Module 17: MFC Programs Without Document or View Classes

Program examples compiled using Visual C++ 6.0 (MFC 6.0) compiler on Windows XP Pro machine with Service Pack 2. Topics and sub topics for this Tutorial are listed below:

MFC Programs Without Document or View Classes

The document-view architecture is useful for many applications, but sometimes a simpler program structure is sufficient. This module illustrates three applications:

1. A dialog-based program.
2. A Single Document Interface (SDI) program, and
3. A Multiple Document Interface (MDI) program.

None of these programs uses document, view, or document-template classes, but they all use command routing and some other MFC library features. In Visual C++ 6.0, you can create all three types of applications using AppWizard. In each example, we'll look at how AppWizard generates code that doesn't rely on the document-view architecture and show you how to add your own code to each example.

The MYMFC23A Example: A Dialog-Based Application

For many applications, a dialog provides a sufficient user interface. The dialog window immediately appears when the user starts the application. The user can minimize the dialog window, and as long as the dialog is not system modal, the user can freely switch to other applications.
In this example, the dialog functions as a simple calculator, as shown in Figure 1. ClassWizard takes charge of defining the class data members and generating the DDX (Dialog Data Exchange) function calls, everything but the coding of the compute function. The application's resource script, **mymfc23A.rc**, defines an icon as well as the dialog.

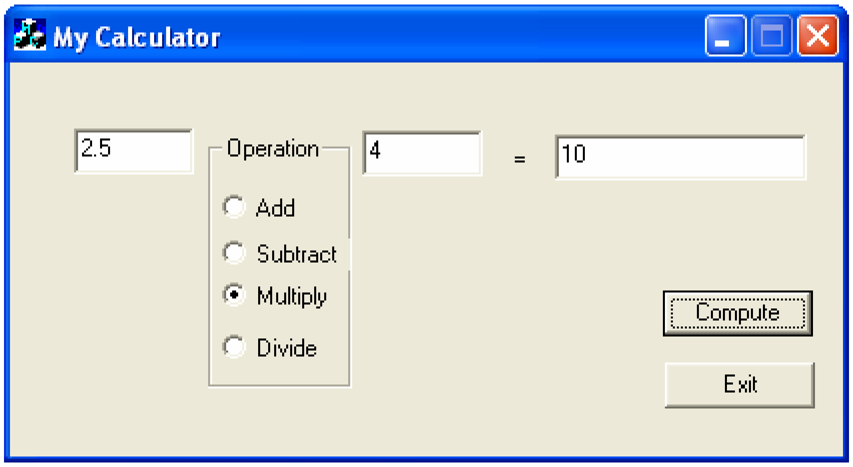
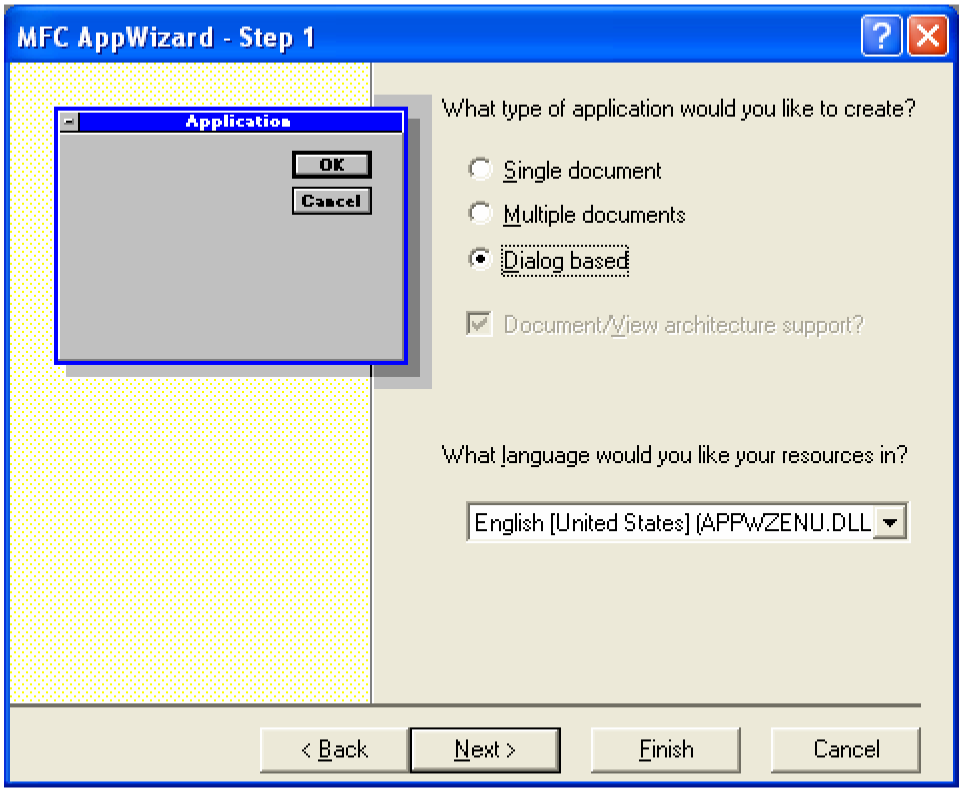


Figure 1: The MYMFC23A Calculator dialog.

AppWizard gives you the option of generating a dialog-based application. Here are the steps for building the MYMFC23A example:

Run AppWizard to produce \mfcproject\mymfc23A. Select the **Dialog Based** option in the AppWizard Step 1 dialog, as shown here.


MFC AppWizard - Step 1

What type of application would you like to create?

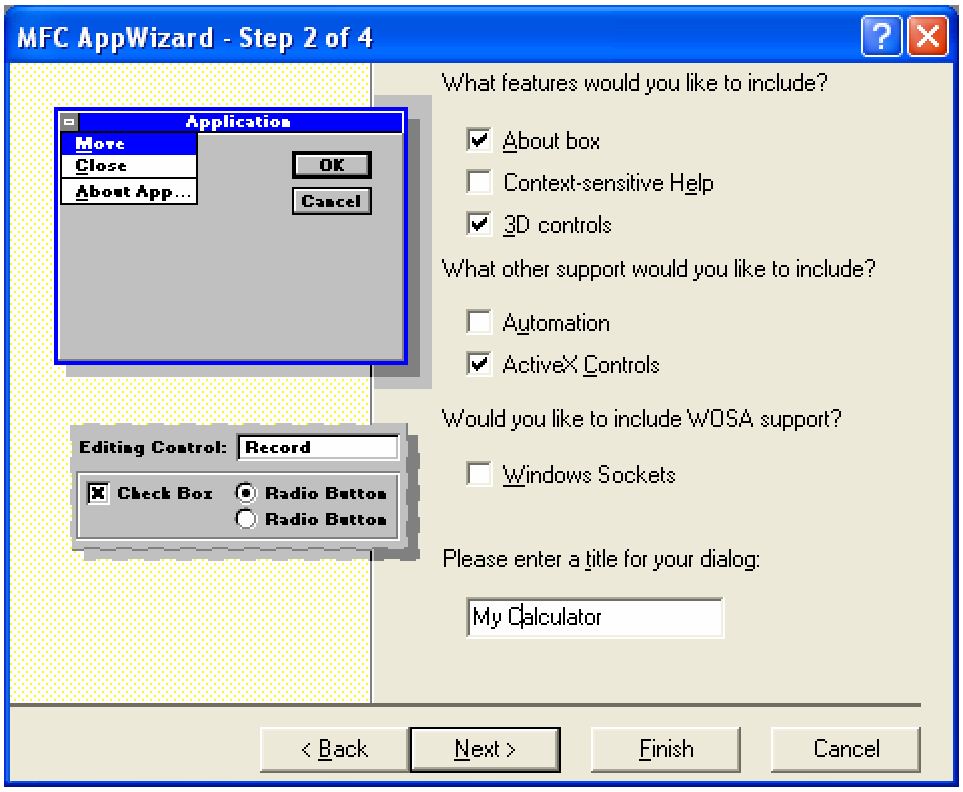
- ○ Single document
- ○ Multiple documents
- ● Dialog based
- ☑ Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 2: Dialog based MFC project.

In the next dialog, enter **My Calculator** as the dialog title. Accept the default for other options.


MFC AppWizard - Step 2 of 4

What features would you like to include?

- ☑ About box
- ☐ Context-sensitive Help
- ☑ 3D controls

What other support would you like to include?

- ☐ Automation
- ☑ ActiveX Controls

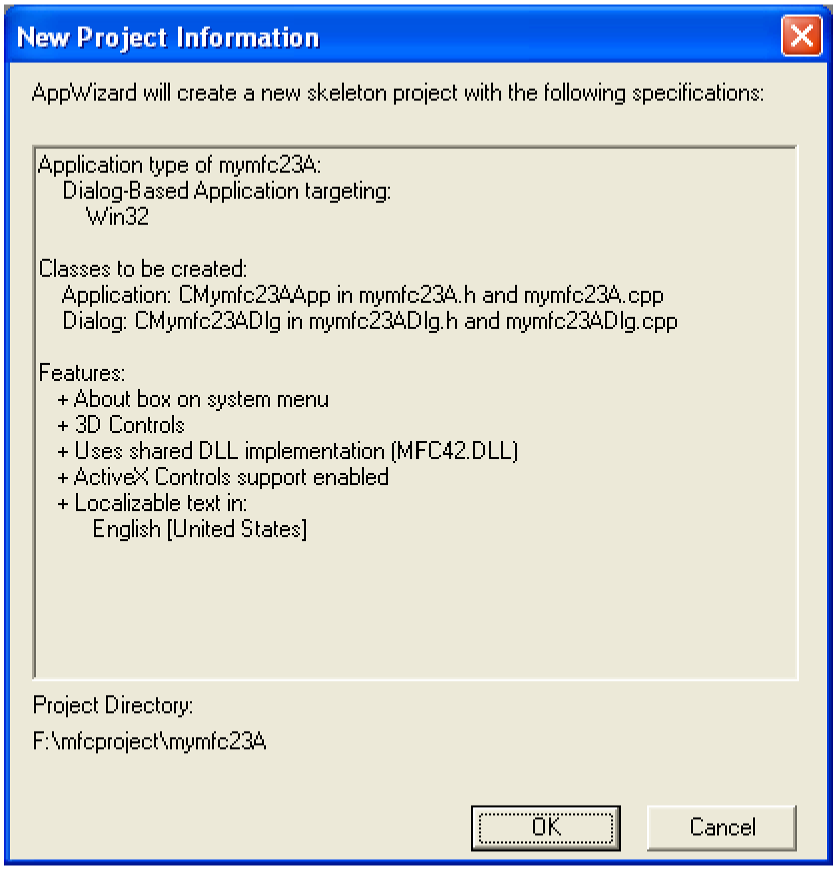
Would you like to include WOSA support?

- ☐ Windows Sockets

Please enter a title for your dialog:

My Calculator

Figure 3: Step 2 of 4 AppWizard, dialog based MFC program, entering the dialog's title.



AppWizard will create a new skeleton project with the following specifications:

```
Application type of mymfc23A:
    Dialog-Based Application targeting:
        Win32

Classes to be created:
    Application: CMymfc23AApp in mymfc23A.h and mymfc23A.cpp
    Dialog: CMymfc23ADlg in mymfc23ADlg.h and mymfc23ADlg.cpp

Features:
   + About box on system menu
   + 3D Controls
   + Uses shared DLL implementation (MFC42.DLL)
   + ActiveX Controls support enabled
   + Localizable text in:
         English [United States]
```

Project Directory:

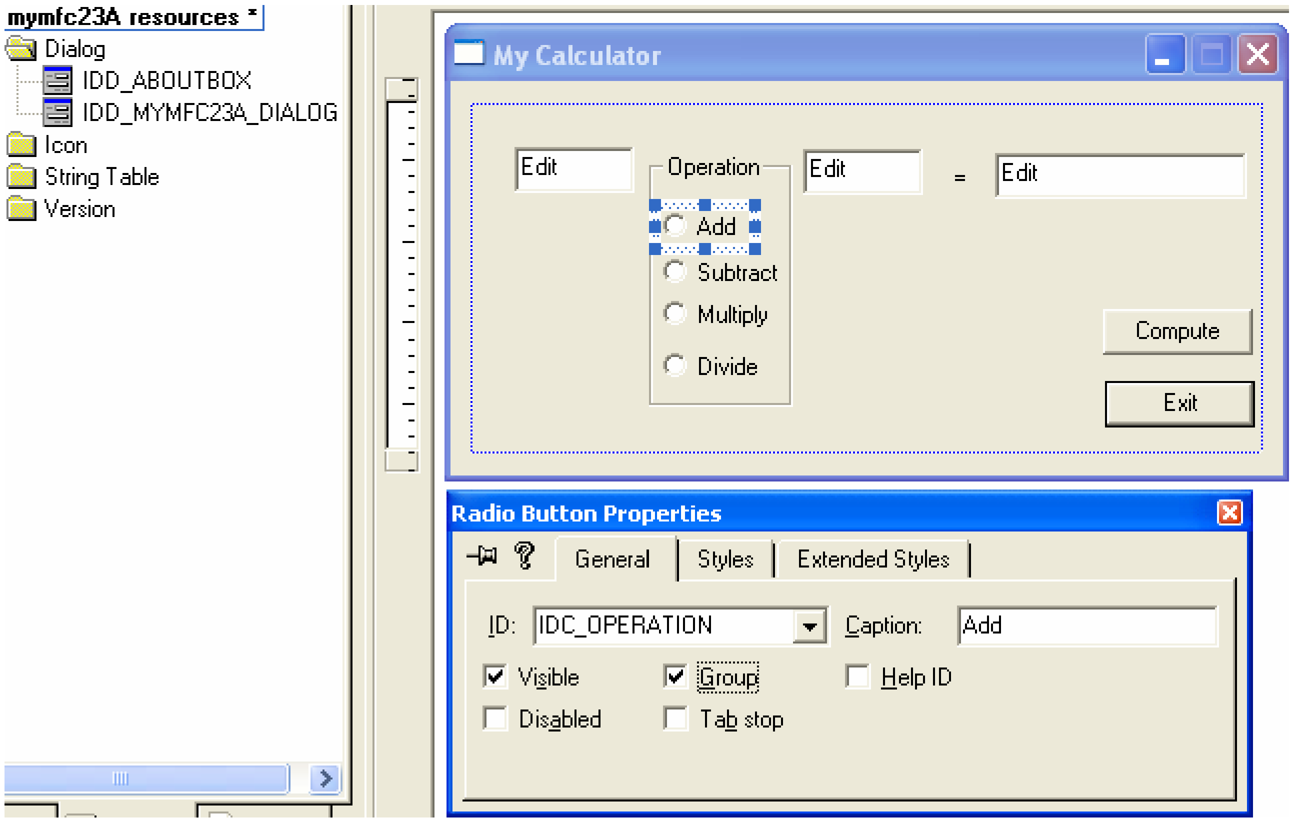
F:\mfcproject\mymfc23A

Figure 4: MYMFC23A, dialog based project summary.

Edit the `IDD_MYMFC23A_DIALOG` resource. Refer to Figure 23-1 as a guide. Use the dialog editor to assign IDs to the controls shown in the table below. For `IDC_OPERATION`, select the **Group** option.

Control	**ID**
Left operand edit control	`IDC_LEFT`
Right operand edit control	`IDC_RIGHT`
Result edit control	`IDC_RESULT`
First radio button (group property set)	`IDC_OPERATION`
Compute pushbutton	`IDC_COMPUTE`
Exit push button	This is the **OK** button

Table 1


mymfc23A resources *

Dialog

IDD_ABOUTBOX

IDD_MYMFC23A_DIALOG

Icon

String Table

Version

My Calculator

Edit | Operation | Edit | = | Edit

Add

Subtract

Multiply

Divide

Compute

Exit

Radio Button Properties

General | Styles | Extended Styles

ID: IDC_OPERATION

Caption: Add

[✓] Visible

[✓] Group

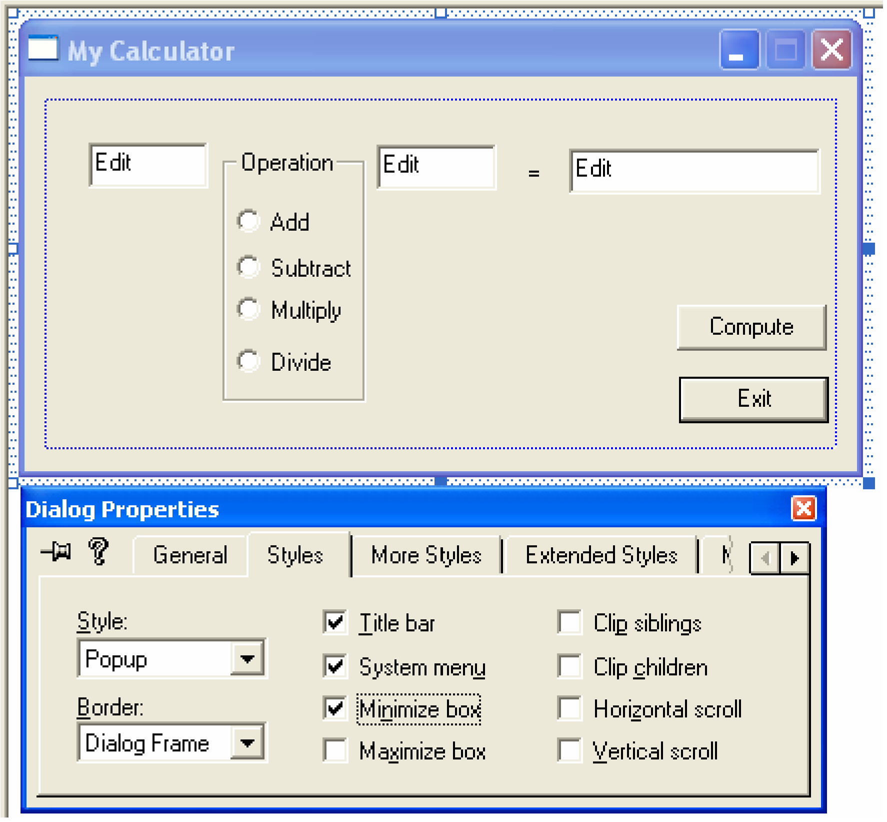
[ ] Help ID

[ ] Disabled

[ ] Tab stop

Figure 5: Adding and modifying dialog and its control properties.

Open the **Properties** dialog box and click on the **Styles** tab. Select the **System Menu** and **Minimize Box** options.


My Calculator

Edit | Operation | Edit | = | Edit

Add

Subtract

Multiply

Divide

Compute

Exit

Dialog Properties

General | Styles | More Styles | Extended Styles

Style: Popup

Border: Dialog Frame

[✓] Title bar

[✓] System menu

[✓] Minimize box

[ ] Maximize box

[ ] Clip siblings

[ ] Clip children

[ ] Horizontal scroll

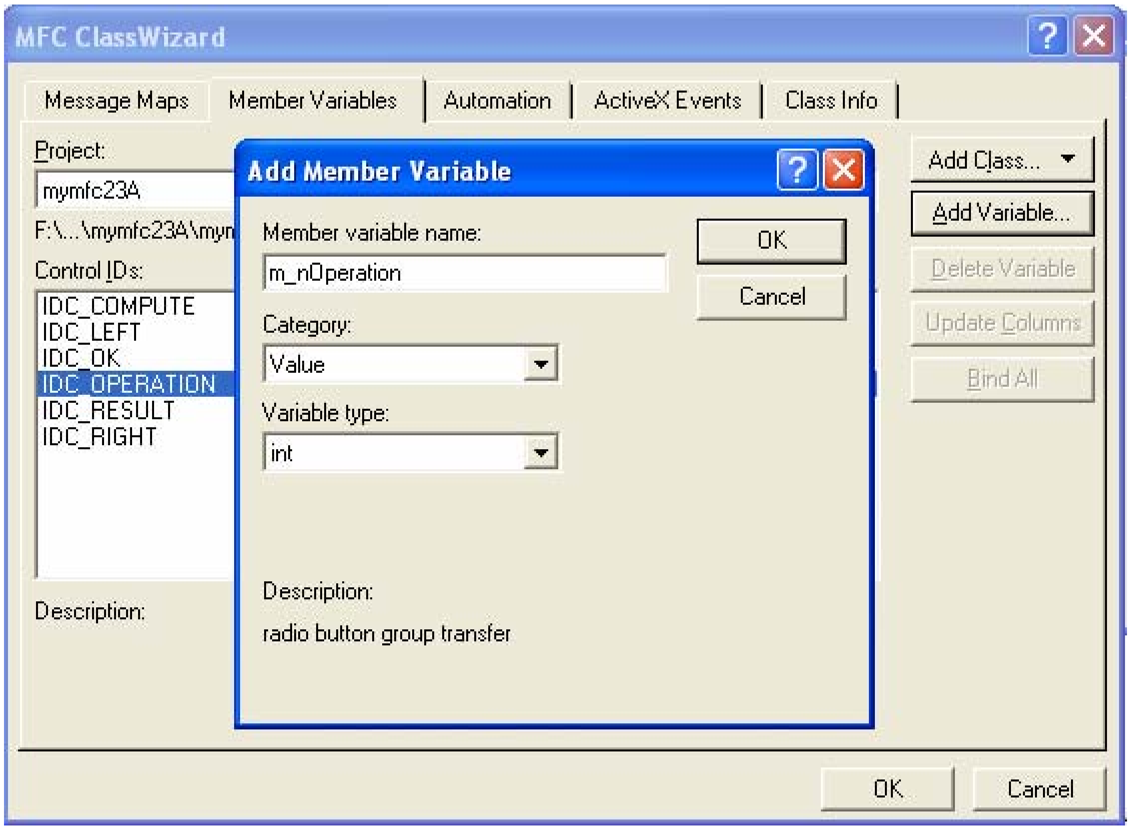
[ ] Vertical scroll

Figure 6: Modifying dialog properties.

Use ClassWizard to add member variables and a command handler. AppWizard has already generated a class `CMymfc23ADlg`. Add the following data members.

Control ID	Member Variable	Type
IDC_LEFT	m_dLeft	double
IDC_RIGHT	m_dRight	double
IDC_RESULT	m_dResult	double
IDC_OPERATION	m_nOperation	int

Table 2



Member variable name: m_nOperation

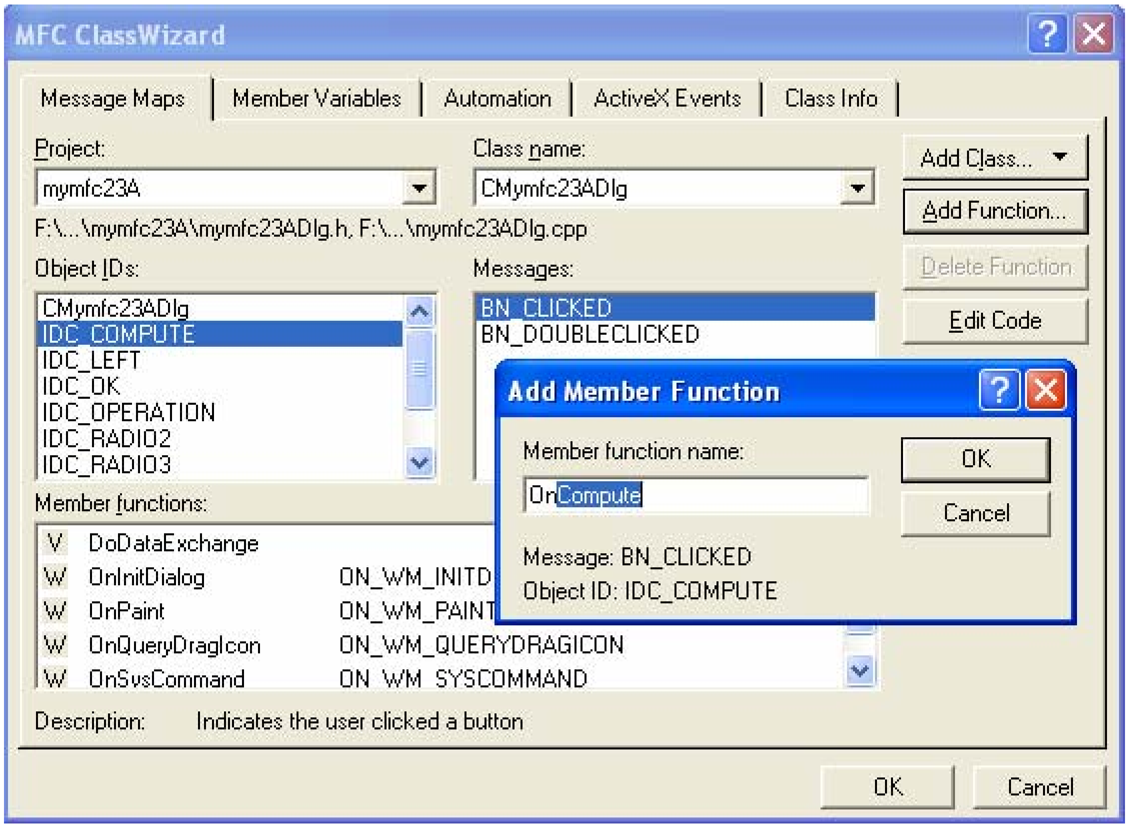
Category: Value

Variable type: int

Description: radio button group transfer

Figure 7: Using ClassWizard to add member variables.

Add the message handler `OnCompute()` for the `IDC_COMPUTE` button and `IDC_OK`.



Project: mymfc23A

Class name: CMymfc23ADlg

F:\...\mymfc23A\mymfc23ADlg.h, F:\...\mymfc23ADlg.cpp

Object IDs:

```
CMymfc23ADlg
IDC_COMPUTE
IDC_LEFT
IDC_OK
IDC_OPERATION
IDC_RADIO2
IDC_RADIO3
```

Messages:

```
BN_CLICKED
BN_DOUBLECLICKED
```

Member functions:

```
V  DoDataExchange
W  OnInitDialog      ON_WM_INITD
W  OnPaint           ON_WM_PAINT
W  OnQueryDragIcon   ON_WM_QUERYDRAGICON
W  OnSysCommand      ON_WM_SYSCOMMAND
```

Description: Indicates the user clicked a button

Add Member Function

Member function name: OnCompute

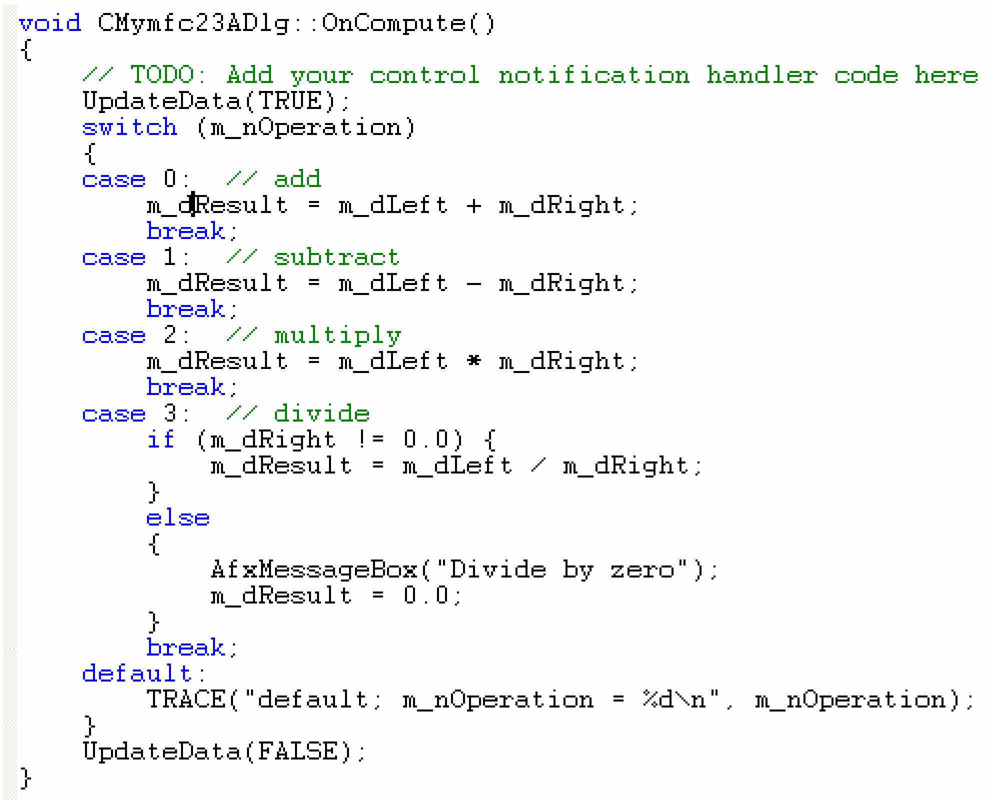
Message: BN_CLICKED

Object ID: IDC_COMPUTE

Figure 8: Message mapping and its handler.

Code the `OnCompute()` member function in the **mymfc23ADlg.cpp** file. Add the following code:

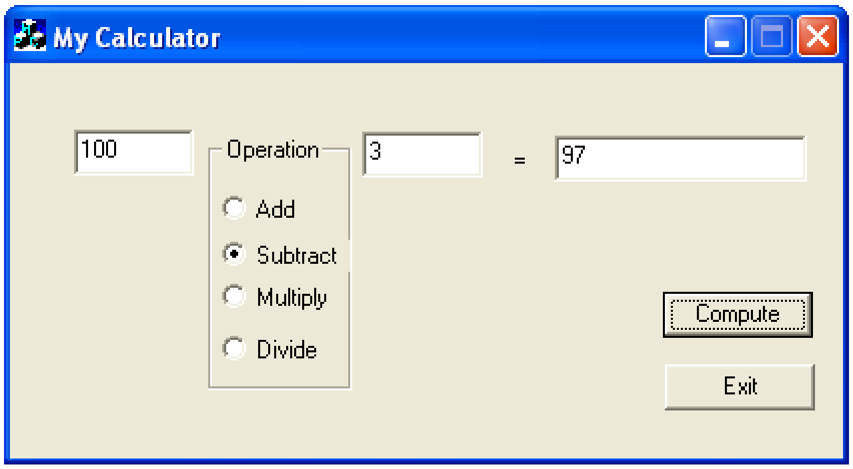
```
void CMymfc23ADlg::OnCompute()
{
    UpdateData(TRUE);
    switch (m_nOperation)
    {
    case 0:  // add
        m_dResult = m_dLeft + m_dRight;
        break;
    case 1:  // subtract
        m_dResult = m_dLeft - m_dRight;
        break;
    case 2:  // multiply
        m_dResult = m_dLeft * m_dRight;
        break;
    case 3:  // divide
        if (m_dRight != 0.0) {
            m_dResult = m_dLeft / m_dRight;
        }
        else
        {
            AfxMessageBox("Divide by zero");
            m_dResult = 0.0;
        }
        break;
    default:
        TRACE("default; m_nOperation = %d\n", m_nOperation);
    }
    UpdateData(FALSE);
}
```



```
void CMymfc23ADlg::OnCompute()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    switch (m_nOperation)
    {
    case 0:  // add
        m_dResult = m_dLeft + m_dRight;
        break;
    case 1:  // subtract
        m_dResult = m_dLeft - m_dRight;
        break;
    case 2:  // multiply
        m_dResult = m_dLeft * m_dRight;
        break;
    case 3:  // divide
        if (m_dRight != 0.0) {
            m_dResult = m_dLeft / m_dRight;
        }
        else
        {
            AfxMessageBox("Divide by zero");
            m_dResult = 0.0;
        }
        break;
    default:
        TRACE("default; m_nOperation = %d\n", m_nOperation);
    }
    UpdateData(FALSE);
}
```

Listing 1.

Build and test the MYMFC23A application. Notice that the program's icon appears in the Microsoft Windows taskbar. Verify that you can minimize the dialog window.



```
My Calculator

100    Operation    3    =    97
       ( ) Add
       (•) Subtract
       ( ) Multiply
       ( ) Divide
```

Figure 9: MYMFC23A program output, a dialog based program.

Try the divide by 0, so you can trigger the `AfxMessageBox()` function.

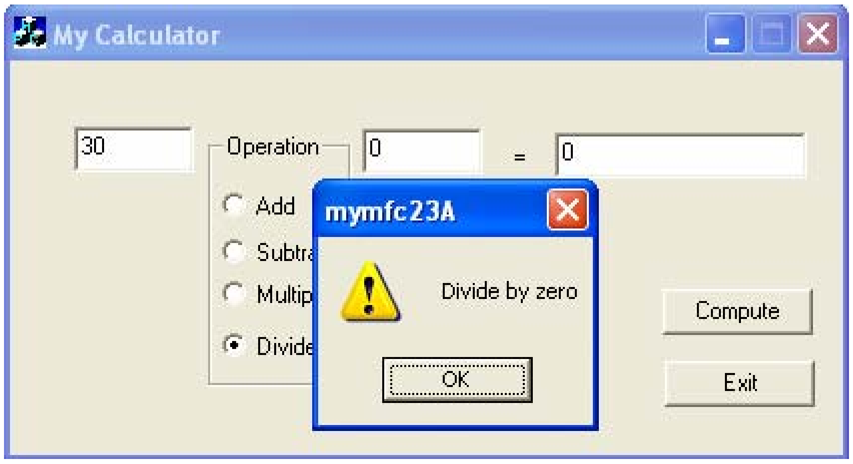


Figure 10: Invoking the `AfxMessageBox()` function.

The Application Class `InitInstance()` Function

The critical element of the MYMFC23A application is the `CMymfc23AApp::InitInstance` function generated by AppWizard. A normal `InitInstance()` function creates a main frame window and returns `TRUE`, allowing the program's message loop to run. The MYMFC23A version constructs a modal dialog object, calls `DoModal()`, and then returns `FALSE`. This means that the application exits after the user exits the dialog. The `DoModal()` function lets the Windows dialog procedure get and dispatch messages, as it always does. Note that AppWizard does not generate a call to `CWinApp::SetRegistryKey`.

Here is the generated `InitInstance()` code from **mymfc23A.cpp**:

```
BOOL CMymfc23AApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();       // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

    CMymfc23ADlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we
    // exit the application, rather than start the application's
    // message pump.
    return FALSE;
}
```

The Dialog Class and the Program Icon

The generated `CMymfc23ADlg` class contains these two message map entries:

```
...
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
...
```

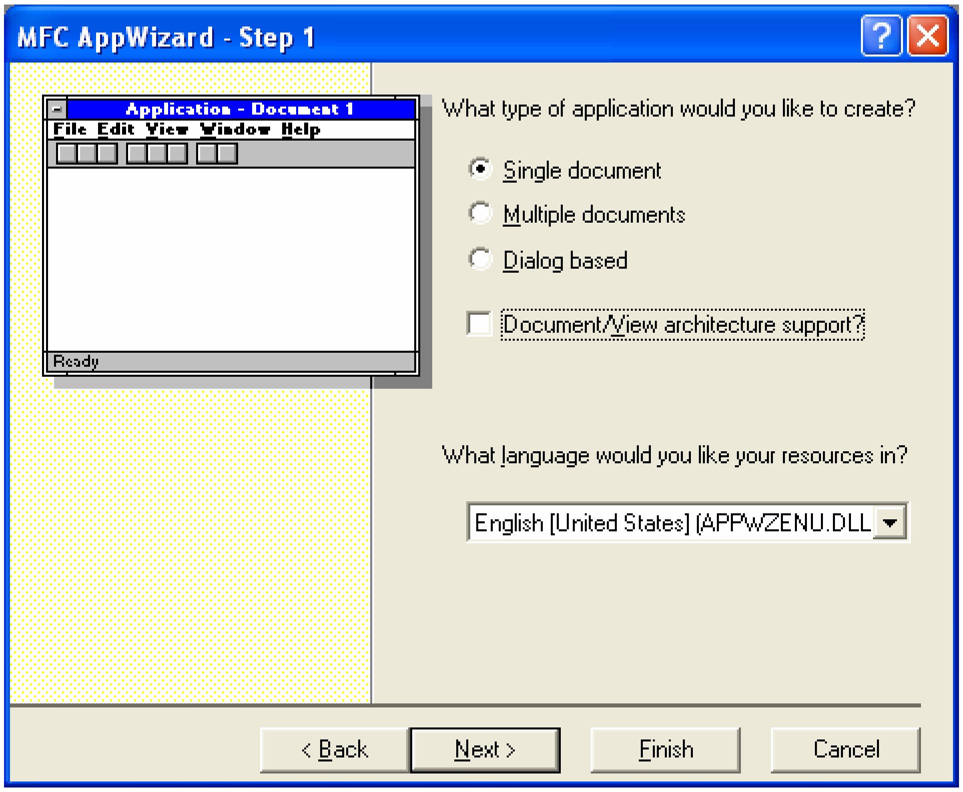
The associated handler functions take care of displaying the application's icon when the user minimizes the program. This code applies only to Microsoft Windows NT version 3.51, in which the icon is displayed on the desktop. You don't need the three handlers for Windows 95, Windows 98, or Windows NT 4.0 because those versions of Windows display the program's icon directly on the taskbar.
There is some icon code that you do need. It's in the dialog's handler for `WM_INITDIALOG`, which is generated by AppWizard. Notice the two `SetIcon()` calls in the `OnInitDialog()` function code below. If you checked the **About** box option, AppWizard generates code to add an **About** box to the **System** menu. The variable `m_hIcon` is a data member of the dialog class that is initialized in the constructor.

```
BOOL CMymfc23ADlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this
    // automatically when the application's main window
    // is not a dialog
    SetIcon(m_hIcon, TRUE);     // Set big icon
    SetIcon(m_hIcon, FALSE);    // Set small icon

    // TODO: Add extra initialization here

    return TRUE;  // return TRUE unless you set the focus to a control
}
```

The MYMFC23B Example: An SDI Application

This is an SDI "`Hello, world!`" classic C/C++ program example. The application has only one window, an object of a class derived from `CFrameWnd`. All drawing occurs inside the frame window and all messages are handled there.

Run AppWizard to produce \mfcproject\mymfc23B. Select the **Single Document** option in the AppWizard Step 1 dialog and uncheck the **Document/View Architecture Support?** option, as shown here. Accept the default for other options.



What type of application would you like to create?

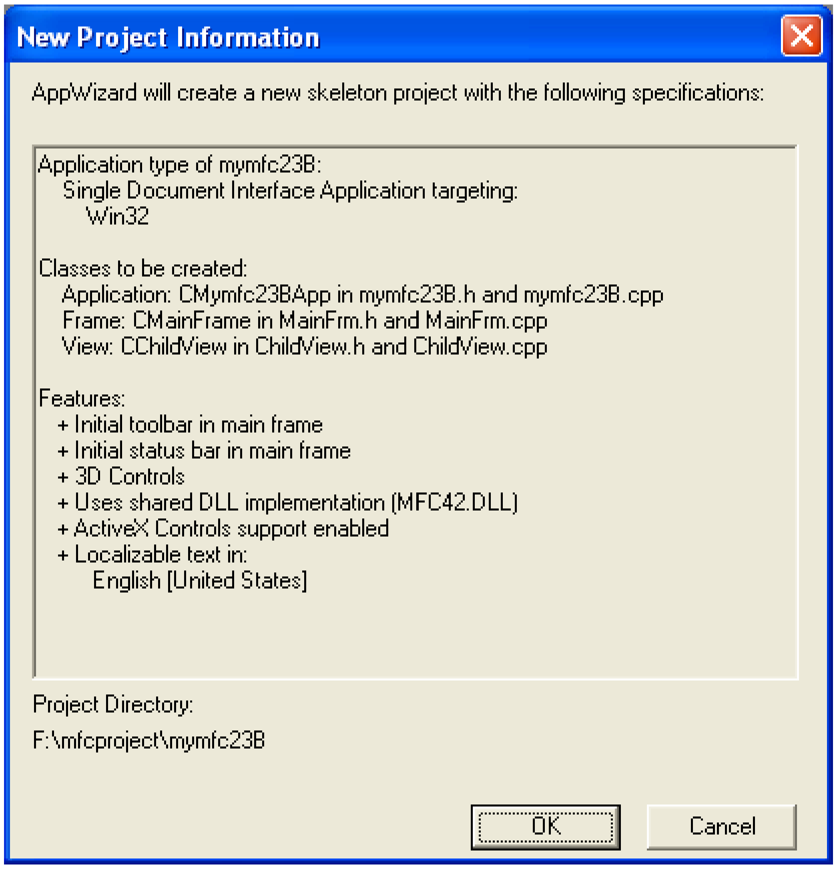
[✓ Single document] Multiple documents Dialog based

☐ Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 11: AppWizard step 1, SDI project without Document/View architecture support.



AppWizard will create a new skeleton project with the following specifications:

```
Application type of mymfc23B:
    Single Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CMymfc23BApp in mymfc23B.h and mymfc23B.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    View: CChildView in ChildView.h and ChildView.cpp

Features:
   + Initial toolbar in main frame
   + Initial status bar in main frame
   + 3D Controls
   + Uses shared DLL implementation (MFC42.DLL)
   + ActiveX Controls support enabled
   + Localizable text in:
         English [United States]
```

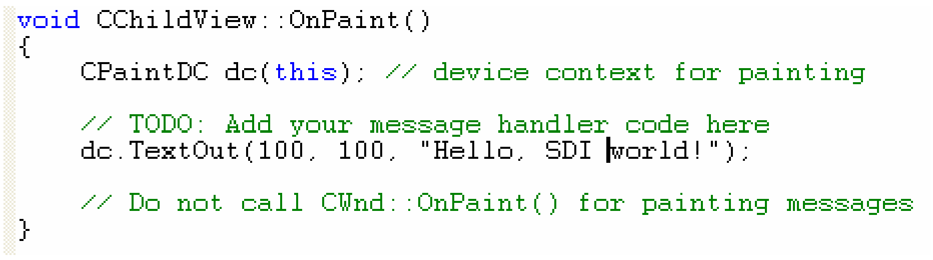
Project Directory:

F:\mfcproject\mymfc23B

Figure 12: MYMFC23B project summary.

Add code to paint in the dialog. Add the following code to the `CChildView::OnPaint` function in the **ChildView.cpp** source code file:

```
void CChildView::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    dc.TextOut(100, 100, "Hello, SDI world!");

    // Do not call CWnd::OnPaint() for painting messages
}
```



```
void CChildView::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    // TODO: Add your message handler code here
    dc.TextOut(100, 100, "Hello, SDI world!");

    // Do not call CWnd::OnPaint() for painting messages
}
```

Listing 2.

Build and run. You now have a complete SDI application that has no dependencies on the document-view architecture.

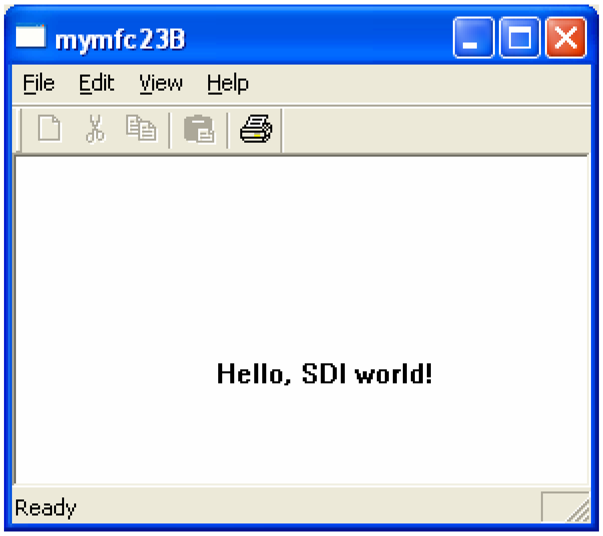


Figure 13: MYMFC23B program output, SDI program without Document/View architecture support.

AppWizard automatically takes out dependencies on the document-view architecture and generates an application for you with the following elements:

- **A main menu**: You can have a Windows-based application without a menu, you don't even need a resource script. But MYMFC23B has both. The application framework routes menu commands to message handlers in the frame class.
- **An icon**: An icon is useful if the program is to be activated from Microsoft Windows Explorer. It's also useful when the application's main frame window is minimized. The icon is stored in the resource, along with the menu.
- **Window close message command handler**: Many an application needs to do special processing when its main window is closed. If you were using documents, you could override the `CDocument::SaveModified` function. Here, to take control of the close process, AppWizard creates

message handlers to process close messages sent as a result of user actions and by Windows itself when it shuts down.

- **Toolbar and status bar**: AppWizard automatically generates a default toolbar and status bar for you and sets up the routing even though there are no document-view classes.

There are several interesting features in the SDI application that have no document-view support, including:

- **`CChildView` class**: Contrary to its name, this class is actually a `CWnd` derivative that is declared in **ChildView.h** and implemented in **ChildView.cpp**. `CChildView` implements only a virtual `OnPaint()` member function, which contains any code that you want to draw in the frame window (as illustrated in the MYMFC23B sample).
- **`CMainFrame` class**: This class contains a data member, `m_wndView` that is created and initialized in the `CMainFrame::OnCreate` member function.
- **`CMainFrame::OnSetFocus` function**: This function makes sure the focus is translated to the `CChildView`:

```
void CMainFrame::OnSetFocus(CWnd* pOldWnd)
{
    // forward focus to the view window
    m_wndView.SetFocus();
}
```

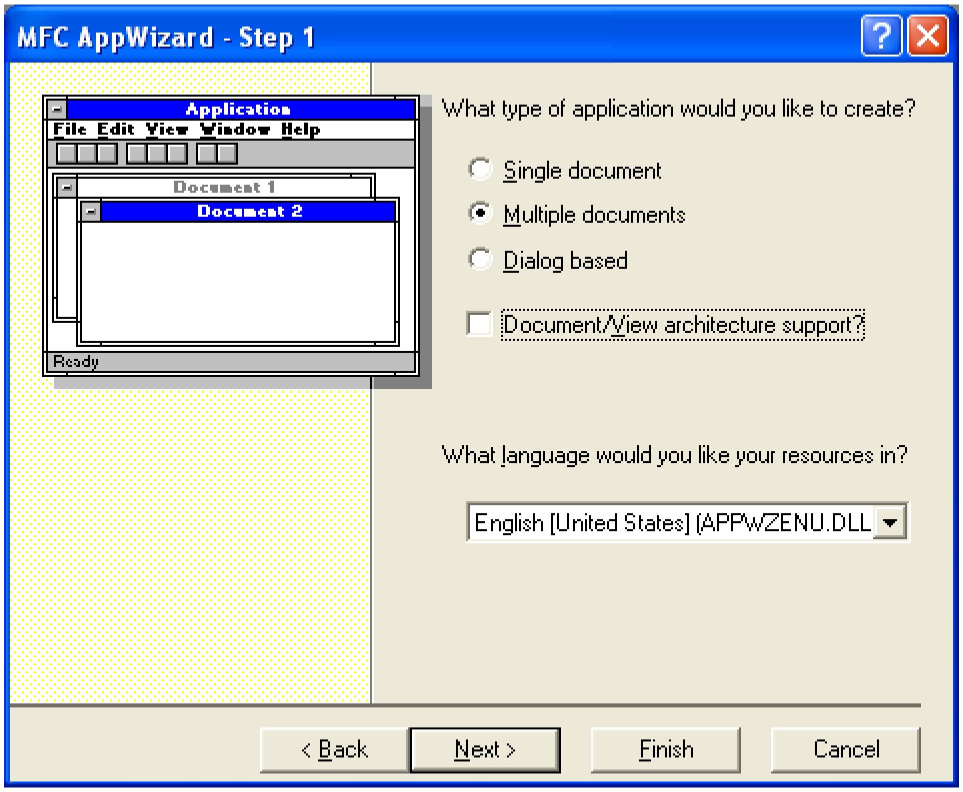
`CMainFrame::OnCmdMsg` function: This function gives the view a chance to handle any command messages first:

```
BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo)
{
    // let the view have first crack at the command
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    // otherwise, do default handling
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
```

The MYMFC23C Example: An MDI Application

Now let's create an MDI application that doesn't use the document-view architecture.

Run AppWizard to produce \mfcproject\mymfc23C. Select the **Multiple Documents** option in the AppWizard Step 1 dialog and uncheck the **Document/View Architecture Support?** option, as shown here. Accept the default for other options.



What type of application would you like to create?

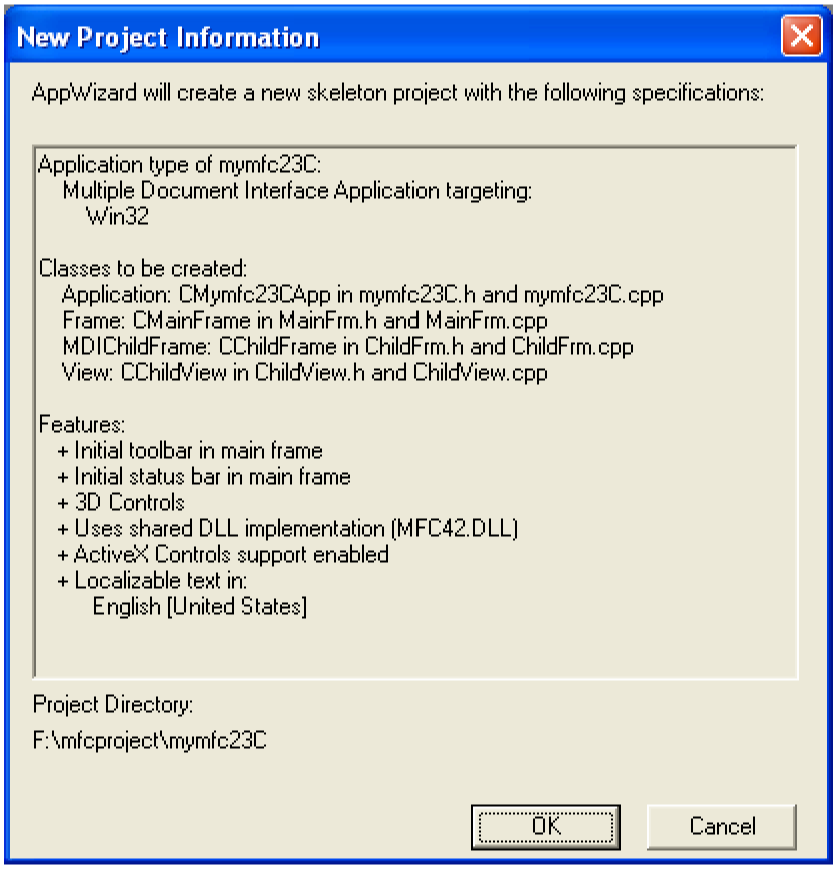
- ( ) Single document
- (●) Multiple documents
- ( ) Dialog based

☐ Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 14: AppWizard step 1, MDI without Document/View architecture support.



AppWizard will create a new skeleton project with the following specifications:

```
Application type of mymfc23C:
   Multiple Document Interface Application targeting:
      Win32

Classes to be created:
   Application: CMymfc23CApp in mymfc23C.h and mymfc23C.cpp
   Frame: CMainFrame in MainFrm.h and MainFrm.cpp
   MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
   View: CChildView in ChildView.h and ChildView.cpp

Features:
  + Initial toolbar in main frame
  + Initial status bar in main frame
  + 3D Controls
  + Uses shared DLL implementation (MFC42.DLL)
  + ActiveX Controls support enabled
  + Localizable text in:
       English [United States]
```

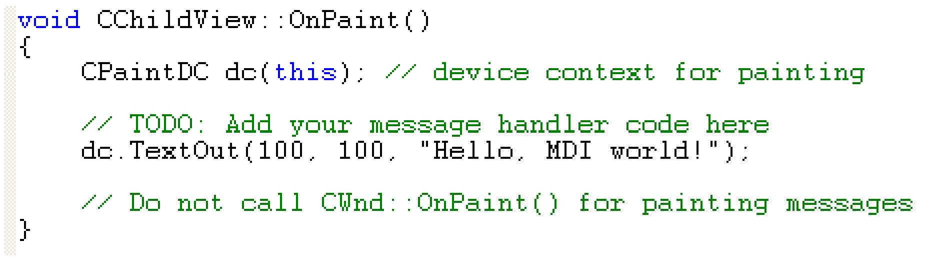
Project Directory:

F:\mfcproject\mymfc23C

Figure 15: MYMFC23C project summary.

Add code to paint in the dialog. Add the following code to the `CChildView::OnPaint` function in the **ChildView.cpp** source code file:

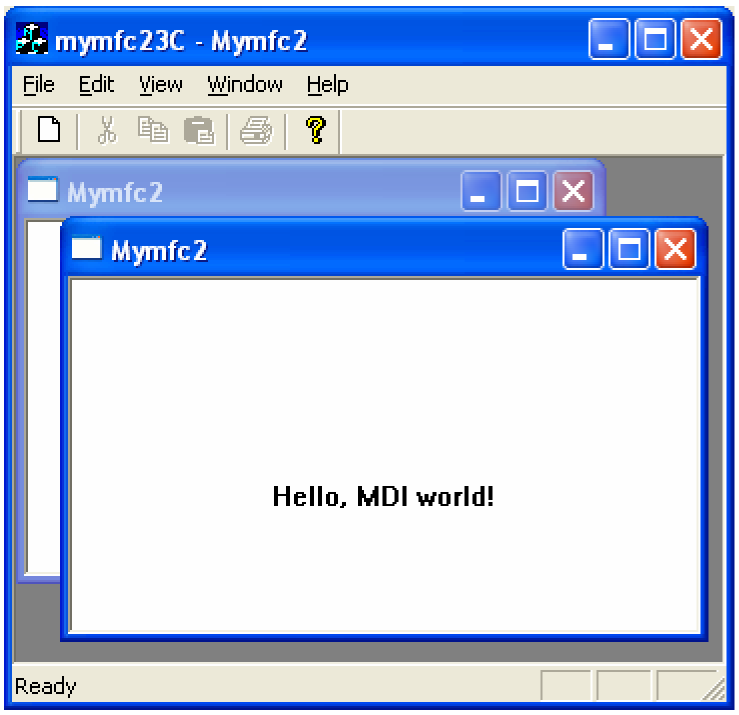
```
void CChildView::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    dc.TextOut(100, 100, "Hello, MDI world!");

    // Do not call CWnd::OnPaint() for painting messages
}
```



```
void CChildView::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    // TODO: Add your message handler code here
    dc.TextOut(100, 100, "Hello, MDI world!");

    // Do not call CWnd::OnPaint() for painting messages
}
```

Listing 3.

Build and run. You now have a complete MDI application without dependencies on the document-view architecture.



```
Hello, MDI world!
```

Figure 16: MYMFC23C program output, MDI program without Document/View architecture support.

As in MYMFC23B, this example automatically creates a `CChildView` class. The main difference between MYMFC23B and MYMFC23C is the fact that in MYMFC23C the `CChildView` class is created in the `CChildFrame::OnCreate` function instead of in the `CMainFrame` class.

In this module you've learned how to create three kinds of applications that do not depend on the document-view architecture. Examining how these applications are generated is also a great way to learn how MFC works. We recommend that you compare the generated results to similar applications with document-view architecture support to get a complete picture of how the document-view classes work with the rest of MFC.

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.