDCANCEL) {
                    // and focus not in Cancel button
                    // validate AFTER drawing finished
                    BOOL rtn = PostMessage( WM_VALIDATE, wParam );
                    TRACE("posted message, rtn = %d\n", rtn);
                }
            }
        }
    }
}

```

```

    }
}
return CFormView::OnCommand(wParam, lParam); // pass it on
}

LONG CValidForm::OnValidate(UINT wParam, LONG lParam)
{
    TRACE("Entering CValidForm::OnValidate\n");
    CDataExchange dx(this, TRUE);
    m_bValidationOn = FALSE; // temporarily off
    UINT controlID = (UINT) LOWORD( wParam );
    try {
        ValidateDlgItem(&dx, controlID);
    }
    catch(CUserException* pUE) {
        pUE->Delete();
        TRACE("CValidForm caught the exception\n");
        // fall through -- user already alerted via message box
    }
    m_bValidationOn = TRUE;
    return 0; // goes no further
}

```

Listing 17.

### CMyex35bDoc Class

Use ClassView or manually add the following public variable.

```

public:
    int m_nPeriod;

```

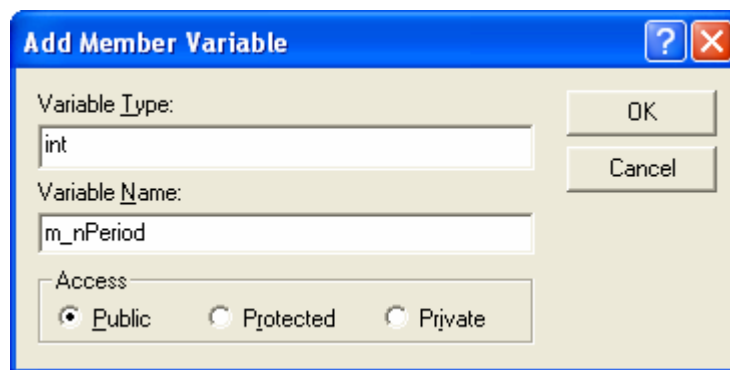


Figure 84: Adding variable.

```

// Implementation
public:
    int m_nPeriod;
    virtual ~CMyex35bDoc();
#ifdef _DEBUG

```

Listing 18.

Initialize the variable as shown below.

```

CMyex35bDoc::CMyex35bDoc()
{
    // Use OLE compound files
    EnableCompoundFile();
    // TODO: add one-time construction code here
    m_nPeriod = 12; // should initialize this some other way
}

```

Edit Serialize() as shown below.

```

void CMyex35bDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        ar << m_nPeriod;
    }
    else
    {
        // TODO: add loading code here
        ar >> m_nPeriod;
    }

    // Calling the base class COleServerDoc enables serialization
    // of the container document's COleClientItem objects.
    COleServerDoc::Serialize(ar);
}

// CMyex35bDoc serialization
void CMyex35bDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        ar << m_nPeriod;
    }
    else
    {
        // TODO: add loading code here
        ar >> m_nPeriod;
    }

    // Calling the base class COleServerDoc
    // of the container document's COleCl:
    COleServerDoc::Serialize(ar);
}

```

Listing 19.

### CMyex35bView Class

Add the `#include "ValForm.h"` directive and change the CView base class to CValidForm in `myex35bView.h`.

```

#include "ValForm.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMyex35bCntrItem;

class CMyex35bView : public CValidForm
{
    .
    .
    .

```

Listing 20.

Then, manually add the following virtual function.

```

virtual void ValidateDlgItem(CDataExchange* pDX, UINT ID);

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
virtual void ValidateDlgItem(CDataExchange* pDX, UINT ID);
//{{AFX_VIRTUAL(CMyex35bView)
public:
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

```

Listing 21.

Using ClassWizard, delete OnDraw() and add DoDataExchange() and OnPrint() virtual functions. Don't forget to delete the implementation part of the deleted function.

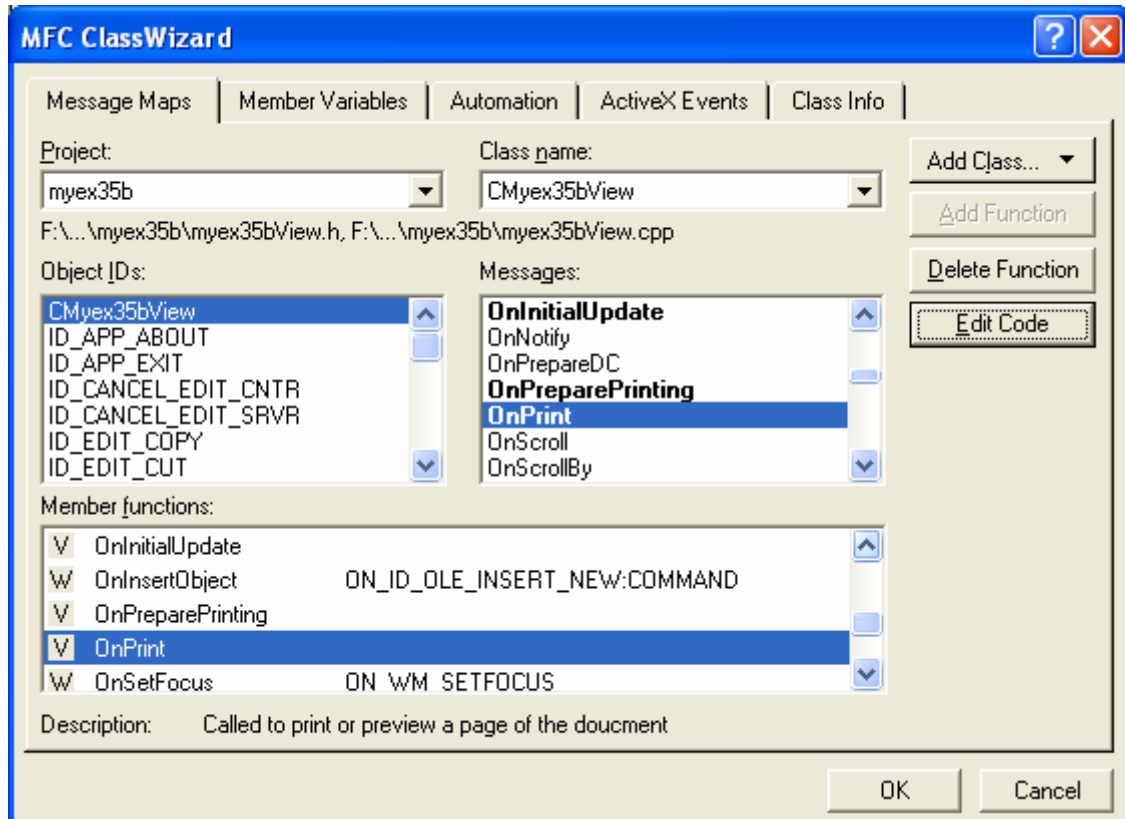


Figure 85: Adding and deleting virtual functions.

```
// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMyex35bView)
public:
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
virtual void OnInitialUpdate(); // called first time after construct
virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
virtual BOOL IsSelected(const CObject* pDocItem) const; // Container support
virtual void DoDataExchange(CDataExchange* pDX);
virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
//}}AFX_VIRTUAL
```

Listing 22.

Next, delete `OnDestroy()` and add the following message map functions. Don't forget to delete the implementation part of the deleted function.

ID	Message	Function
IDC_CANCEL	BN_CLICK	OnCancel()
IDC_SUBMIT	BN_CLICK	OnSubmit()
ID_SERVER_ADDRESS	COMMAND	OnServerAddress()
OnPostComplete() – Added manually if you cannot found it in ClassWizard.		
afx_msg LONG OnPostComplete(UINT wParam, LONG lParam);		

Table 8.

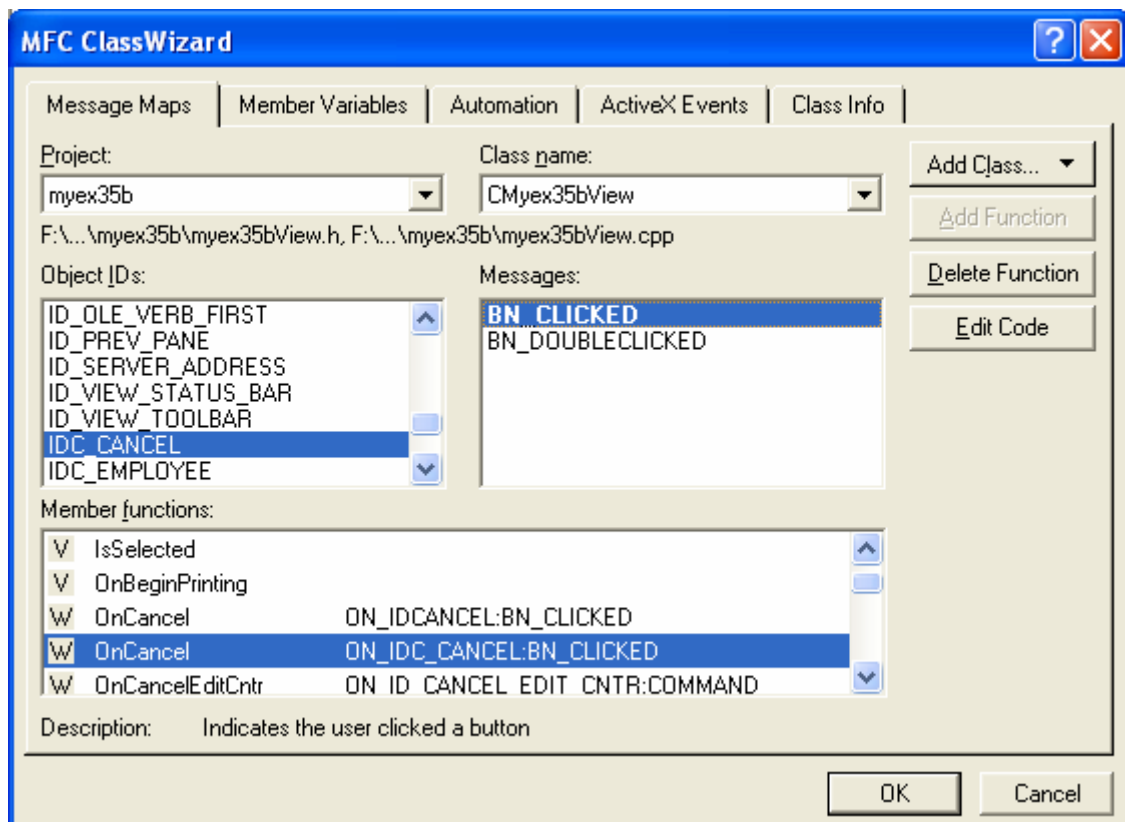


Figure 86: Adding message handlers.

```
// Generated message map functions
protected:
   //{{AFX_MSG(CMyex35bView)
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnInsertObject();
    afx_msg void OnCancelEditCntr();
    afx_msg void OnCancelEditSrvr();
    afx_msg void OnSubmit();
    afx_msg void OnServerAddress();
    afx_msg void OnCancel();
   //}}AFX_MSG
    afx_msg LONG OnPostComplete(UINT wParam, LONG lParam);
    DECLARE_MESSAGE_MAP()
};
```

Listing 23.

Manually, add the following AFX DATA.

```
public:
    //{{AFX_DATA(CMyex35bView)
    enum { IDD = IDD_MYEX35B_FORM };
    double      m_dHours;
    CString     m_strEmployee;
    int         m_nJob;
    int         m_nPeriod;
    //}}AFX_DATA

class CMyex35bView : public CValidForm
{
protected: // create from serialization only
    CMyex35bView();
    DECLARE_DYNCREATE(CMyex35bView)

public:
    //{{AFX_DATA(CMyex35bView)
    enum { IDD = IDD_MYEX35B_FORM };
    double      m_dHours;
    CString     m_strEmployee;
    int         m_nJob;
    int         m_nPeriod;
    //}}AFX_DATA
|
```

Listing 24.

Add the following `#include` directives to **myex35bView.cpp**

```
#include "AddrDialog.h"
#include "PostThread.h"

#include "myex35bDoc.h"
#include "CntrlItem.h"
#include "myex35bView.h"
#include "AddrDialog.h"
#include "PostThread.h"
|
#ifdef _DEBUG
```

Listing 25.



```

IMPLEMENT_DYNCREATE(CMyex35bView, CValidForm)

BEGIN_MESSAGE_MAP(CMyex35bView, CValidForm)
    //{{AFX_MSG_MAP(CMyex35bView)
    ON_WM_SETFOCUS()
    ON_WM_SIZE()
    ON_COMMAND(ID_OLE_INSERT_NEW, OnInsertObject)
    ON_COMMAND(ID_CANCEL_EDIT_CNTR, OnCancelEditCntr)
    ON_COMMAND(ID_CANCEL_EDIT_SRVR, OnCancelEditSrvr)
    ON_BN_CLICKED(IDC_SUBMIT, OnSubmit)
    ON_BN_CLICKED(IDCANCEL, OnCancel)
    ON_COMMAND(ID_SERVER_ADDRESS, OnServerAddress)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CValidForm::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CValidForm::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CValidForm::OnFilePrintPreview)
    ON_MESSAGE(WM_POSTCOMPLETE, OnPostComplete)
END_MESSAGE_MAP()

////////////////////////////////////
// CMyex35bView construction/destruction

CMyex35bView::CMyex35bView()
    : CValidForm(CMyex35bView::IDD)
{
    //{{AFX_DATA_INIT(CMyex35bView)
    m_dHours = 0.0;
    m_strEmployee = _T("");
    m_nJob = 0;
    m_nPeriod = 0;
    //}}AFX_DATA_INIT
    m_pSelection = NULL;
    // TODO: add construction code here
}

CMyex35bView::~CMyex35bView()
{
}

void CMyex35bView::DoDataExchange(CDataExchange* pDX)
{
    CValidForm::DoDataExchange(pDX);
    DDX_CBString(pDX, IDC_EMPLOYEE, m_strEmployee);
    DDV_MaxChars(pDX, m_strEmployee, 10);
    if(pDX->m_bSaveAndValidate && m_strEmployee.IsEmpty()) {
        AfxMessageBox("Must supply an employee name");
        pDX->Fail();
    }
    //{{AFX_DATA_MAP(CMyex35bView)
    DDX_Text(pDX, IDC_HOURS, m_dHours);
    DDV_MinMaxDouble(pDX, m_dHours, 0.1, 100.);
    DDX_Text(pDX, IDC_JOB, m_nJob);
    DDV_MinMaxInt(pDX, m_nJob, 1, 20);
    DDX_Text(pDX, IDC_PERIOD, m_nPeriod);
    //}}AFX_DATA_MAP
}

BOOL CMyex35bView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CValidForm::PreCreateWindow(cs);
}

```

```

}

void CMyex35bView::OnInitialUpdate()
{
    CMyex35bDoc* pDoc = GetDocument();
    m_nPeriod = pDoc->m_nPeriod;
    UpdateData(FALSE);

    // TODO: remove this code when final selection model code is written
    m_pSelection = NULL;    // initialize selection
}

////////////////////////////////////
// CMyex35bView printing

BOOL CMyex35bView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CMyex35bView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CMyex35bView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

void CMyex35bView::OnPrint(CDC* pDC, CPrintInfo*)
{
    // TODO: add code to print the controls
}

////////////////////////////////////
// OLE Client support and commands

BOOL CMyex35bView::IsSelected(const CObject* pDocItem) const
{
    // The implementation below is adequate if your selection consists of
    // only CMyex35bCntrItem objects. To handle different selection
    // mechanisms, the implementation here should be replaced.

    // TODO: implement this function that tests for a selected OLE client item

    return pDocItem == m_pSelection;
}

void CMyex35bView::OnInsertObject()
{
    // Invoke the standard Insert Object dialog box to obtain information
    // for new CMyex35bCntrItem object.
    COleInsertDialog dlg;
    if (dlg.DoModal() != IDOK)
        return;

    BeginWaitCursor();

    CMyex35bCntrItem* pItem = NULL;
    TRY
    {
        // Create new item connected to this document.
        CMyex35bDoc* pDoc = GetDocument();
    }
}

```

```

        ASSERT_VALID(pDoc);
        pItem = new CMyex35bCntrItem(pDoc);
        ASSERT_VALID(pItem);

        // Initialize the item from the dialog data.
        if (!dlg.CreateItem(pItem))
            AfxThrowMemoryException(); // any exception will do
        ASSERT_VALID(pItem);

        // If item created from class list (not from file) then launch
        // the server to edit the item.
        if (dlg.GetSelectionType() == ColeInsertDialog::createNewItem)
            pItem->DoVerb(OLEIVERB_SHOW, this);

        ASSERT_VALID(pItem);

        // As an arbitrary user interface design, this sets the selection
        // to the last item inserted.

        // TODO: re-implement selection as appropriate for your application

        m_pSelection = pItem; // set selection to last inserted item
        pDoc->UpdateAllViews(NULL);
    }
    CATCH(CException, e)
    {
        if (pItem != NULL)
        {
            ASSERT_VALID(pItem);
            pItem->Delete();
        }
        AfxMessageBox(IDP_FAILED_TO_CREATE);
    }
    END_CATCH

    EndWaitCursor();
}

// The following command handler provides the standard keyboard
// user interface to cancel an in-place editing session. Here,
// the container (not the server) causes the deactivation.
void CMyex35bView::OnCancelEditCntr()
{
    // Close any in-place active item on this view.
    ColeClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
    {
        pActiveItem->Close();
    }
    ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
}

// Special handling of OnSetFocus and OnSize are required for a container
// when an object is being edited in-place.
void CMyex35bView::OnSetFocus(CWnd* pOldWnd)
{
    ColeClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL &&
        pActiveItem->GetItemState() == ColeClientItem::activeUIState)
    {
        // need to set focus to this item if it is in the same view
        CWnd* pWnd = pActiveItem->GetInPlaceWindow();
        if (pWnd != NULL)
        {
            pWnd->SetFocus(); // don't call the base class
            return;
        }
    }
}

```

```

    }

    CValidForm::OnSetFocus(pOldWnd);
}

void CMyex35bView::OnSize(UINT nType, int cx, int cy)
{
    CValidForm::OnSize(nType, cx, cy);
    COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
        pActiveItem->SetItemRects();
}

////////////////////////////////////
// OLE Server support

// The following command handler provides the standard keyboard
// user interface to cancel an in-place editing session. Here,
// the server (not the container) causes the deactivation.
void CMyex35bView::OnCancelEditSrvr()
{
    GetDocument()->OnDeactivateUI(FALSE);
}

////////////////////////////////////
// CMyex35bView diagnostics

#ifdef _DEBUG
void CMyex35bView::AssertValid() const
{
    CValidForm::AssertValid();
}

void CMyex35bView::Dump(CDumpContext& dc) const
{
    CValidForm::Dump(dc);
}

CMyex35bDoc* CMyex35bView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMyex35bDoc));
    return (CMyex35bDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
// CMyex35bView message handlers

void CMyex35bView::OnCancel()
{
    CMyex35bDoc* pDoc = GetDocument();
    m_dHours = 0;
    m_strEmployee = "";
    m_nJob = 0;
    m_nPeriod = pDoc->m_nPeriod;
    UpdateData(FALSE);
}

void CMyex35bView::OnSubmit()
{
    if(UpdateData(TRUE) == TRUE) {
        GetDlgItem(IDC_SUBMIT)->EnableWindow(FALSE);
        CString strHours, strJob, strPeriod;
        strPeriod.Format("%d", m_nPeriod);
        strHours.Format("%3.2f", m_dHours);
    }
}

```

```

        strJob.Format("%d", m_nJob);
        g_strParameters = "Period=" + strPeriod + "&Employee=" +
            m_strEmployee + "&Hours=" + strHours +
            "&Job=" + strJob + "\r\n";
        TRACE("parameter string = %s", g_strParameters);
        AfxBeginThread(PostThreadProc, GetSafeHwnd(), THREAD_PRIORITY_NORMAL);
    }
}

// overridden CValidForm function
void CMyex35bView::ValidatedDlgItem(CDataExchange* pDX, UINT uID)
{
    ASSERT(this);
    TRACE("CMyex35bView::ValidatedDlgItem\n");
    switch (uID) {
    case IDC_EMPLOYEE:
        DDX_CBString(pDX, IDC_EMPLOYEE, m_strEmployee);
        // need custom DDV for empty string
        DDV_MaxChars(pDX, m_strEmployee, 10);
        if(m_strEmployee.IsEmpty()) {
            AfxMessageBox("Must supply an employee name");
            pDX->Fail();
        }
        break;
    case IDC_HOURS:
        DDX_Text(pDX, IDC_HOURS, m_dHours);
        DDV_MinMaxDouble(pDX, m_dHours, 0.1, 100.);
        break;
    case IDC_JOB:
        DDX_Text(pDX, IDC_JOB, m_nJob);
        DDV_MinMaxInt(pDX, m_nJob, 1, 20);
        break;
    default:
        break;
    }
    return;
}

void CMyex35bView::OnServerAddress()
{
    CAddrDialog dlg;
    dlg.m_strServerName = g_strServerName;
    dlg.m_strFile = g_strFile;
    if(dlg.DoModal() == IDOK) {
        g_strFile = dlg.m_strFile;
        g_strServerName = dlg.m_strServerName;
    }
}

// thread sends this message when it's done wParam = 1 for success
LONG CMyex35bView::OnPostComplete(UINT wParam, LONG lParam)
{
    TRACE("CMyex35bView::OnPostComplete - %d\n", wParam);
    if(wParam == 0) {
        AfxMessageBox("Server did not accept the transaction");
    }
    else OnCancel();
    GetDlgItem(IDC_SUBMIT)->EnableWindow(TRUE);
    return 0;
}

```

Listing 28.

Build and run MYEX35B. The following is the output. Activate the **Server Address** menu.

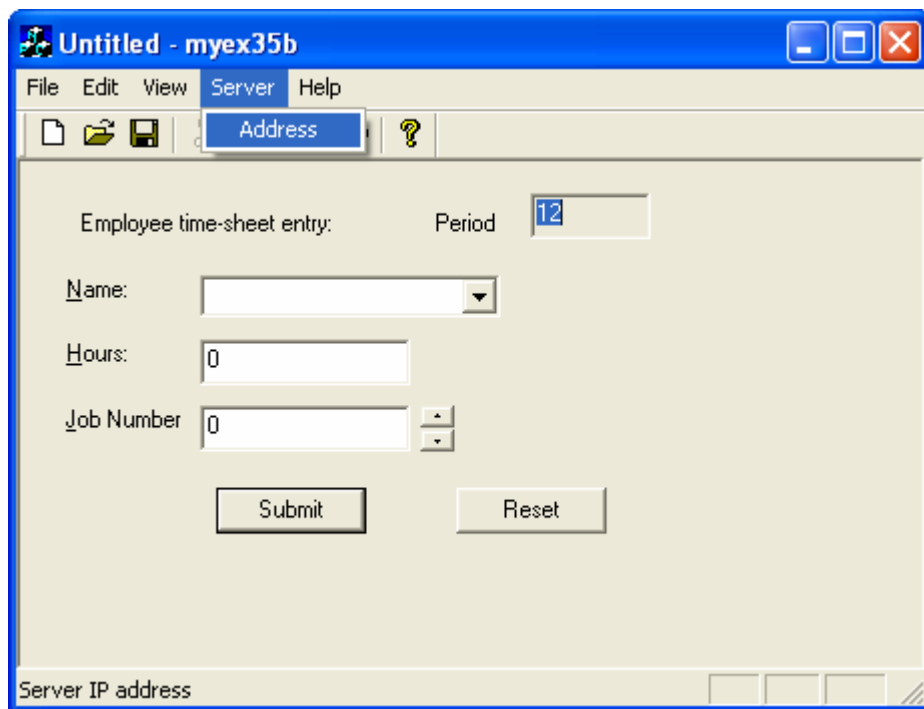


Figure 87: MYEX35B program output.

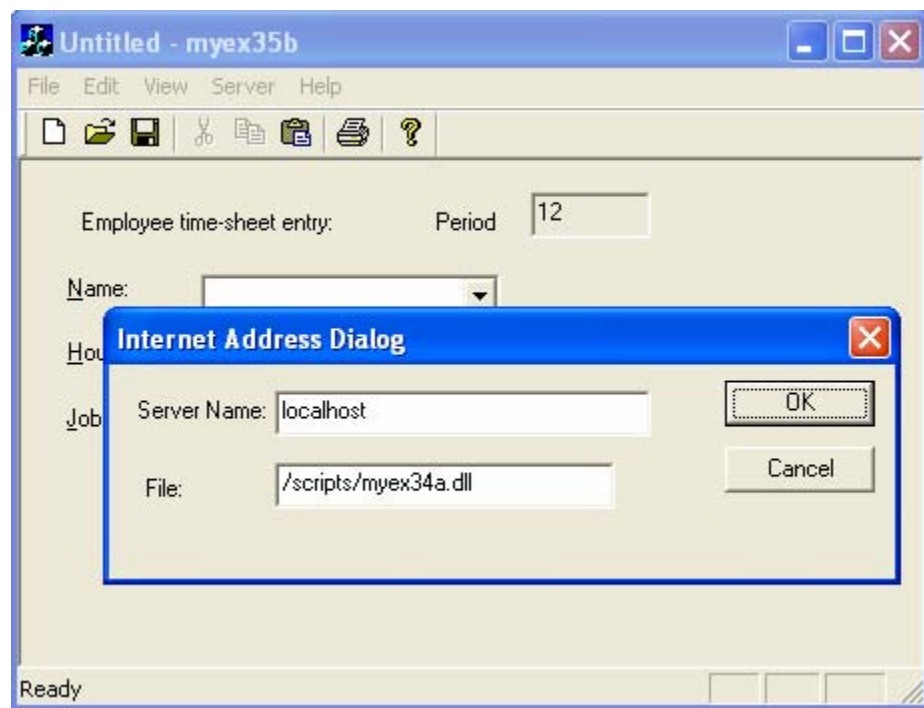


Figure 88: The **Address** sub menu displaying the server address and DLL file location settings. You can change this.

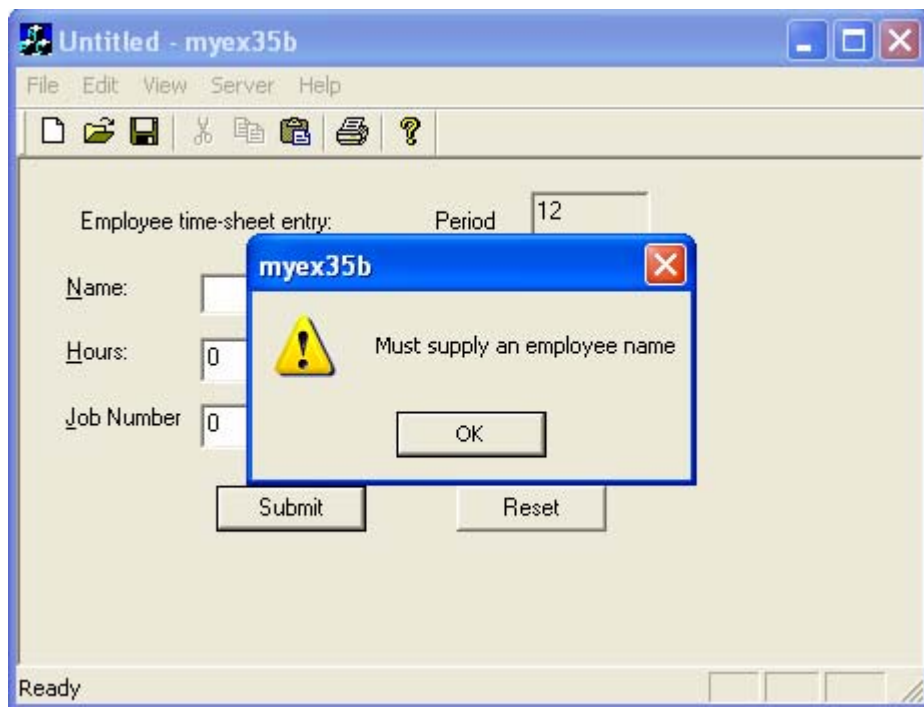


Figure 89: The validation message box for the **Name** field.

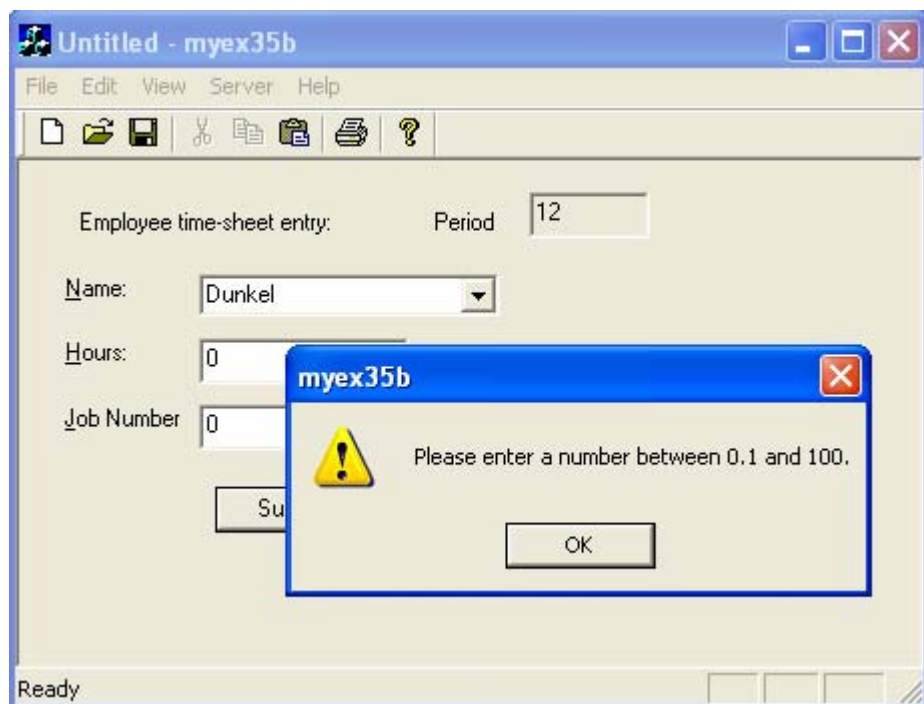


Figure 90: The validation message box for the **Hours** field.

Enter some data as shown below and click the **Submit** button.

test35b.35b - myex35b

File Edit View Server Help

Employee time-sheet entry: Period 12

Name: Matt Cullt

Hours: 9.5

Job Number 9

Submit Reset

Ready

Figure 91: Entering some data to myex35b form.

Untitled - myex35b

File Edit View Server Help

- New Ctrl+N
- Open... Ctrl+O
- Save Ctrl+S
- Save As...
- Print... Ctrl+P
- Print Preview
- Print Setup...
- Recent File
- Exit

Employee time-sheet entry: Period 12

Name: Matt Cullt

Hours: 9.5

Job Number 9

Submit Reset

Save the active document

Figure 92: Saving the form.

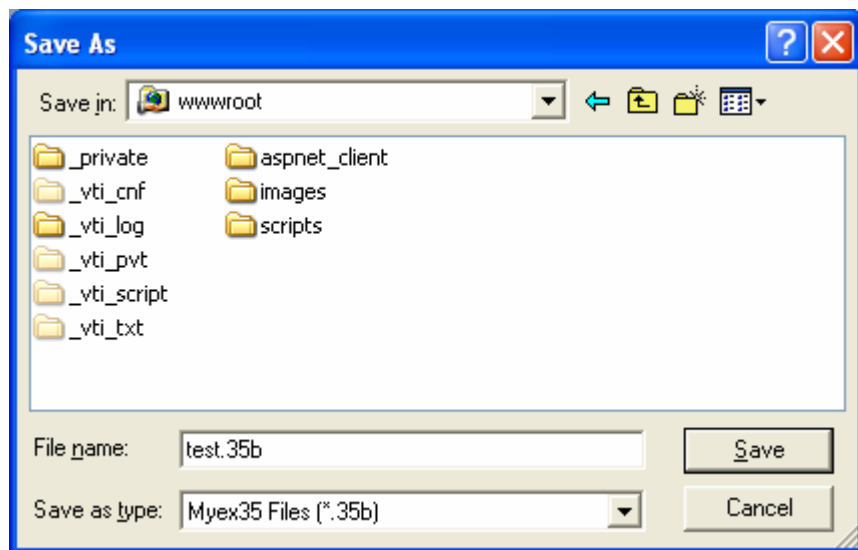


Figure 93: Saving the form, test.35b under **wwwroot** sub directory.

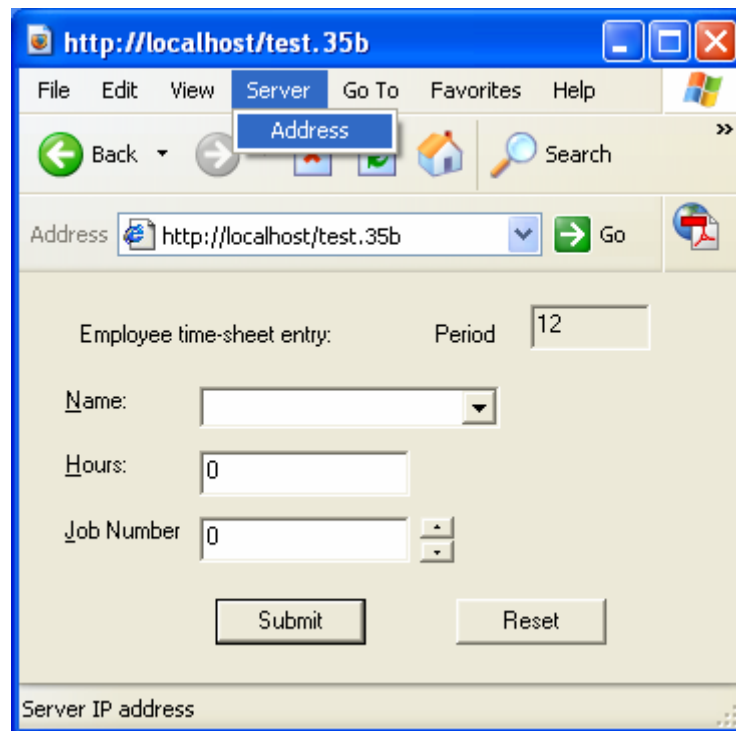


Figure 94: Opening **test.35b** in internet Explorer browser.

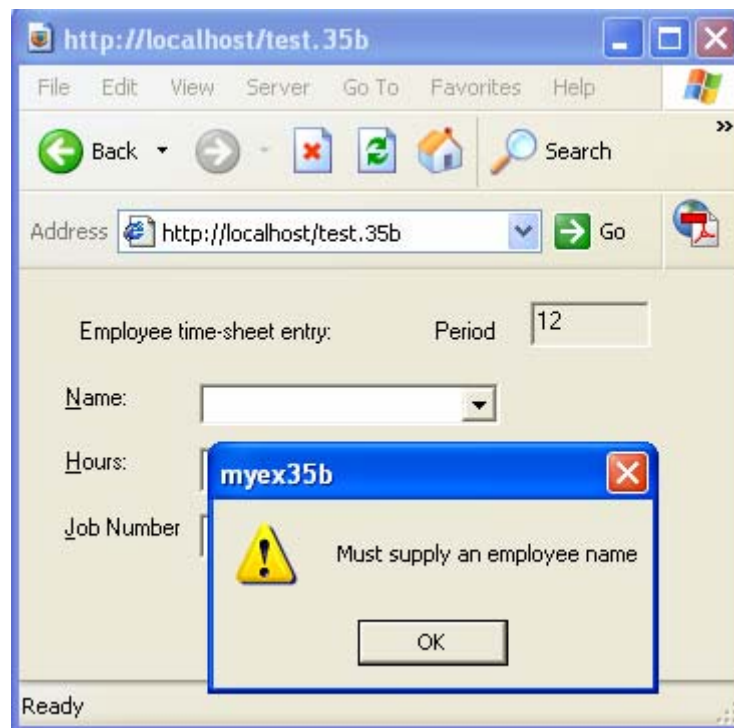


Figure 95: Testing the validation in the browser.

## The Story

### Field Validation in an MFC Form View

**Problem:** MFC's standard validation scheme validates data only when `CWnd::UpdateData(TRUE)` is called, usually when the user exits the dialog. Applications often need to validate data the moment the user leaves a field (edit control, list box, and so on). The problem is complex because Windows permits the user to freely jump between fields in any sequence by using the mouse. Ideally, standard MFC DDX/DDV (data exchange/validation) code should be used for field validation and the standard `DoDataExchange()` function should be called when the user finishes the transaction.

**Solution:** Derive your field validation form view classes from the class `CValidForm`, derived from `CFormView`, with this header:

```
// valform.h
#ifndef _VALIDFORM
#define _VALIDFORM

#define WM_VALIDATE WM_USER + 5

class CValidForm : public CFormView
{
    DECLARE_DYNAMIC(CValidForm)
private:
    BOOL m_bValidationOn;
public:
    CValidForm(UINT ID);
    // override in derived dlg to perform validation
    virtual void ValidateDlgItem(CDataExchange* pDX, UINT ID);
    //{{AFX_VIRTUAL(CValidForm)
protected:
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
    //}}AFX_VIRTUAL
}
```

```

    //{AFX_MSG(CValidForm)
    afx_msg LONG OnValidate(UINT wParam, LONG lParam);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#endif // _VALIDFORM

```

This class has one virtual function, `ValidateDlgItem()`, which accepts the control ID as the second parameter. The derived form view class implements this function to call the DDX/DDV functions for the appropriate field. Here is a sample `ValidateDlgItem()` implementation for a form view that has two numeric edit controls:

```

void CMyForm::ValidateDlgItem(CDataExchange* pDX, UINT uID)
{
    switch (uID) {
    case IDC_EDIT1:
        DDX_Text(pDX, IDC_EDIT1, m_nEdit1);
        DDV_MinMaxInt(pDX, m_nEdit1, 0, 10);
        break;
    case IDC_EDIT2:
        DDX_Text(pDX, IDC_EDIT2, m_nEdit2);
        DDV_MinMaxInt(pDX, m_nEdit2, 20, 30);
        break;
    default:
        break;
    }
}

```

Notice the similarity to the wizard-generated `DoDataExchange()` function:

```

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    //{AFX_DATA_MAP(CMyForm)
    DDX_Text(pDX, IDC_EDIT1, m_nEdit1);
    DDV_MinMaxInt(pDX, m_nEdit1, 0, 10);
    DDX_Text(pDX, IDC_EDIT2, m_nEdit2);
    DDV_MinMaxInt(pDX, m_nEdit2, 20, 30);
    //}}AFX_DATA_MAP
}

```

How does it work? The `CValidForm` class traps the user's attempt to move away from a control. When the user presses the Tab key or clicks on another control, the original control sends a killfocus command message (a control notification message) to the parent window, the exact format depending on the kind of control. An edit control, for example, sends an `EN_KILLFOCUS` command. When the form window receives this killfocus message, it invokes the DDX/DDV code that is necessary for that field, and if there's an error, the focus is set back to that field.

There are some complications, however. First, we want to allow the user to freely switch the focus to another application - we're not interested in trapping the killfocus message in that case. Next, we must be careful how we set the focus back to the control that produced the error. We can't just call `SetFocus()` in direct response to the killfocus message; instead we must allow the killfocus process to complete. We can achieve this by posting a user-defined `WM_VALIDATE` message back to the form window. The `WM_VALIDATE` handler calls our `ValidateDlgItem()` virtual function after the focus has been moved to the next field. Also, we must ignore the killfocus message that results when we switch back from the control that the user tried to select, and we must allow the `IDCANCEL` button to abort the transaction without validation.

Most of the work here is done in the view's virtual `OnCommand()` handler, which is called for all control notification messages. We could, of course, individually map each control's killfocus message in our derived form view class, but that would be too much work. Here is the `OnCommand()` handler:

```

BOOL CValidForm::OnCommand(WPARAM wParam, LPARAM lParam)
{
    // specific for WIN32 - wParam/lParam processing different for

```

```

// WIN16
TRACE("CValidForm::OnCommand, wParam = %x, lParam = %x\n",
      wParam, lParam);
TRACE("m_bValidationOn = %d\n", m_bValidationOn);
if(m_bValidationOn) { // might be a killfocus
    UINT notificationCode = (UINT) HIWORD(wParam);
    if((notificationCode == EN_KILLFOCUS) ||
        (notificationCode == LBN_KILLFOCUS) ||
        (notificationCode == CBN_KILLFOCUS)) {
        CWnd* pFocus = CWnd::GetFocus(); // static function call
        // if we're changing focus to another control in the
        // same form
        if(pFocus && (pFocus->GetParent() == this)) {
            if(pFocus->GetDlgCtrlID() != IDCANCEL) {
                // and focus not in Cancel button
                // validate AFTER drawing finished
                BOOL rtn = PostMessage(WM_VALIDATE, wParam);
                TRACE("posted message, rtn = %d\n", rtn);
            }
        }
    }
}
return CFormView::OnCommand(wParam, lParam); // pass it on
}

```

Note that `m_bValidationOn` is a Boolean data member in `CValidForm`. Finally, here is the `OnValidate()` message handler, mapped to the user-defined `WM_VALIDATE` message:

```

LONG CValidForm::OnValidate(UINT wParam, LONG lParam)
{
    TRACE("Entering CValidForm::OnValidate\n");
    CDataExchange dx(this, TRUE);
    m_bValidationOn = FALSE; // temporarily off
    UINT controlId = (UINT) LOWORD(wParam);
    try {
        ValidateDlgItem(&dx, controlId);
    }
    catch(CUserException* pUE)
    {
        pUE->Delete();
        TRACE("CValidForm caught the exception\n");
        // fall through - user already alerted via message box
    }
    m_bValidationOn = TRUE;
    return 0; // goes no further
}

```

Instructions for use:

1. Add `valform.h` and **`valform.cpp`** to your project.
2. Insert the following statement in your view class header file:

```
#include "valform.h"
```

3. Change your view class base class from `CFormView` to `CValidForm`.
4. Override `ValidateDlgItem()` for your form's controls as shown above.

That's all. For dialogs, follow the same steps, but use **`valid.h`** and **`valid.cpp`**. Derive your dialog class from `CValidDialog` instead of from `CDialog`.

## Generating POST Requests Under Program Control

The heart of the MYEX35B program is a worker thread that generates a POST request and sends it to a remote server. The server doesn't care whether the POST request came from an HTML form or from your program. It could process the POST request with an ISAPI DLL, with a PERL script, or with a Common Gateway Interface (CGI) executable program. Here's what the server receives when the user clicks the MYEX35B **Submit** button:

```
POST scripts/myex34a.dll?ProcessTimesheet HTTP/1.0
(request headers)
(blank line)
Period=12&Name=Dunkel&Hours=6.5&Job=5
```

And here's the thread code from **PostThread.cpp**:

```
// PostThread.cpp (uses MFC WinInet calls)

#include <stdafx.h>
#include "PostThread.h"

#define MAXBUF 50000

CString g_strFile = "/scripts/ex35a.dll";
CString g_strServerName = "localhost";
CString g_strParameters;

UINT PostThreadProc(LPVOID pParam)
{
    CInternetSession session;
    CHttpConnection* pConnection = NULL;
    CHttpFile* pFile1 = NULL;
    char* buffer = new char[MAXBUF];
    UINT nBytesRead = 0;
    DWORD dwStatusCode;
    BOOL bOkStatus = FALSE;
    try {
        pConnection = session.GetHttpConnection(g_strServerName,
            (INTERNET_PORT) 80);
        pFile1 = pConnection->OpenRequest(0, g_strFile +
            "?ProcessTimesheet", // POST request
            NULL, 1, NULL, NULL, INTERNET_FLAG_KEEP_CONNECTION |
            INTERNET_FLAG_RELOAD); // no cache
        pFile1->SendRequest(NULL, 0,
            (LPVOID) (const char*) g_strParameters,
            g_strParameters.GetLength());
        pFile1->QueryInfoStatusCode(dwStatusCode);
        if(dwStatusCode == 200) { // OK status
            // doesn't matter what came back from server - we're looking
            // for OK status
            bOkStatus = TRUE;
            nBytesRead = pFile1->Read(buffer, MAXBUF - 1);
            buffer[nBytesRead] = '\0'; // necessary for TRACE
            TRACE(buffer);
            TRACE("\n");
        }
    }
    catch(CInternetException* pe) {
        char text[100];
        pe->GetErrorMessage(text, 99);
    }
}
```

```

        TRACE("WinInet exception %s\n", text);
        pe->Delete();
    }
    if(pFile1) delete pFile1; // does the close – prints a warning
    if(pConnection) delete pConnection; // Why does it print a warning?
    delete [] buffer;
    ::PostMessage((HWND) pParam, WM_POSTCOMPLETE, (WPARAM) bOkStatus, 0);
    return 0;
}

```

The main thread assembles the `g_strParameters` string based on what the user typed, and the worker thread sends the POST request using the `CHttpFile::SendRequest` call. The `tQueryInfoStatusCode` to find out if the server sent back an OK response. Before exiting, the thread posts a message to the main thread, using the `bOkStatus` value in `wParam` to indicate success or failure.

## The MYEX35B View Class

The `CMYEX35BView` class is derived from `CValidForm`, as described in "Field Validation in an MFC Form View". `CMYEX35BView` collects user data and starts the post thread when the user clicks the **Submit** button after all fields have been successfully validated. Field validation is independent of the internet application. You could use `CValidForm` in any MFC form view application. Here is the code for the overridden or the overridden `ValidateDlgItem()` member function, which is called whenever the user moves from one control to another:

```

void CMyex35bView::ValidateDlgItem(CDataExchange* pDX, UINT uID)
{
    ASSERT(this);
    TRACE("CMyex35bView::ValidateDlgItem\n");
    switch (uID) {
    case IDC_EMPLOYEE:
        DDX_CBString(pDX, IDC_EMPLOYEE, m_strEmployee);
        // need custom DDV for empty string
        DDV_MaxChars(pDX, m_strEmployee, 10);
        if(m_strEmployee.IsEmpty()) {
            AfxMessageBox("Must supply an employee name");
            pDX->Fail();
        }
        break;
    case IDC_HOURS:
        DDX_Text(pDX, IDC_HOURS, m_dHours);
        DDV_MinMaxDouble(pDX, m_dHours, 0.1, 100.);
        break;
    case IDC_JOB:
        DDX_Text(pDX, IDC_JOB, m_nJob);
        DDV_MinMaxInt(pDX, m_nJob, 1, 20);
        break;
    default:
        break;
    }
    return;
}

```

The `OnSubmit()` member function is called when the user clicks the **Submit** button. `CWnd::UpdateData` returns `TRUE` only when all the fields have been successfully validated. At that point, the function disables the **Submit** button, formats `g_strParameters`, and starts the post thread.

```

void CMyex35bView::OnSubmit()
{
    if(UpdateData(TRUE) == TRUE) {

```

```

        GetDlgItem(IDC_SUBMIT)->EnableWindow(FALSE);
        CString strHours, strJob, strPeriod;
        strPeriod.Format("%d", m_nPeriod);
        strHours.Format("%3.2f", m_dHours);
        strJob.Format("%d", m_nJob);
        g_strParameters = "Period=" + strPeriod + "&Employee=" +
            m_strEmployee + "&Hours=" + strHours + "&Job=" +
            strJob + "\r\n";
        TRACE("parameter string = %s", g_strParameters);
        AfxBeginThread(PostThreadProc, GetSafeHwnd(),
            THREAD_PRIORITY_NORMAL);
    }
}

```

The OnCancel ( ) member function is called when the user clicks the **Reset** button. The CValidForm logic requires that the button's control ID be IDCANCEL.

```

void CMyex35bView::OnCancel()
{
    CMyex35bDoc* pDoc = GetDocument();
    m_dHours = 0;
    m_strEmployee = "";
    m_nJob = 0;
    m_nPeriod = pDoc->m_nPeriod;
    UpdateData(FALSE);
}

```

The OnPostComplete ( ) handler is called in response to the user-defined WM\_POSTCOMPLETE message sent by the post thread:

```

LONG CMyex35bView::OnPostComplete(UINT wParam, LONG lParam)
{
    TRACE("CMyex35bView::OnPostComplete - %d\n", wParam);
    if(wParam == 0) {
        AfxMessageBox("Server did not accept the transaction");
    }
    else
        OnCancel();
    GetDlgItem(IDC_SUBMIT)->EnableWindow(TRUE);
    return 0;
}

```

This function displays a message box if the server didn't send an OK response. It then enables the **Submit** button, allowing the user to post another time-sheet entry.

## Building and Testing MYEX35B

Build the MYEX35B project, and then run it once in stand-alone mode to register it and to save a document file sample called **test.35b** in your WWW root directory. Make sure the MYEX34A DLL (**myex34a.dll**) is available in the **scripts** directory (with execute permission) because that DLL contains an ISAPI function, ProcessTimesheet ( ), which handles the server end of the POST request. Be sure that you have IIS or some other ISAPI-capable server running. Now run Internet Explorer and load **test.35b** from your server. The MYEX35B program should be running in the Browser window, and you should be able to enter time-sheet transactions.

## ActiveX Document Servers vs. VB Script

It's possible to insert VB Script (or JavaScript) code into an HTML file. We're not experts on VB Script, but we've seen some sample code. You could probably duplicate the MYEX35B time-sheet application with VB Script, but you would be limited to the standard HTML input elements. It would be interesting to see how a VB Script programmer would

solve the problem. In any case, you're a C++ programmer, not a Visual Basic programmer, so you might as well stick to what you know.

## Going Further with ActiveX Document Servers

MYEX35A used a worker thread to read a text file from an Internet server. It used the MFC WinInet classes, and it assumed that a standard HTTP server was available. An ActiveX document server could just as easily make Winsock calls using the `CBlockingSocket` class from [Module 32](#). That would imply that you were going beyond the HTTP and FTP protocols. You could, for example, write a custom internet server program that listened on port 81. That server could run concurrently with IIS if necessary. Your ActiveX document server could use a custom TCP/IP protocol to get binary data from an open socket. The server could use this data to update its window in real-time, or it could send the data to another device, such as a sound card.

-----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](#).