Module 12: Serialization: Reading and Writing Documents—MDI Applications

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. You can compare the standard C file I/O, standard C++ file I/O and Win32 directory, file and access controls with the MFC serialization. So many things lor! Similar but not same. Those links also given at the end of this tutorial.

Reading and Writing Documents: MDI Applications

This module introduces the Microsoft Foundation Class (MFC) Library version 6.0 Multiple Document Interface (MDI) application and demonstrates how it reads and writes its document files. The MDI application appears to be the preferred MFC library program style. It's the AppWizard default, and most of the sample programs that come with Microsoft Visual C++ are MDI applications.
In this module, you'll learn the similarities and differences between Single Document Interface (SDI) and MDI applications and you'll learn how to convert an SDI application to an MDI application. Be sure you thoroughly understand the SDI application described in Module 11 before you attack the MDI application in this module.

The MDI Application

Before you look at the MFC library code for MDI applications, you should be familiar with the operation of Microsoft Windows MDI programs. Take a close look at Visual C++ now. It's an MDI application whose "multiple documents" are program source code files. Visual C++ is not the most typical MDI application, however, because it collects its documents into projects. It's better to examine Microsoft Word or, better yet, a real MFC library MDI application, the kind that AppWizard generates.

A Typical MDI Application, MFC Style

This module's example, MYMFC18, is an MDI version of MYMFC17. Run the MYMFC17 example to see an illustration of the SDI version after the user has selected a file. Now look at the MDI equivalent, as shown in Figure 1.

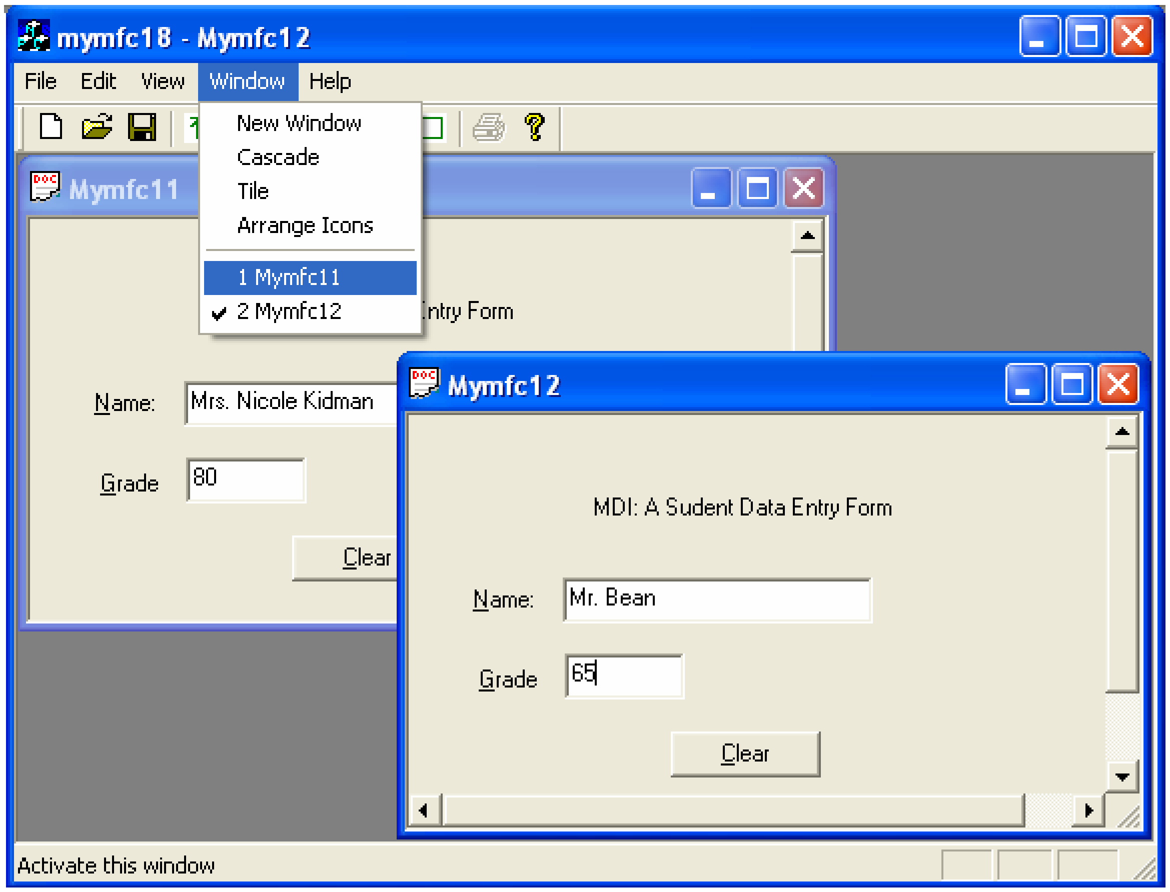


Figure 1: The MYMFC18 application with two files open and the Window menu shown.

The user has two separate document files open, each in a separate MDI child window, but only one child window is active, the lower window, which lies on top of the other child window. The application has only one menu and one toolbar, and all commands are routed to the **active child window**. The main window's title bar reflects the name of the active child window's document file. The child window's minimize box allows the user to reduce the child window to an icon in the main window. The application's Window menu (shown in Figure 18-1) lets the user control the presentation through the following items.

Menu Item	Action
New Window	Opens as an additional child window for the selected document
Cascade	Arranges the existing windows in an overlapped pattern
Tile	Arranges the existing windows in a non-overlapped, tiled pattern
Arrange Icons	Arranges minimized windows in the frame window
(document names)	Selects the corresponding child window and brings it to the top

Table 1

If the user closes both child windows (and opens the **File** menu), the application looks like the Figure 2.

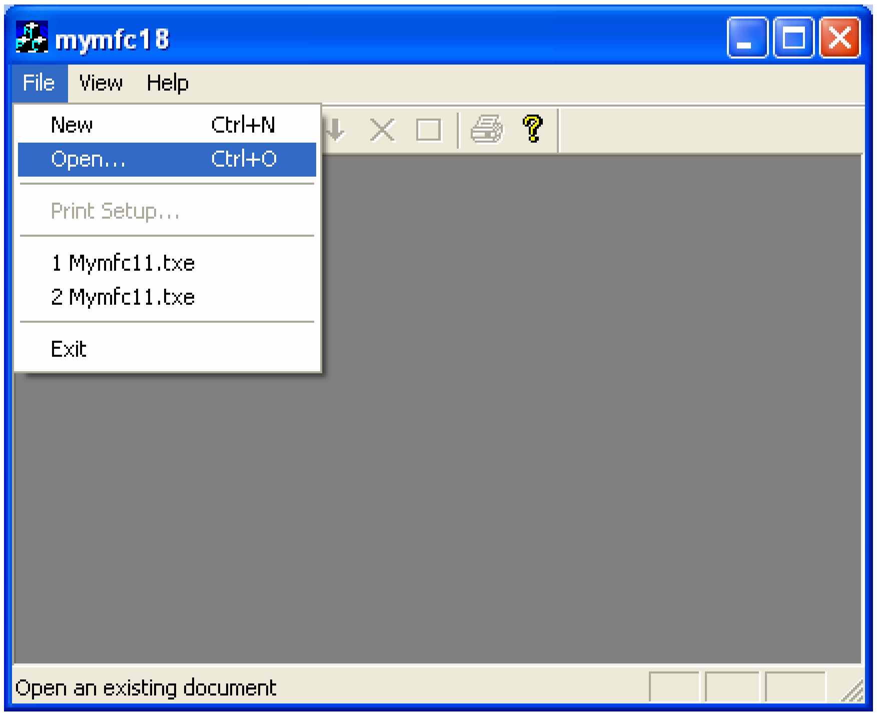


Figure 2: MYMFC18 with no child windows.

The **File** menu is different, most toolbar buttons are disabled, and the window caption does not show a filename. The only choices the user has are to start a new document or to open an existing document from disk. Figure 3 shows the application when it first starts up and a new document is created. The single child window has been maximized.

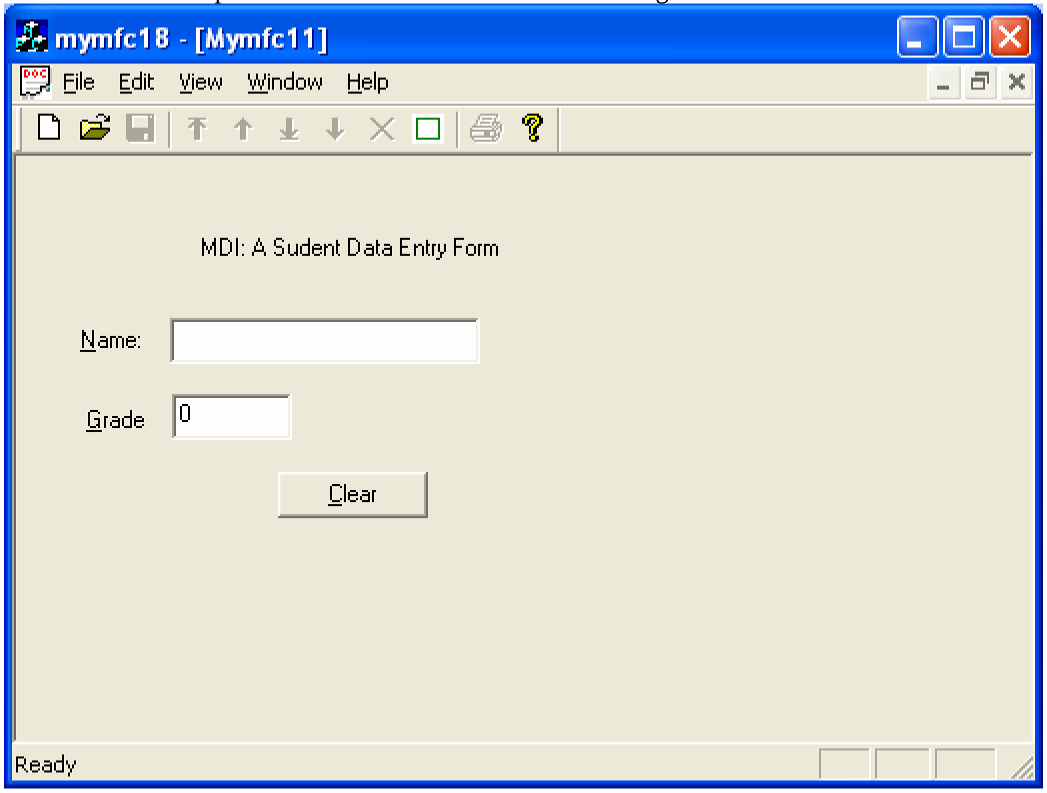


Figure 3: MYMFC18 with initial child window.

The single, empty child window has the default document name Mymfc11. This name is based on the **Doc Type Name** (Mymfc1) that you selected in the **Advanced Options** dialog after clicking the **Advanced** button in Step 4 of AppWizard. The first new file is **Mymfc11**, the second is **Mymfc12**, and so forth. The user normally chooses a different name when saving the document.
An MFC library MDI application, like many commercial MDI applications, starts up with a new, empty document. (Visual C++ is an exception.) If you want your application to start up with a blank frame, you can modify the argument to the `ProcessShellCommand()` call in the application class file, as shown in example MYMFC18.

For Win32 Programmers

Starting with version 3.0, Windows directly supports MDI applications. The MFC library builds on this Windows support to create an MDI environment that parallels the SDI environment. In a Win32 MDI application, a main application frame window contains the menu and a single client window. The client window manages various child windows that correspond to documents. The MDI client window has its own pre-registered window class (not to be confused with a C++ class) with a procedure that handles special messages such as `WM_MDICASCADE` and `WM_MDITILE`. An MDI child window procedure is similar to the window procedure for an SDI main window.
In the MFC library, the CMDIFrameWnd class encapsulates the functions of both the main frame window and the MDI client window. This class has message handlers for all the Windows MDI messages and thus can manage its child windows, which are represented by objects of class `CMDIChildWnd`.

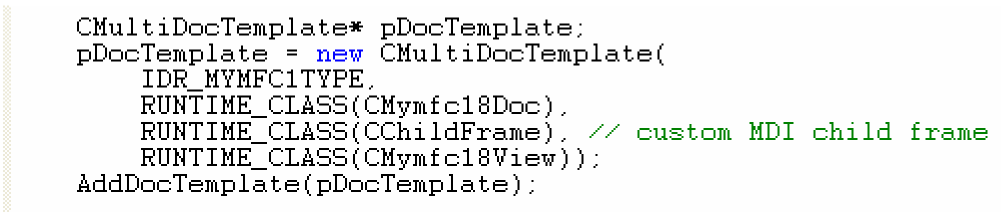
The MDI Application Object

You're probably wondering how an MDI application works and what code makes it different from an SDI application. Actually, the startup sequences are pretty much the same. An application object of a class derived from class `CWinApp` has an overridden `InitInstance()` member function. This `InitInstance()` function is somewhat different from the SDI `InitInstance()` function, starting with the call to `AddDocTemplate()`.

The MDI Document Template Class

The MDI template construction call in `InitInstance()` looks something like this:

```
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_MYMFC1TYPE,
    RUNTIME_CLASS(CMymfc18Doc),
    RUNTIME_CLASS(CChildFrame), // custom MDI child frame
    RUNTIME_CLASS(CMymfc18View));
AddDocTemplate(pDocTemplate);
```



```
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_MYMFC1TYPE,
    RUNTIME_CLASS(CMymfc18Doc),
    RUNTIME_CLASS(CChildFrame), // custom MDI child frame
    RUNTIME_CLASS(CMymfc18View));
AddDocTemplate(pDocTemplate);
```

Listing 1.

Unlike the SDI application you saw in Module 11, an MDI application can use multiple document types and allows the simultaneous existence of more than one document object. This is the essence of the MDI application. The single `AddDocTemplate()` call shown above permits the MDI application to support multiple child windows, each connected to a document object and a view object. It's also possible to have several child windows (and corresponding view objects) connected to the same document object. In this module, we'll start with only one view class and one document class. You'll see multiple view classes and multiple document classes in Module 14. When your application is running, the document template object maintains a list of active document objects that were created from the template. The `CMultiDocTemplate()` member functions `GetFirstDocPosition()` and `GetNextDoc()` allow you to iterate through the list. Use `CDocument::GetDocTemplate` to navigate from a document to its template.

The MDI Frame Window and the MDI Child Window

The SDI examples had only one frame window class and only one frame window object. For SDI applications, AppWizard generated a class named `CMainFrame`, which was derived from the class `CFrameWnd`. An MDI application has two frame window classes and many frame objects, as shown in the table below. The MDI frame-view window relationship is shown in Figure 4.

Base Class	**AppWizard-Generated Class**	**Number of Objects**	**Menu and Control Bars**	**Contains a View**	**Object Constructed**
`CMDIFrameWnd`	`CMainFrame`	1 only	Yes	No	In application class's `InitInstance()` function
`CMDIChildWnd`	`CChildFrame`	1 per child window	No	Yes	By application framework when a new child window is opened.

Table 2.

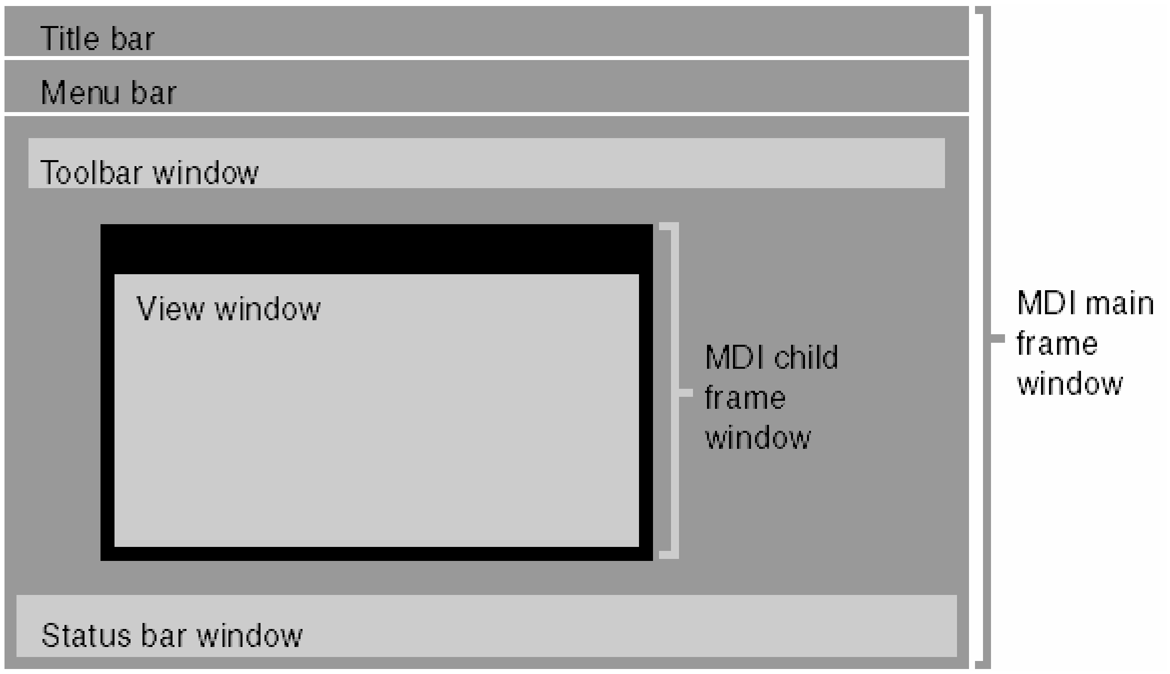
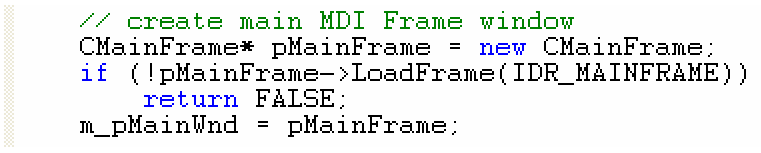


Figure 4: The MDI frame-view window relationship.

In an SDI application, the `CMainFrame` object frames the application and contains the view object. In an MDI application, the two roles are separated. Now the `CMainFrame` object is explicitly constructed in `InitInstance()`, and the `CChildFrame` object contains the view. AppWizard generates the following code:

```
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
m_pMainWnd = pMainFrame;
```

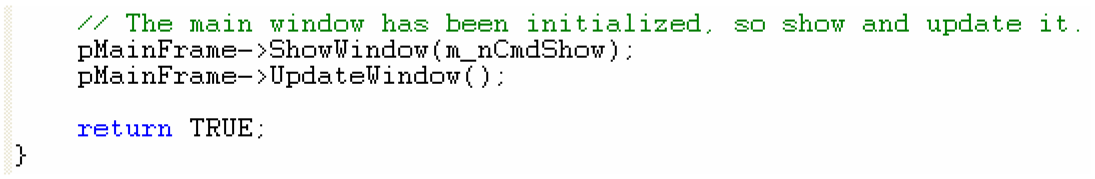


```
// create main MDI Frame window
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
m_pMainWnd = pMainFrame;
```

Listing 2.

Code calls `ProcessShellCommand()` to create child frame.

```
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();
```



```
    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}
```

Listing 3.

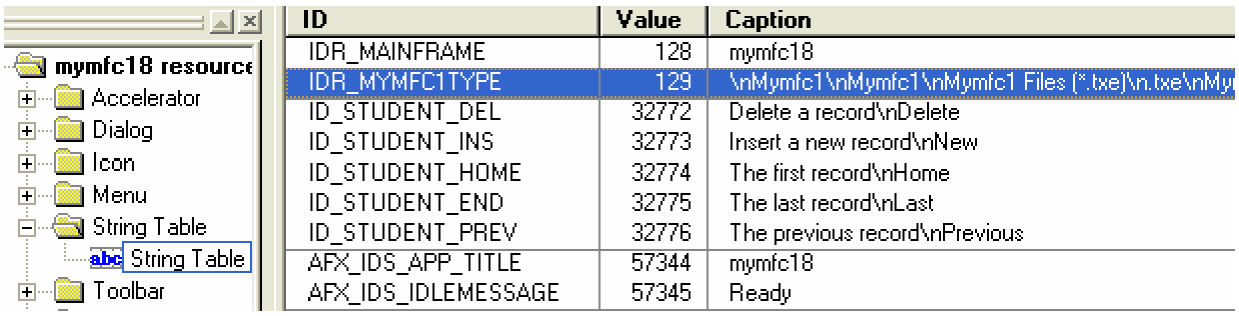
The application framework can create the `CChildFrame` objects dynamically because the `CChildFrame` runtime class pointer is passed to the `CMultiDocTemplate` constructor. The MDI `InitInstance()` function sets the `CWinApp` data member `m_pMainWnd` to point to the application's main frame window. This means you can access `m_pMainWnd` through the global `AfxGetApp()` function anytime you need to get your application's main frame window.

The Main Frame and Document Template Resources

An MDI application (MYMFC18, as described later in this module) has two separate string and menu resources, identified by the `IDR_MAINFRAME` and `IDR_MYMFC18TYPE` constants. The first resource set goes with the empty main frame window; the second set goes with the occupied main frame window. Here are the two string resources with substrings broken out:

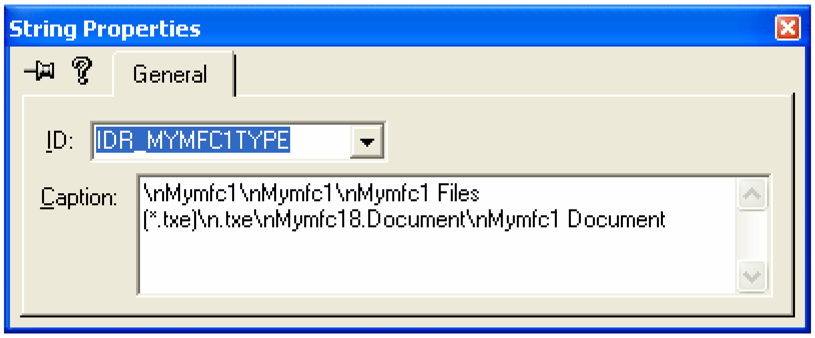
```
IDR_MAINFRAME
    "mymfc18"                   // application window caption

IDR_MYMFC18TYPE
    "\n"                        // (not used)
    "Mymfc1\n"                  // root for default document name
    "Mymfc1\n"                  // document type name
    "Mymfc1 Files (*.txe)\n"    // document type description and filter
    ".txe\n"                    // extension for documents of this type
    "Mymfc18.Document\n"        // Registry file type ID
    "Mymfc1 Document"           // Registry file type description
```

You can see this by double clicking the **String Table** in the **ResourceView** and the `IDR_MYMFC1TYPE` on the left window as shown below.



ID	Value	Caption
IDR_MAINFRAME	128	mymfc18
IDR_MYMFC1TYPE	129	\nMymfc1\nMymfc1\nMymfc1 Files (*.txe)\n.txe\nMy
ID_STUDENT_DEL	32772	Delete a record\nDelete
ID_STUDENT_INS	32773	Insert a new record\nNew
ID_STUDENT_HOME	32774	The first record\nHome
ID_STUDENT_END	32775	The last record\nLast
ID_STUDENT_PREV	32776	The previous record\nPrevious
AFX_IDS_APP_TITLE	57344	mymfc18
AFX_IDS_IDLEMESSAGE	57345	Ready

Figure 5: String table.



ID: IDR_MYMFC1TYPE

Caption: \nMymfc1\nMymfc1\nMymfc1 Files (*.txe)\n.txe\nMymfc18.Document\nMymfc1 Document

Figure 6: String properties for IDR_MYMFC1TYPE.

The resource compiler won't accept the string concatenations as shown above. If you examine the **mymfc18.rc** file, you'll see the substrings combined in one long string. The application window caption comes from the IDR_MAINFRAME string. When a document is open, the document filename is appended. The last two substrings in the IDR_MYMFC18TYPE string support embedded launch and drag and drop.

Creating an Empty Document: The CWinApp::OnFileNew Function

The MDI InitInstance() function calls OnFileNew() (through ProcessShellCommand()), as did the SDI InitInstance() function. This time, however, the main frame window has already been created. OnFileNew(), through a call to the CMultiDocTemplate() function OpenDocumentFile(), now does the following:

1. Constructs a document object but does not attempt to read data from disk.
2. Constructs a child frame window object (of class CChildFrame). Also creates the child frame window but does not show it. In the main frame window, the IDR_MYMFC18TYPE menu replaces the IDR_MAINFRAME menu. IDR_MYMFC18TYPE also identifies an icon resource that is used when the child window is minimized within the frame.
3. Constructs a view object. Also creates the view window but does not show it.
4. Establishes connections among the document, the main frame, and view objects. Do not confuse these object connections with the class associations established by the call to AddDocTemplate().
5. Calls the virtual OnNewDocument() member function for the document object.
6. Calls the virtual OnInitialUpdate() member function for the view object.
7. Calls the virtual ActivateFrame() member function for the child frame object to show the frame window and the view window.

The OnFileNew() function is also called in response to the **File New** menu command. In an MDI application, OnFileNew() performs exactly the same steps as it does when called from InitInstance(). Some functions listed above are not called directly by OpenDocumentFile() but are called indirectly through the application framework.

Creating an Additional View for an Existing Document

If you choose the **New Window** command from the **Window** menu, the application framework opens a new child window that is linked to the currently selected document. The associated CMDIFrameWnd() function, OnWindowNew(), does the following:

1. Constructs a child frame object (of class CChildFrame). Also creates the child frame window but does not show it.
2. Constructs a view object. Also creates the view window but does not show it.
3. Establishes connections between the new view object and the existing document and main frame objects.
4. Calls the virtual OnInitialUpdate() member function for the view object.
5. Calls the virtual ActivateFrame() member function for the child frame object to show the frame window and the view window.

Loading and Storing Documents

In MDI applications, documents are loaded and stored the same way as in SDI applications but with two important differences: a new document object is constructed each time a document file is loaded from disk, and the document object is destroyed when the child window is closed. Don't worry about clearing a document's contents before loading, but override the `CDocument::DeleteContents` function anyway to make the class portable to the SDI environment.

Multiple Document Templates

An MDI application can support multiple document templates through multiple calls to the `AddDocTemplate()` function. Each template can specify a different combination of document, view, and MDI child frame classes. When the user chooses **New** from the **File** menu of an application with multiple templates, the application framework displays a list box that allows the user to select a template by name as specified in the string resource (document type substring). Multiple `AddDocTemplate()` calls are not supported in SDI applications because the document, view, and frame objects are constructed once for the life of the application.
When your application is running, the application object keeps a list of active document template objects. The `CWinApp` member functions `GetFirstDocTemplatePosition()` and `GetNextDocTemplate()` allow you to iterate through the list of templates. These functions, together with the `CDocTemplate()` member functions `GetFirstDocPosition()` and `GetNextDoc()`, allow you to access all of the application's document objects. If you don't want the template list box, you can edit the **File** menu to add a **New** menu item for each document type. Code the command message handlers as shown below, using the document type substring from each template.

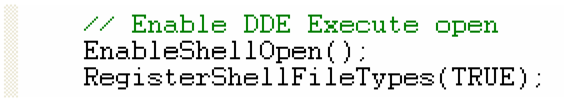
```
void CMyApp::OnFileNewStudent()
{
    OpenNewDocument("Studnt");
}

void CMyApp::OnFileNewTeacher()
{
    OpenNewDocument("Teachr");
}
```

Then add the `OpenNewDocument()` helper function as follows:

```
BOOL CMyApp::OpenNewDocument(const CString& strTarget)
{
    CString strDocName;
    CDocTemplate* pSelectedTemplate;
    POSITION pos = GetFirstDocTemplatePosition();
    while (pos != NULL)
    {
        pSelectedTemplate = (CDocTemplate*) GetNextDocTemplate(pos);
        ASSERT(pSelectedTemplate != NULL);
        ASSERT(pSelectedTemplate->IsKindOf(RUNTIME_CLASS(CDocTemplate)));
        pSelectedTemplate->GetDocString(strDocName, CDocTemplate::docName);
        if (strDocName == strTarget) { // from template's
                                       // string resource
            pSelectedTemplate->OpenDocumentFile(NULL);
            return TRUE;
        }
    }
    return FALSE;
}
```

Explorer Launch and Drag and Drop

When you double-click on a document icon for an MDI application in Windows Explorer, the application launches only if it was not running already; otherwise, a new child window opens in the running application for the document you selected. The `EnableShellOpen()` call in the application class `InitInstance()` function is necessary for this to work.



```
    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);
```

Listing 4.

Drag and drop works much the same way in an MDI application as it does in an SDI application. If you drag a file from Windows Explorer to your MDI main frame window, the program opens a new child frame (with associated document and view) just as if you'd chosen the **File Open** command. As with SDI applications, you must use the AppWizard Step 4 Advanced button to specify the filename extension.

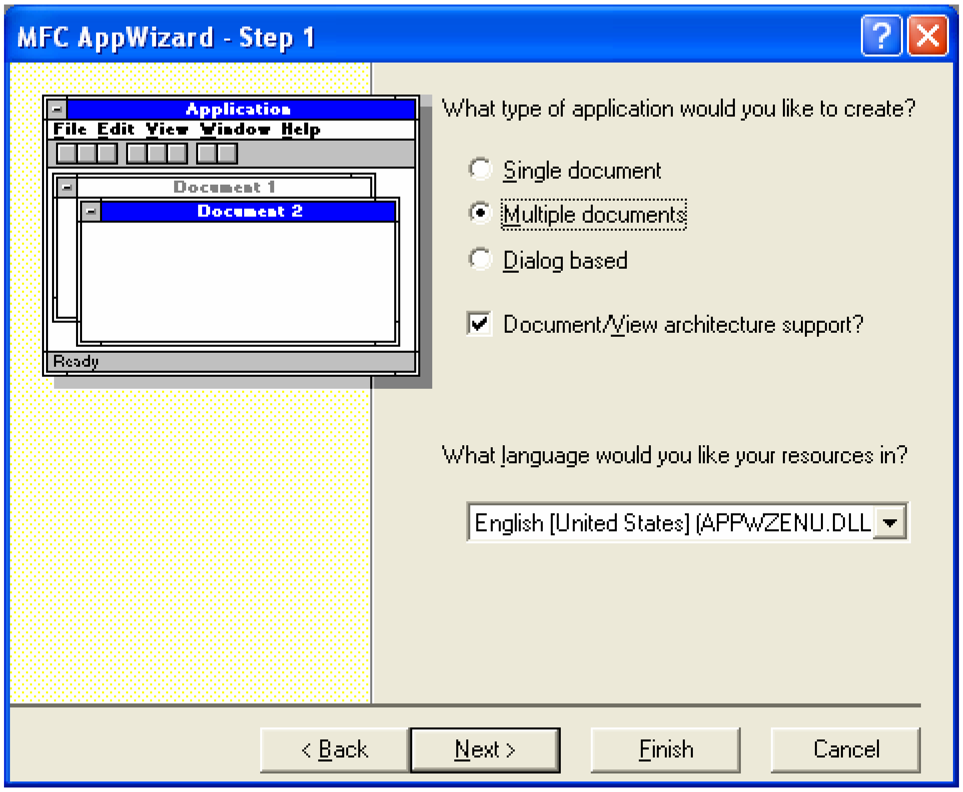
The MYMFC18 Example

This example is the MDI version of the MYMFC17 example from the previous module. It uses the same document and view class code and the same resources (except the program name). The **application code** and **main frame class code** are different, however. All the new code is listed here, including the code that AppWizard generates. A list of the files and classes in the MYMFC18 example are shown in the table below.

Header File	Source Code File	Class	Description
mymfc18.h	mymfc18.cpp	`CMymfc18App`	Application class (from AppWizard)
		`CAboutDlg`	About dialog
MainFrm.h	MainFrm.cpp	`CMainFrame`	MDI main frame
ChildFrm.h	ChildFrm.cpp	`CChildFrame`	MDI child frame
Mymfc18Doc.h	Mymfc18Doc.cpp	`CMymfc18Doc`	Student document (from MYMFC17)
Mymfc18View.h	Mymfc18View.cpp	`CMymfc18View`	Student form view (from MYMFC18)
Student.h	Student.cpp	`CStudent`	Student record (from MYMFC17)
StdAfx.h	StdAfx.h		Precompiled headers (with **afxtempl.h** included)

Table 3.

The steps for this program example are as follows:


MFC AppWizard - Step 1

What type of application would you like to create?

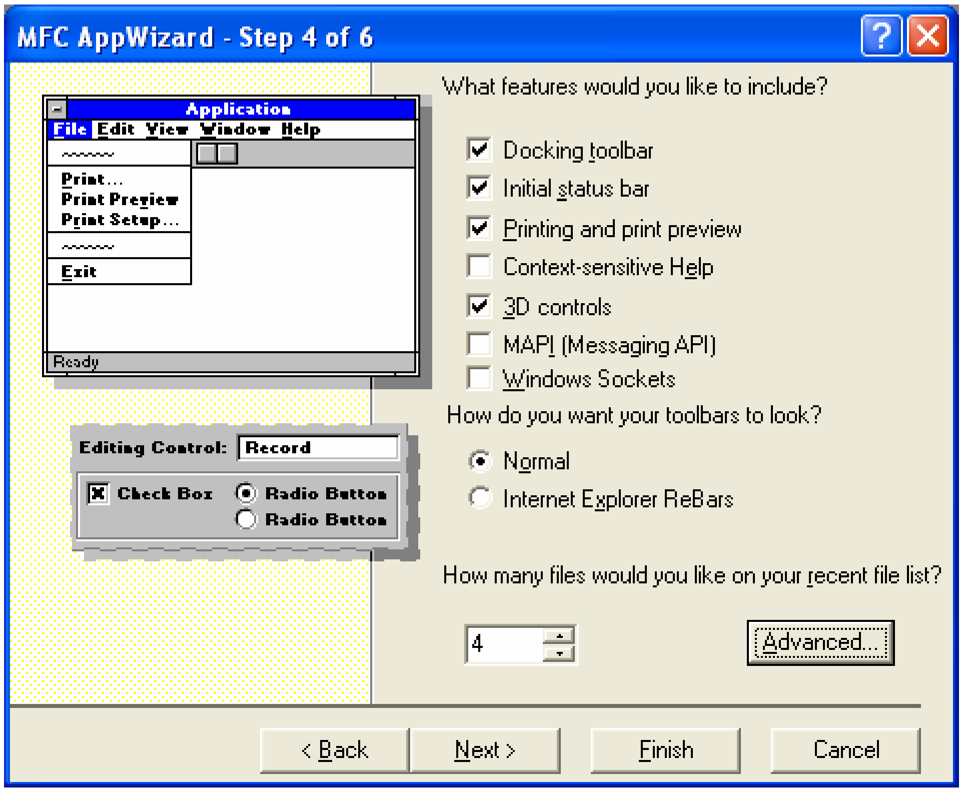
- ( ) Single document
- (●) Multiple documents
- ( ) Dialog based

[✓] Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 7: AppWizard step 1 of 6 for MYMFC18 project, selecting **Multiple documents**.


MFC AppWizard - Step 4 of 6

What features would you like to include?

- [✓] Docking toolbar
- [✓] Initial status bar
- [✓] Printing and print preview
- [ ] Context-sensitive Help
- [✓] 3D controls
- [ ] MAPI (Messaging API)
- [ ] Windows Sockets

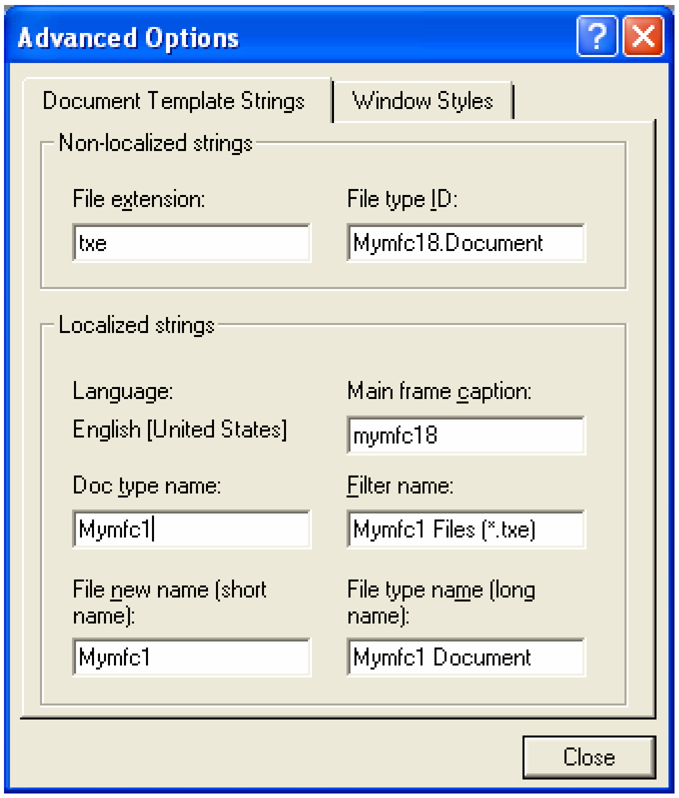
How do you want your toolbars to look?

- (●) Normal
- ( ) Internet Explorer ReBars

How many files would you like on your recent file list?

4

Figure 8: AppWizard step 4 of 6 for MYMFC18 project, setting the **Advanced** options.



Non-localized strings

File extension: txe

File type ID: Mymfc18.Document

Localized strings

Language: English (United States)

Main frame caption: mymfc18

Doc type name: Mymfc1

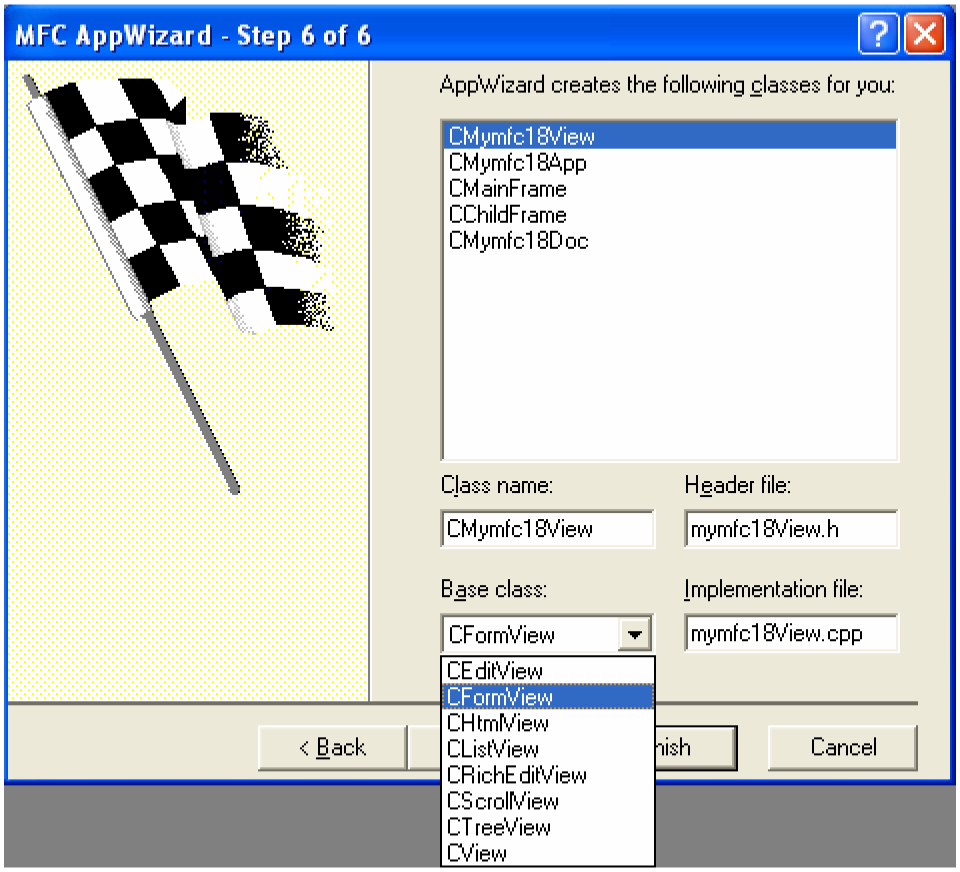
Filter name: Mymfc1 Files (*.txe)

File new name (short name): Mymfc1

File type name (long name): Mymfc1 Document

Figure 9: Entering the **File extension**, you can change other options as required but accept the shown name for this exercise.

Notice the maximum number of the **Doc** type name and for the file extension, it is recommended for this example to use the 3 characters.


MFC AppWizard - Step 6 of 6

AppWizard creates the following classes for you:

CMymfc18View
CMymfc18App
CMainFrame
CChildFrame
CMymfc18Doc

Class name: CMymfc18View

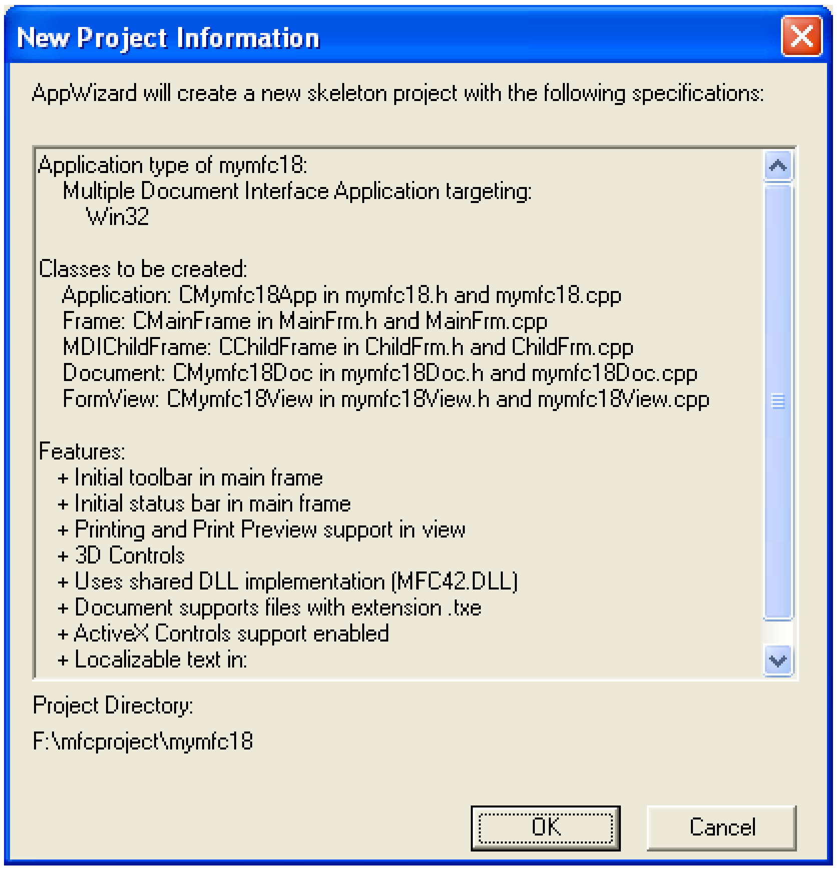
Header file: mymfc18View.h

Base class: CFormView

Implementation file: mymfc18View.cpp

CEditView
CFormView
CHtmlView
CListView
CRichEditView
CScrollView
CTreeView
CView

Figure 10: AppWizard step 6 of 6 for MYMFC18 project, selecting `CFormView` for the view base class.



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

```
Application type of mymfc18:
    Multiple Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CMymfc18App in mymfc18.h and mymfc18.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
    Document: CMymfc18Doc in mymfc18Doc.h and mymfc18Doc.cpp
    FormView: CMymfc18View in mymfc18View.h and mymfc18View.cpp

Features:
   + Initial toolbar in main frame
   + Initial status bar in main frame
   + Printing and Print Preview support in view
   + 3D Controls
   + Uses shared DLL implementation (MFC42.DLL)
   + Document supports files with extension .txe
   + ActiveX Controls support enabled
   + Localizable text in:
```

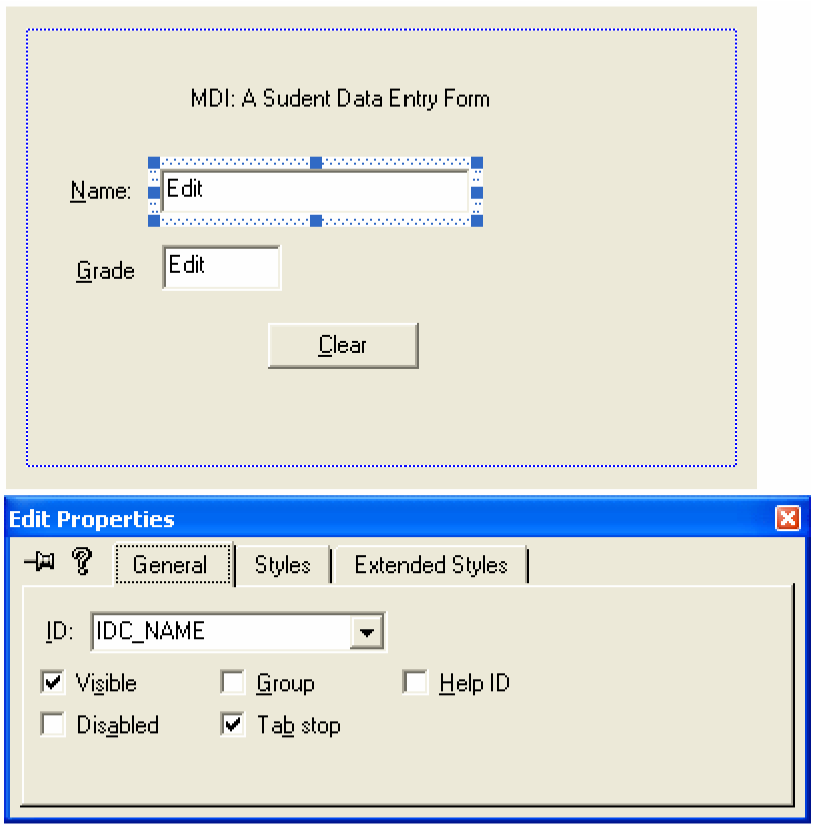
Project Directory:

F:\mfcproject\mymfc18

Figure 11: MYMFC18 project summary.

Control	**ID**
The dialog template	`IDD_MYMFC18_FORM`
Name edit control	`IDC_NAME`
Grade edit control	`IDC_GRADE`
Clear pushbutton	`IDC_CLEAR`

Table 4: Object IDs for MYMFC18.



ID: IDC_NAME

[✓] Visible [ ] Group [ ] Help ID

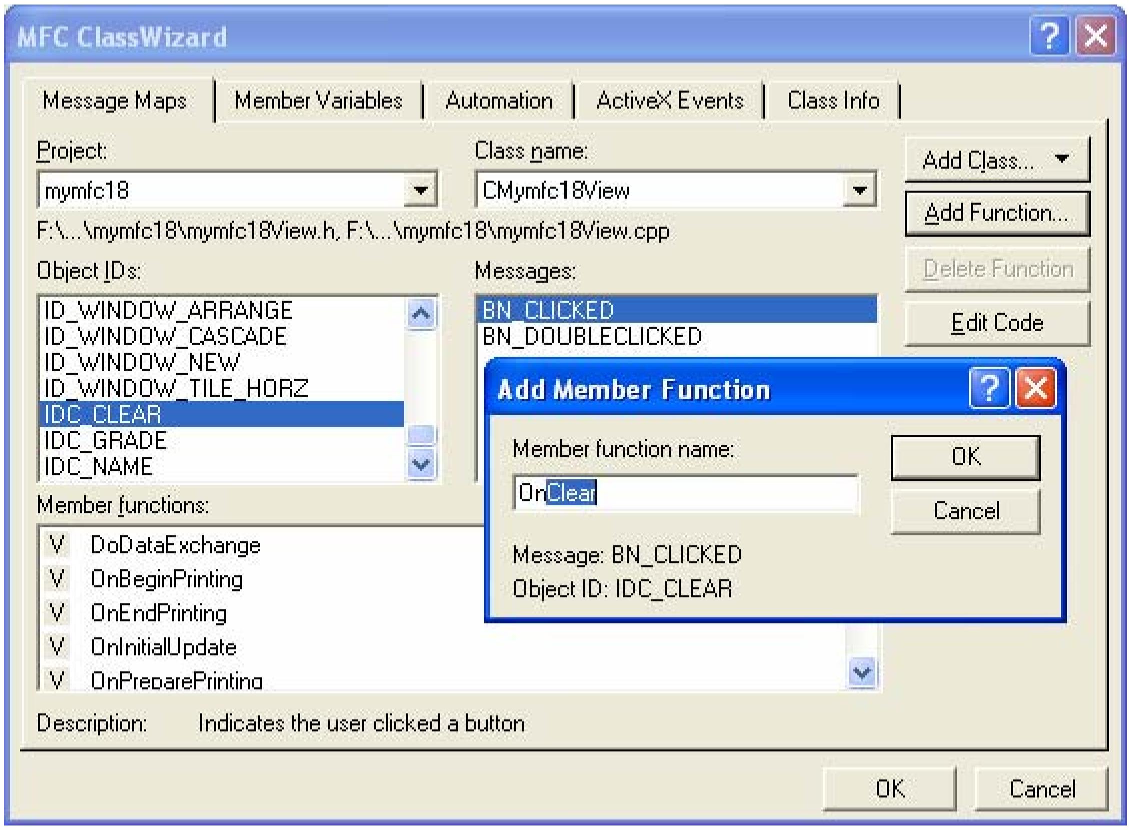
[ ] Disabled [✓] Tab stop

Figure 12: Adding and modifying dialog and its controls properties.

ClassWizard was used to map the `CMymfc18View` **Clear** pushbutton notification message as follows.

Object ID	Message	Member Function
`IDC_CLEAR`	`BN_CLICKED`	`OnClear()`

Table 5.



Project: mymfc18

Class name: CMymfc18View

F:\...\mymfc18\mymfc18View.h, F:\...\mymfc18\mymfc18View.cpp

Object IDs:

```
ID_WINDOW_ARRANGE
ID_WINDOW_CASCADE
ID_WINDOW_NEW
ID_WINDOW_TILE_HORZ
IDC_CLEAR
IDC_GRADE
IDC_NAME
```

Messages:

```
BN_CLICKED
BN_DOUBLECLICKED
```

Member functions:

```
V   DoDataExchange
V   OnBeginPrinting
V   OnEndPrinting
V   OnInitialUpdate
V   OnPreparePrinting
```

Description: Indicates the user clicked a button

Add Member Function

Member function name: OnClear

Message: BN_CLICKED

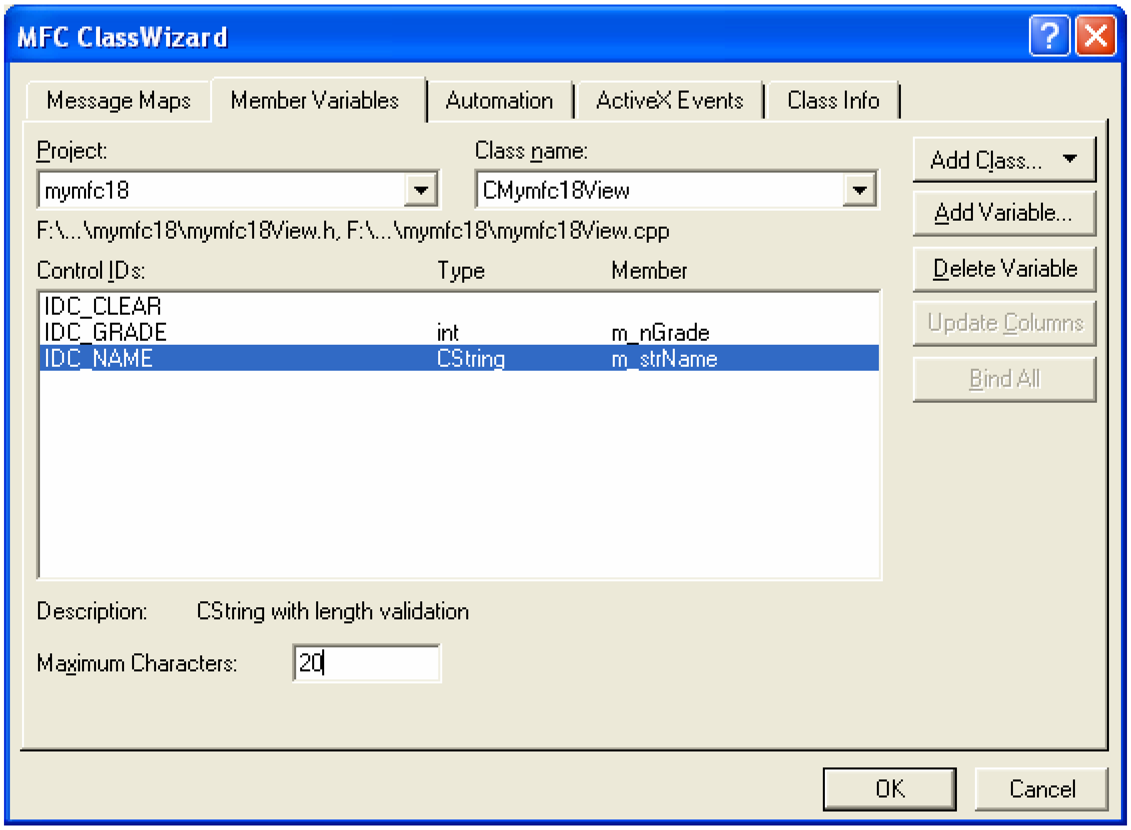
Object ID: IDC_CLEAR

Figure 13: Mapping the `CMymfc18View` **Clear** pushbutton notification message.

And the member variables.

Control ID	Member Variable	Category	Variable Type	Other
`IDC_GRADE`	`m_nGrade`	`Value`	`int`	`Min = 0, max = 100`
`IDC_NAME`	`m_strName`	`Value`	`CString`	`Max= 20`

Table 6.



Project: mymfc18

Class name: CMymfc18View

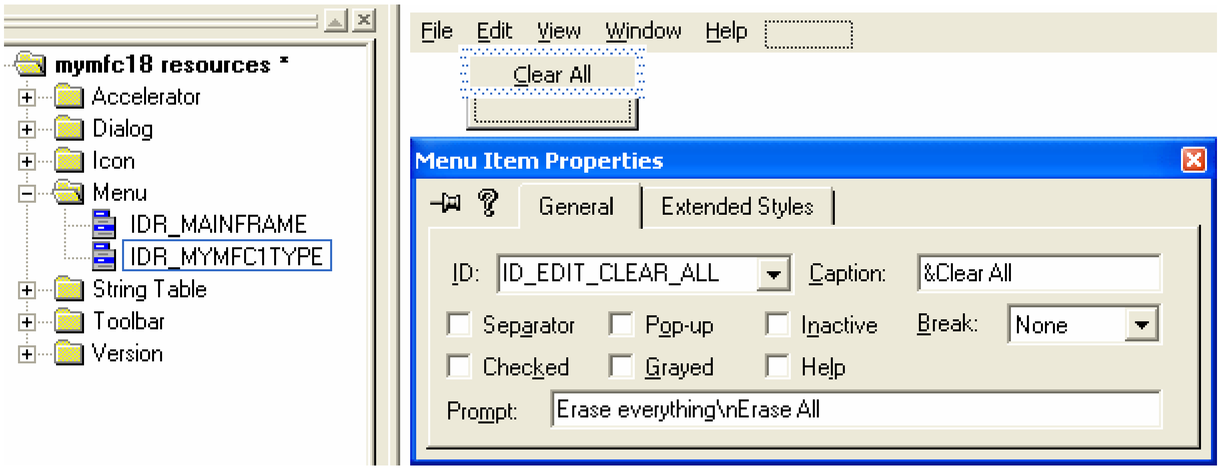
F:\...\mymfc18\mymfc18View.h, F:\...\mymfc18\mymfc18View.cpp

Control IDs:	Type	Member
IDC_CLEAR		
IDC_GRADE	int	m_nGrade
IDC_NAME	CString	m_strName

Description: CString with length validation

Maximum Characters: 20

Figure 14: Using ClassWizard to add member variables.



ID: ID_EDIT_CLEAR_ALL

Caption: &Clear All

Separator, Pop-up, Inactive, Break: None

Checked, Grayed, Help

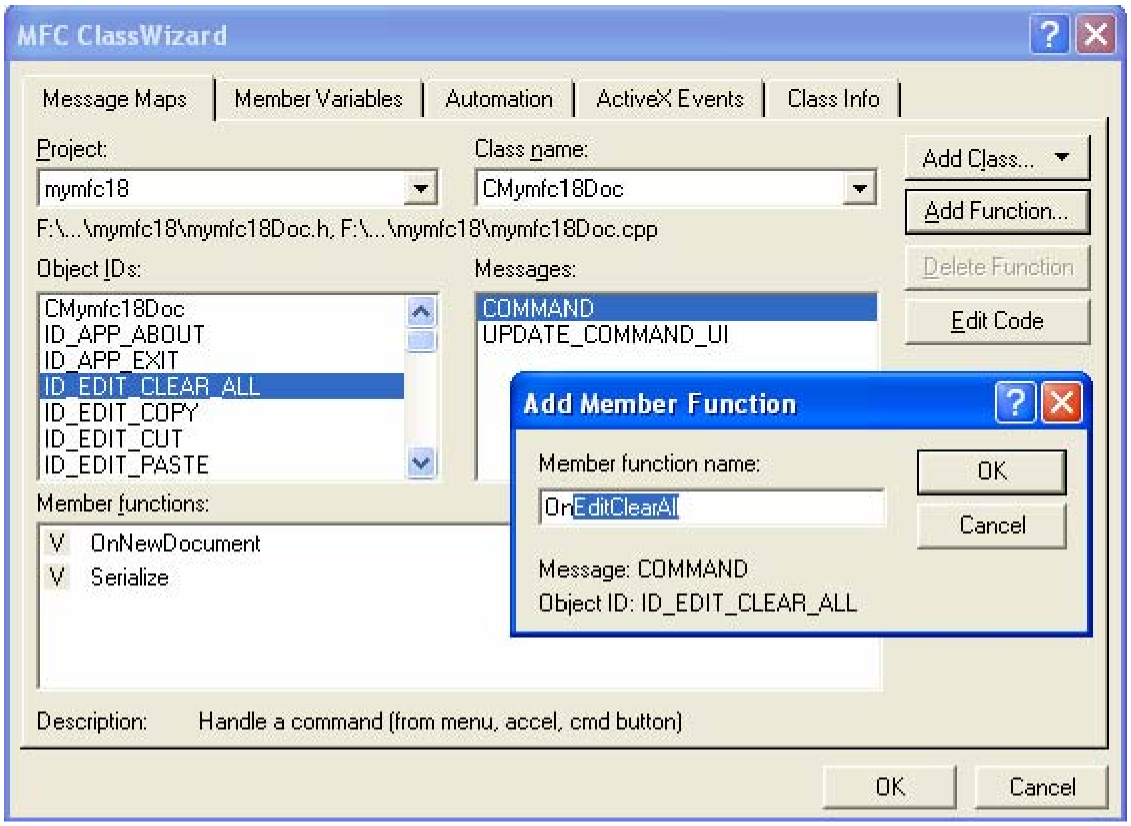
Prompt: Erase everything\nErase All

Figure 15: Adding and modifying **Clear All** menu properties.

The **Edit Clear All** command is handled in the **document** class. The following message handlers were added through ClassWizard.

Object ID	Message	Member Function
ID_EDIT_CLEAR_ALL	COMMAND	OnEditClearAll()
ID_EDIT_CLEAR_ALL	ON_UPDATE_COMMAND_UI	OnUpdateEditClearAll()

Table 7.


MFC ClassWizard

Project: mymfc18

Class name: CMymfc18Doc

F:\...\mymfc18\mymfc18Doc.h, F:\...\mymfc18\mymfc18Doc.cpp

Object IDs: CMymfc18Doc, ID_APP_ABOUT, ID_APP_EXIT, ID_EDIT_CLEAR_ALL, ID_EDIT_COPY, ID_EDIT_CUT, ID_EDIT_PASTE

Messages: COMMAND, UPDATE_COMMAND_UI

Member functions:

V OnNewDocument

V Serialize

Description: Handle a command (from menu, accel, cmd button)

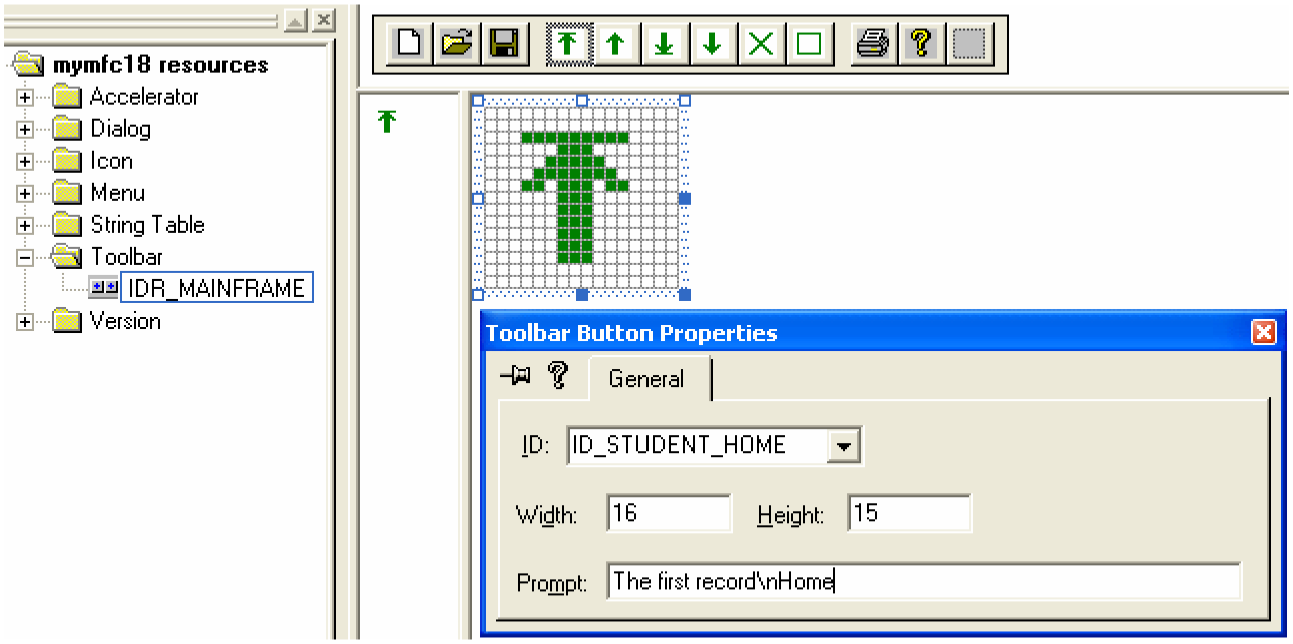
Add Member Function

Member function name: OnEditClearAll

Message: COMMAND

Object ID: ID_EDIT_CLEAR_ALL

Figure 16: Adding message handlers for `ID_EDIT_CLEAR_ALL`.



ID: ID_STUDENT_HOME

Width: 16

Height: 15

Prompt: The first record\nHome

Figure 17: Creating and modifying toolbar buttons properties.

Button	Function
🡅	Retrieves the first student record

⤓	Retrieves the last student record
↑	Retrieves the previous student record
↓	Retrieves the next student record
✕	Deletes the current student record
□	Inserts a new student record

Table 8.

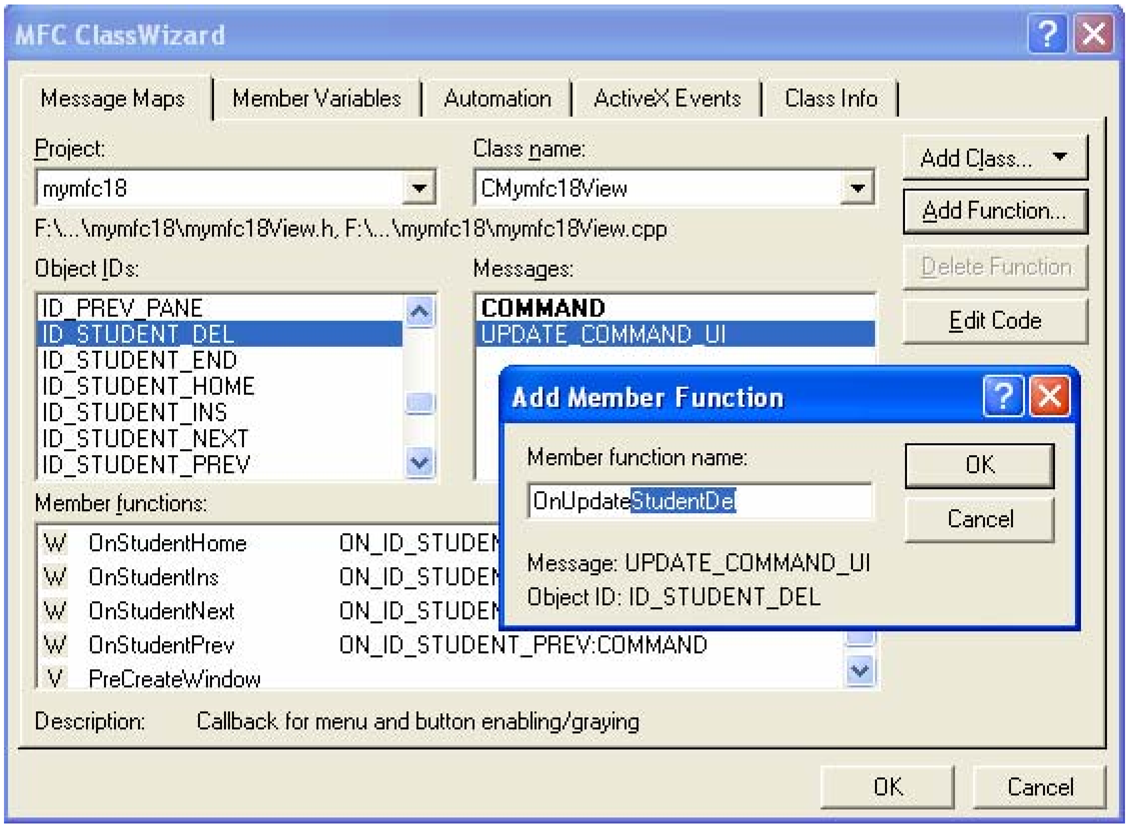
Back to the `CMymfc18View` class.

	Object ID	**Message**	**Member Function**
⤒	`ID_STUDENT_HOME`	`COMMAND`	`OnStudentHome()`
⤓	`ID_STUDENT_END`	`COMMAND`	`OnStudentEnd()`
↑	`ID_STUDENT_PREV`	`COMMAND`	`OnStudentPrev()`
↓	`ID_STUDENT_NEXT`	`COMMAND`	`OnStudentNext()`
✕	`ID_STUDENT_INS`	`COMMAND`	`OnStudentIns()`
□	`ID_STUDENT_DEL`	`COMMAND`	`OnStudentDel()`

Table 9.

Object ID	**Message**	**Member Function**
`ID_STUDENT_HOME`	`UPDATE_COMMAND_UI`	`OnUpdateStudentHome()`
`ID_STUDENT_END`	`UPDATE_COMMAND_UI`	`OnUpdateStudentEnd()`
`ID_STUDENT_PREV`	`UPDATE_COMMAND_UI`	`OnUpdateStudentHome()`
`ID_STUDENT_NEXT`	`UPDATE_COMMAND_UI`	`OnUpdateStudentEnd()`
`ID_STUDENT_DEL`	`UPDATE_COMMAND_UI`	`OnUpdateCommandDel()`

Table 10.



Project: mymfc18

Class name: CMymfc18View

F:\...\mymfc18\mymfc18View.h, F:\...\mymfc18\mymfc18View.cpp

Object IDs:

```
ID_PREV_PANE
ID_STUDENT_DEL
ID_STUDENT_END
ID_STUDENT_HOME
ID_STUDENT_INS
ID_STUDENT_NEXT
ID_STUDENT_PREV
```

Messages:

```
COMMAND
UPDATE_COMMAND_UI
```

Member functions:

```
W   OnStudentHome     ON_ID_STUDEN
W   OnStudentIns      ON_ID_STUDEN
W   OnStudentNext     ON_ID_STUDEN
W   OnStudentPrev     ON_ID_STUDENT_PREV:COMMAND
V   PreCreateWindow
```

Description: Callback for menu and button enabling/graying

Add Member Function

Member function name: OnUpdateStudentDel

Message: UPDATE_COMMAND_UI

Object ID: ID_STUDENT_DEL

Figure 18: Adding message handlers for toolbar buttons.

Now we are ready for the coding part.

`CMymfc18App` Class

In the `CMymfc18App` source code listing, the `OpenDocumentFile()` member function is overridden only for the purpose of inserting a `TRACE` statement. Also, a few lines have been added before the `ProcessShellCommand()` call in `InitInstance()`. They check the argument to `ProcessShellCommand()` and change it if necessary to prevent the creation of any empty document window on startup. Listing 5 shows the source code.

```
MYMFC18.H
// mymfc18.h : main header file for the MYMFC18 application
//

#if
!defined(AFX_MYMFC18_H__7B5FE267_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_)
#define AFX_MYMFC18_H__7B5FE267_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_

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

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CMymfc18App:
// See mymfc18.cpp for the implementation of this class
//
```

```
class CMymfc18App : public CWinApp
{
public:
    CMymfc18App();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMymfc18App)
    public:
    virtual BOOL InitInstance();
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName);
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CMymfc18App)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions
here.
        //    DO NOT EDIT what you see in these blocks of generated code
!
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif //
!defined(AFX_MYMFC18_H__7B5FE267_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_)
```

MYMFC18.CPP

```
// mymfc18.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "mymfc18.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "Mymfc18Doc.h"
#include "Mymfc18View.h"

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

/////////////////////////////////////////////////////////////////////////////
// CMymfc18App

BEGIN_MESSAGE_MAP(CMymfc18App, CWinApp)
    //{{AFX_MSG_MAP(CMymfc18App)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros
here.
        //    DO NOT EDIT what you see in these blocks of generated
code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMymfc18App construction

CMymfc18App::CMymfc18App()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CMymfc18App object

CMymfc18App theApp;

/////////////////////////////////////////////////////////////////////////////
// CMymfc18App initialization

BOOL CMymfc18App::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

    // Change the registry key under which our settings are stored.
    // TODO: You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings();  // Load standard INI file options
                               //  (including MRU)
    // Register the application's document templates.  Document
    //  templates serve as the connection between documents, frame
    // windows and views.

    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_MYMFC1TYPE,
        RUNTIME_CLASS(CStudentDoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CStudentView));
    AddDocTemplate(pDocTemplate);

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);
    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // no empty document window on startup
    if (cmdInfo.m_nShellCommand == CCommandLineInfo::FileNew) {
        cmdInfo.m_nShellCommand = CCommandLineInfo::FileNothing;
    }

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}
//////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
                                                     // support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
// App command to run the dialog
void CMymfc18App::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}
//////////////////////////////////////////////////////////////////////
// CMymfc18App message handlers

CDocument* CMymfc18App::OpenDocumentFile(LPCTSTR lpszFileName)
{
    TRACE("CMymfc18App::OpenDocumentFile\n");
    return CWinApp::OpenDocumentFile(lpszFileName);
}
```

Listing 5: The `CMymfc18App` source code listing.

`CMainFrame` Class

This main frame class, listed in Listing 6, is almost identical to the SDI version, except that it's derived from `CMDIFrameWnd` instead of `CFrameWnd`.

MAINFRM.H

```
// MainFrm.h : interface of the CMainFrame class
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_MAINFRM_H__7B5FE26B_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_)
#define AFX_MAINFRM_H__7B5FE26B_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_

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

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Attributes
public:
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

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

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;
// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        // NOTE - the ClassWizard will add and remove member functions
here.
        //    DO NOT EDIT what you see in these blocks of generated
code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif //
!defined(AFX_MAINFRM_H__7B5FE26B_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_)
```

MAINFRM.CPP

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "mymfc18.h"

#include "MainFrm.h"

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

//////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros
here.
        //    DO NOT EDIT what you see in these blocks of generated code
!
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

//////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
}
```

```
CMainFrame::~CMainFrame()
{
}

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

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD
        | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS
        | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }
    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
    }

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if ( !CMDIFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return TRUE;
}
/////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

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

Listing 6: The `CMainFrame` class listing.

CChildFrame Class

This child frame class, listed in Listing 7, lets you conveniently control the child frame window's characteristics by adding code in the `PreCreateWindow()` function. You can also map messages and override other virtual functions.

CHILDFRM.H

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CHILDFRM_H__7B5FE26D_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_)
#define AFX_CHILDFRM_H__7B5FE26D_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_

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

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void ActivateFrame(int nCmdShow = -1);
    //}}AFX_VIRTUAL

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

// Generated message map functions
protected:
    //{{AFX_MSG(CChildFrame)
        // NOTE - the ClassWizard will add and remove member functions
here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif //
```

```
!defined(AFX_CHILDFRM_H__7B5FE26D_85E9_11D0_8FE3_00C04FC2A0C2__INCLUDED_)
```

CHILDFRM.CPP

```
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "mymfc18.h"

#include "ChildFrm.h"

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

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
        // NOTE - the ClassWizard will add and remove mapping macros
here.
        //    DO NOT EDIT what you see in these blocks of generated code
!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{
    // TODO: add member initialization code here
}

CChildFrame::~CChildFrame()
{
}

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

    if ( !CMDIChildWnd::PreCreateWindow(cs) )
        return FALSE;

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif //_DEBUG

//////////////////////////////////////////////////////////////////////
// CChildFrame message handlers

void CChildFrame::ActivateFrame(int nCmdShow)
{
    TRACE("Entering CChildFrame::ActivateFrame\n");
    CMDIChildWnd::ActivateFrame(nCmdShow);
}
```

Listing 7: The `CChildFrame` class listing.

CMymfc18Doc Class

AppWizard originally generated the `CMymfc18Doc` class. Listing 8 shows the code used in the MYMFC18 example and the class is the same as the `CMymfc17Doc` class from the previous module.

```
MYMFC18DOC.H
//MYMFC18DOC.H
// MYMFC18DOC.h : interface of the CMYMFC18DOC class
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_MYMFC18DOC_H__4D011047_7E1C_11D0_8FE0_00C04FC2A0C2__INCLUDED_)
#define AFX_MYMFC18DOC_H__4D011047_7E1C_11D0_8FE0_00C04FC2A0C2__INCLUDED_

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

#include "student.h"

class CMymfc18Doc : public CDocument
{
protected: // create from serialization only
    CMymfc18Doc();
    DECLARE_DYNCREATE(CMymfc18Doc)

// Attributes
public:
    CStudentList* GetList() {
        return &m_studentList;
    }

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMYMFC18DOC)
       public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void DeleteContents();
       virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
       //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMymfc18Doc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CMYMFC18DOC)
    afx_msg void OnEditClearAll();
    afx_msg void OnUpdateEditClearAll(CCmdUI* pCmdUI);
       afx_msg void OnUpdateFileSave(CCmdUI* pCmdUI);
       //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CStudentList m_studentList;
};

/////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif //
!defined(AFX_MYMFC18DOC_H__4D011047_7E1C_11D0_8FE0_00C04FC2A0C2__INCLUDED_)
```

MYMFC18DOC.CPP

```
// MYMFC18DOC.cpp : implementation of the CMYMFC18DOC class
//

#include "stdafx.h"
#include "mymfc18.h"

#include "mymfc18Doc.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////
// CMYMFC18DOC

IMPLEMENT_DYNCREATE(CMymfc18Doc, CDocument)

BEGIN_MESSAGE_MAP(CMymfc18Doc, CDocument)
    //{{AFX_MSG_MAP(CMYMFC18DOC)
    ON_COMMAND(ID_EDIT_CLEAR_ALL, OnEditClearAll)
    ON_UPDATE_COMMAND_UI(ID_EDIT_CLEAR_ALL, OnUpdateEditClearAll)
       ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
       //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////
// CMYMFC18DOC construction/destruction

CMymfc18Doc::CMymfc18Doc()
{
    TRACE("Entering CMYMFC18DOC constructor\n");
#ifdef _DEBUG
    afxDump.SetDepth(1); // Ensure dump of list elements
#endif // _DEBUG
}
```

```
CMymfc18Doc::~CMymfc18Doc()
{
}

BOOL CMymfc18Doc::OnNewDocument()
{
    TRACE("Entering CMymfc18Doc::OnNewDocument\n");
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add re-initialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}
//////////////////////////////////////////////////////////////////////
// CMymfc18Doc serialization

void CMymfc18Doc::Serialize(CArchive& ar)
{
    TRACE("Entering CMymfc18Doc::Serialize\n");
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
    m_studentList.Serialize(ar);
}

//////////////////////////////////////////////////////////////////////
// CMymfc18Doc diagnostics

#ifdef _DEBUG
void CMymfc18Doc::AssertValid() const
{
    CDocument::AssertValid();
}

void CMymfc18Doc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
    dc << "\n" << m_studentList << "\n";
}
#endif //_DEBUG

//////////////////////////////////////////////////////////////////////
// CMymfc18Doc commands

void CMymfc18Doc::DeleteContents()
{
TRACE("Entering CMymfc18Doc::DeleteContents\n");
    while (m_studentList.GetHeadPosition())
       {
        delete m_studentList.RemoveHead();
    }
}

void CMymfc18Doc::OnEditClearAll()
{
    DeleteContents();
    UpdateAllViews(NULL);
}
```

```
void CMymfc18Doc::OnUpdateEditClearAll(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(!m_studentList.IsEmpty());
}

BOOL CMymfc18Doc::OnOpenDocument(LPCTSTR lpszPathName)
{
    TRACE("Entering CMymfc18Doc::OnOpenDocument\n");
     if (!CDocument::OnOpenDocument(lpszPathName))
        return FALSE;

    // TODO: Add your specialized creation code here

    return TRUE;
}

void CMymfc18Doc::OnUpdateFileSave(CCmdUI* pCmdUI)
{
       // TODO: Add your command update UI handler code here
       pCmdUI->Enable(IsModified());

}
```

Figure 8: The `CMymfc18Doc` class listing.

`CMymfc18View` Class

Listing 9 shows the code for the `CMymfc18View` class similar with the MYMFC17 and MYMFC16 program examples.

MYMFC18VIEW.H

```
// Mymfc18View.h : interface of the CMymfc18View class
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_MYMFC18VIEW_H__4D011049_7E1C_11D0_8FE0_00C04FC2A0C2__INCLUDED_)
#define AFX_MYMFC18VIEW_H__4D011049_7E1C_11D0_8FE0_00C04FC2A0C2__INCLUDED_

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

class CMymfc18View : public CFormView
{
protected:
    POSITION      m_position; // current position in document list
    CStudentList* m_pList;    // copied from document

protected: // create from serialization only
    CMymfc18View();
    DECLARE_DYNCREATE(CMymfc18View)

public:
    //{{AFX_DATA(CMymfc18View)
    enum { IDD = IDD_MYMFC18_FORM };
    int     m_nGrade;
    CString m_strName;
    //}}AFX_DATA

// Attributes
public:
    CMymfc18Doc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMymfc18View)
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    virtual void OnInitialUpdate(); // called first time after construct
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMymfc18View();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    virtual void ClearEntry();
    virtual void InsertEntry(POSITION position);
    virtual void GetEntry(POSITION position);

// Generated message map functions
protected:
    //{{AFX_MSG(CMymfc18View)
    afx_msg void OnClear();
    afx_msg void OnStudentHome();
    afx_msg void OnStudentEnd();
    afx_msg void OnStudentPrev();
    afx_msg void OnStudentNext();
    afx_msg void OnStudentIns();
    afx_msg void OnStudentDel();
    afx_msg void OnUpdateStudentHome(CCmdUI* pCmdUI);
    afx_msg void OnUpdateStudentEnd(CCmdUI* pCmdUI);
    afx_msg void OnUpdateStudentDel(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in Mymfc18View.cpp
inline CMymfc18Doc* CMymfc18View::GetDocument()
   { return (CMymfc18Doc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif //
!defined(AFX_MYMFC18VIEW_H__4D011049_7E1C_11D0_8FE0_00C04FC2A0C2__INCLUDED_)
```

MYMFC18VIEW.CPP

```
// Mymfc18View.cpp : implementation of the CMymfc18View class
//

#include "stdafx.h"
#include "mymfc18.h"
```

```
#include "MYMFC18DOC.h"
#include "Mymfc18View.h"

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

//////////////////////////////////////////////////////////////////////
// CMymfc18View

IMPLEMENT_DYNCREATE(CMymfc18View, CFormView)
BEGIN_MESSAGE_MAP(CMymfc18View, CFormView)
    //{{AFX_MSG_MAP(CMymfc18View)
    ON_BN_CLICKED(IDC_CLEAR, OnClear)
    ON_COMMAND(ID_STUDENT_HOME, OnStudentHome)
    ON_COMMAND(ID_STUDENT_END, OnStudentEnd)
    ON_COMMAND(ID_STUDENT_PREV, OnStudentPrev)
    ON_COMMAND(ID_STUDENT_NEXT, OnStudentNext)
    ON_COMMAND(ID_STUDENT_INS, OnStudentIns)
    ON_COMMAND(ID_STUDENT_DEL, OnStudentDel)
    ON_UPDATE_COMMAND_UI(ID_STUDENT_HOME, OnUpdateStudentHome)
    ON_UPDATE_COMMAND_UI(ID_STUDENT_END, OnUpdateStudentEnd)
    ON_UPDATE_COMMAND_UI(ID_STUDENT_PREV, OnUpdateStudentHome)
    ON_UPDATE_COMMAND_UI(ID_STUDENT_NEXT, OnUpdateStudentEnd)
    ON_UPDATE_COMMAND_UI(ID_STUDENT_DEL, OnUpdateStudentDel)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////
// CMymfc18View construction/destruction

CMymfc18View::CMymfc18View() : CFormView(CMymfc18View::IDD)
{
    TRACE("Entering CMymfc18View constructor\n");
    //{{AFX_DATA_INIT(CMymfc18View)
    m_nGrade = 0;
    m_strName = _T("");
    //}}AFX_DATA_INIT
    m_position = NULL;
}

CMymfc18View::~CMymfc18View()
{
}

void CMymfc18View::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMymfc18View)
    DDX_Text(pDX, IDC_GRADE, m_nGrade);
    DDV_MinMaxInt(pDX, m_nGrade, 0, 100);
    DDX_Text(pDX, IDC_NAME, m_strName);
    DDV_MaxChars(pDX, m_strName, 20);
    //}}AFX_DATA_MAP
}
BOOL CMymfc18View::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    return CFormView::PreCreateWindow(cs);
}

void CMymfc18View::OnInitialUpdate()
```

```cpp
{
    TRACE("Entering CMymfc18View::OnInitialUpdate\n");
    m_pList = GetDocument()->GetList();
    CFormView::OnInitialUpdate();
}

//////////////////////////////////////////////////////////////////////
// CMymfc18View diagnostics

#ifdef _DEBUG
void CMymfc18View::AssertValid() const
{
    CFormView::AssertValid();
}

void CMymfc18View::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CMymfc18Doc* CMymfc18View::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMymfc18Doc)));
    return (CMymfc18Doc*)m_pDocument;
}
#endif //_DEBUG

//////////////////////////////////////////////////////////////////////
// CMymfc18View message handlers

void CMymfc18View::OnClear()
{
    TRACE("Entering CMymfc18View::OnClear\n");
    ClearEntry();
}

void CMymfc18View::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    // called by OnInitialUpdate and by UpdateAllViews
    TRACE("Entering CMymfc18View::OnUpdate\n");
    m_position = m_pList->GetHeadPosition();
    GetEntry(m_position); // initial data for view
}

void CMymfc18View::OnStudentHome()
{
    TRACE("Entering CMymfc18View::OnStudentHome\n");
    // need to deal with list empty condition
    if (!m_pList->IsEmpty()) {
        m_position = m_pList->GetHeadPosition();
        GetEntry(m_position);
    }
}

void CMymfc18View::OnStudentEnd()
{
    TRACE("Entering CMymfc18View::OnStudentEnd\n");
    if (!m_pList->IsEmpty()) {
        m_position = m_pList->GetTailPosition();
        GetEntry(m_position);
    }
}

void CMymfc18View::OnStudentPrev()
{
    POSITION pos;
    TRACE("Entering CMymfc18View::OnStudentPrev\n");
    if ((pos = m_position) != NULL) {
        m_pList->GetPrev(pos);
        if (pos) {
            GetEntry(pos);
      m_position = pos;
        }
    }
}

void CMymfc18View::OnStudentNext()
{
    POSITION pos;
    TRACE("Entering CMymfc18View::OnStudentNext\n");
    if ((pos = m_position) != NULL) {
        m_pList->GetNext(pos);
        if (pos) {
            GetEntry(pos);
            m_position = pos;
        }
    }
}
void CMymfc18View::OnStudentIns()
{
    TRACE("Entering CMymfc18View::OnStudentIns\n");
    InsertEntry(m_position);
    GetDocument()->SetModifiedFlag();
    GetDocument()->UpdateAllViews(this);
}

void CMymfc18View::OnStudentDel()
{
    // deletes current entry and positions to next one or head
    POSITION pos;
    TRACE("Entering CMymfc18View::OnStudentDel\n");
    if ((pos = m_position) != NULL) {
        m_pList->GetNext(pos);
        if (pos == NULL) {
            pos = m_pList->GetHeadPosition();
            TRACE("GetHeadPos = %ld\n", pos);
            if (pos == m_position) {
                pos = NULL;
            }
        }
        GetEntry(pos);
        CStudent* ps = m_pList->GetAt(m_position);
        m_pList->RemoveAt(m_position);
        delete ps;
        m_position = pos;
        GetDocument()->SetModifiedFlag();
        GetDocument()->UpdateAllViews(this);
    }
}

void CMymfc18View::OnUpdateStudentHome(CCmdUI* pCmdUI)
{
    // called during idle processing and when Student menu drops down
    POSITION pos;

    // enables button if list not empty and not at home already
    pos = m_pList->GetHeadPosition();
    pCmdUI->Enable((m_position != NULL) && (pos != m_position));
}

void CMymfc18View::OnUpdateStudentEnd(CCmdUI* pCmdUI)
{
    // called during idle processing and when Student menu drops down
    POSITION pos;

    // enables button if list not empty and not at end already
    pos = m_pList->GetTailPosition();
    pCmdUI->Enable((m_position != NULL) && (pos != m_position));
}

void CMymfc18View::OnUpdateStudentDel(CCmdUI* pCmdUI)
{
    // called during idle processing and when Student menu drops down
    pCmdUI->Enable(m_position != NULL);
}

void CMymfc18View::GetEntry(POSITION position)
{
    if (position) {
        CStudent* pStudent = m_pList->GetAt(position);
        m_strName = pStudent->m_strName;
        m_nGrade = pStudent->m_nGrade;
    }
    else {
        ClearEntry();
    }
    UpdateData(FALSE);
}

void CMymfc18View::InsertEntry(POSITION position)
{
    if (UpdateData(TRUE)) {
        // UpdateData returns FALSE if it detects a user error
        CStudent* pStudent = new CStudent;
        pStudent->m_strName = m_strName;
        pStudent->m_nGrade = m_nGrade;
        m_position = m_pList->InsertAfter(m_position, pStudent);
    }
}

void CMymfc18View::ClearEntry()
{
    m_strName = "";
    m_nGrade = 0;
    UpdateData(FALSE);
    ((CDialog*) this)->GotoDlgCtrl(GetDlgItem(IDC_NAME));
}
```

Listing 9: The `CMymfc18View` class listing.

CStudent Class

The following steps and code are for the `CStudent` class. It is similar to the project example of the previous module, add to the project new **Student.h** and **Student.cpp** files and paste the codes. Then do some code modification.

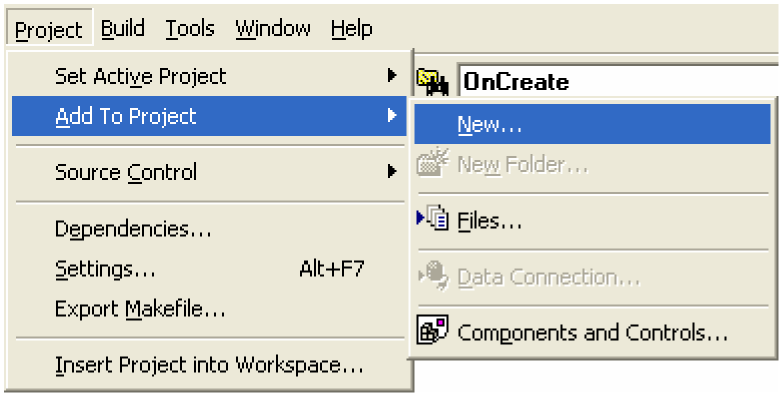
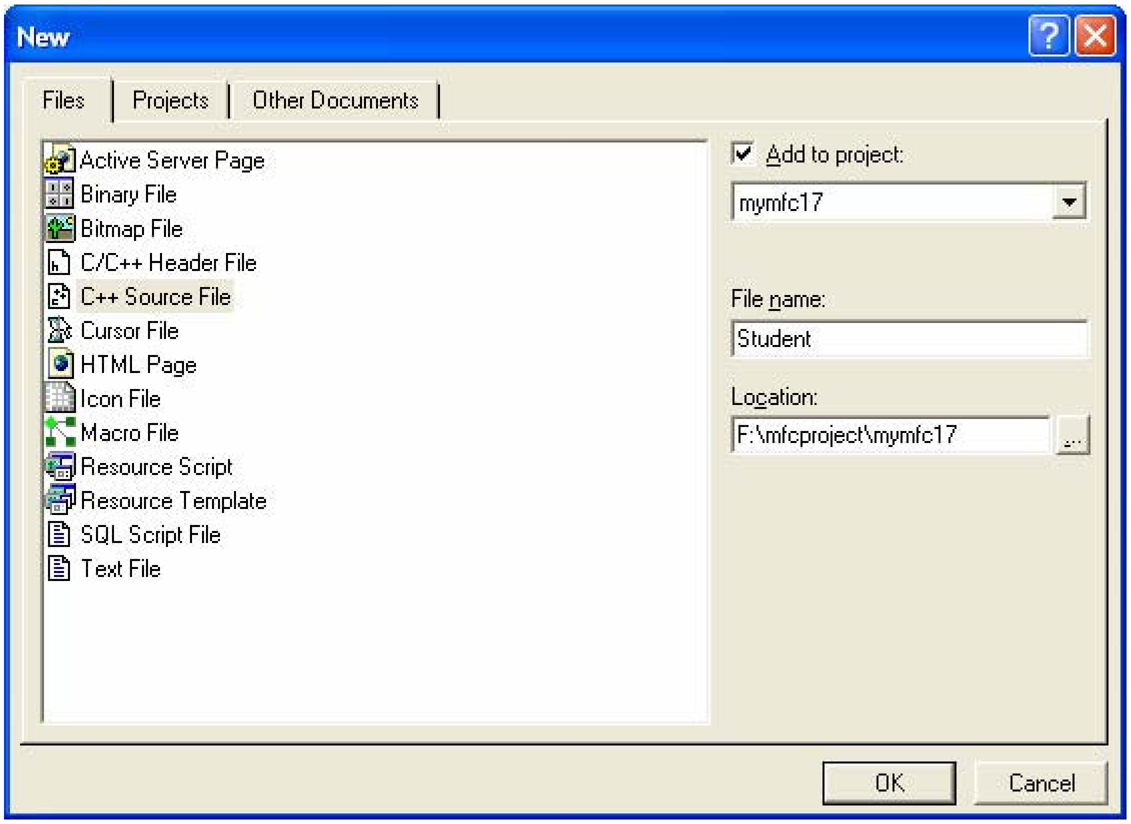


Figure 19: Creating and adding the `CStudent` class files.



Add to project: ✓
mymfc17

File name:
Student

Location:
F:\mfcproject\mymfc17

Figure 20: Entering the **Student.cpp** file name.

STUDENT.H

```
// student.h

#ifndef _INSIDE_VISUAL_CPP_STUDENT
#define _INSIDE_VISUAL_CPP_STUDENT

class CStudent : public CObject
{
    DECLARE_SERIAL(CStudent)
public:
    CString m_strName;
    int m_nGrade;

    CStudent()
    {
        m_nGrade = 0;
    }

    CStudent(const char* szName, int nGrade) : m_strName(szName)
    {
        m_nGrade = nGrade;
    }

    CStudent(const CStudent& s) : m_strName(s.m_strName)
    {
        // copy constructor
        m_nGrade = s.m_nGrade;
    }

    const CStudent& operator =(const CStudent& s)
    {
        m_strName = s.m_strName;
        m_nGrade = s.m_nGrade;
        return *this;
    }

    BOOL operator ==(const CStudent& s) const
    {
        if ((m_strName == s.m_strName) && (m_nGrade == s.m_nGrade))
              {
            return TRUE;
        }
        else
              {
            return FALSE;
        }
    }

    BOOL operator !=(const CStudent& s) const
    {
        // Let's make use of the operator we just defined!
        return !(*this == s);
    }
#ifdef _DEBUG
    void Dump(CDumpContext& dc) const;
#endif // _DEBUG
};

#endif // _INSIDE_VISUAL_CPP_STUDENT

typedef CTypedPtrList<CObList, CStudent*> CStudentList;
```

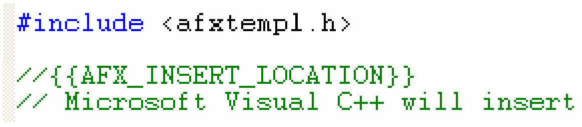
STUDENT.CPP

```
#include "stdafx.h"
#include "student.h"

IMPLEMENT_SERIAL(CStudent, CObject, 0)

#ifdef _DEBUG
void CStudent::Dump(CDumpContext& dc) const
{
    CObject::Dump(dc);
    dc << "m_strName = " << m_strName << "\nm_nGrade = " <<m_nGrade;
}
#endif // _DEBUG
```

Listing 10.

The use of the MFC template collection classes requires the following statement in **StdAfx.h**:

```
#include <afxtempl.h>
```



```
#include <afxtempl.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert
```

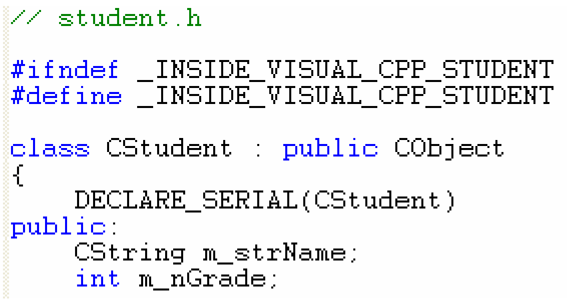
Listing 11.

The MYMFC18 **Student.h** file is almost the same as the file in the MYMFC16 project. The header contains the macro:

```
DECLARE_SERIAL(CStudent)
```

instead of:

```
DECLARE_DYNAMIC(CStudent)
```



```
// student.h

#ifndef _INSIDE_VISUAL_CPP_STUDENT
#define _INSIDE_VISUAL_CPP_STUDENT

class CStudent : public CObject
{
    DECLARE_SERIAL(CStudent)
public:
    CString m_strName;
    int m_nGrade;
```

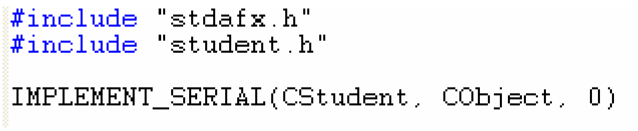
Listing 12.

and the implementation file contains the macro:

```
IMPLEMENT_SERIAL(CStudent, CObject, 0)
```

instead of:

```
IMPLEMENT_DYNAMIC(CStudent, Cobject)
```



```
#include "stdafx.h"
#include "student.h"

IMPLEMENT_SERIAL(CStudent, CObject, 0)
```

Listing 13.

The virtual Serialize function has also been added.

Testing the MYMFC18 Application

Do the build, run the program from Visual C++, and then make several documents.

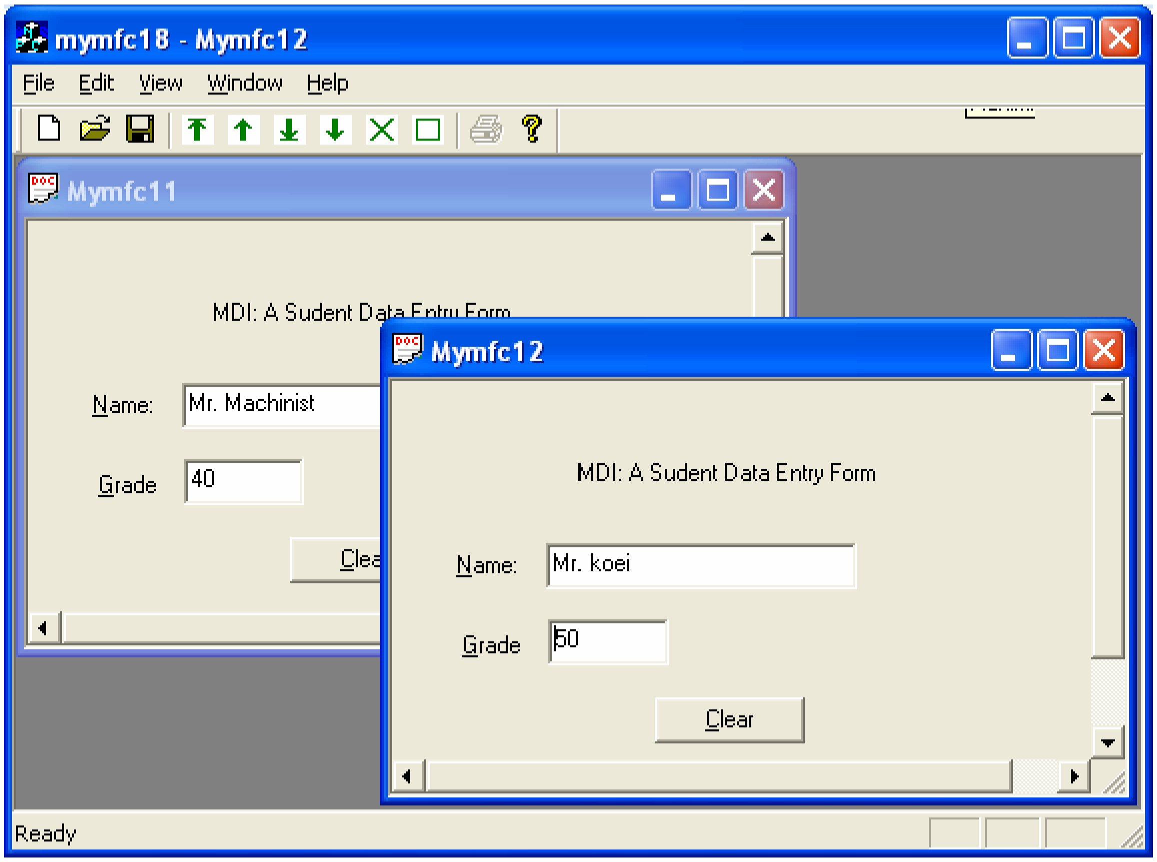


Figure 21: MYMFC18's MDI program output with serialization in action.

Try saving the documents on disk, closing them, and reloading them. Also, choose **New Window** from the **Window** menu. Notice that you now have two views (and child frames) that attached to the same document.

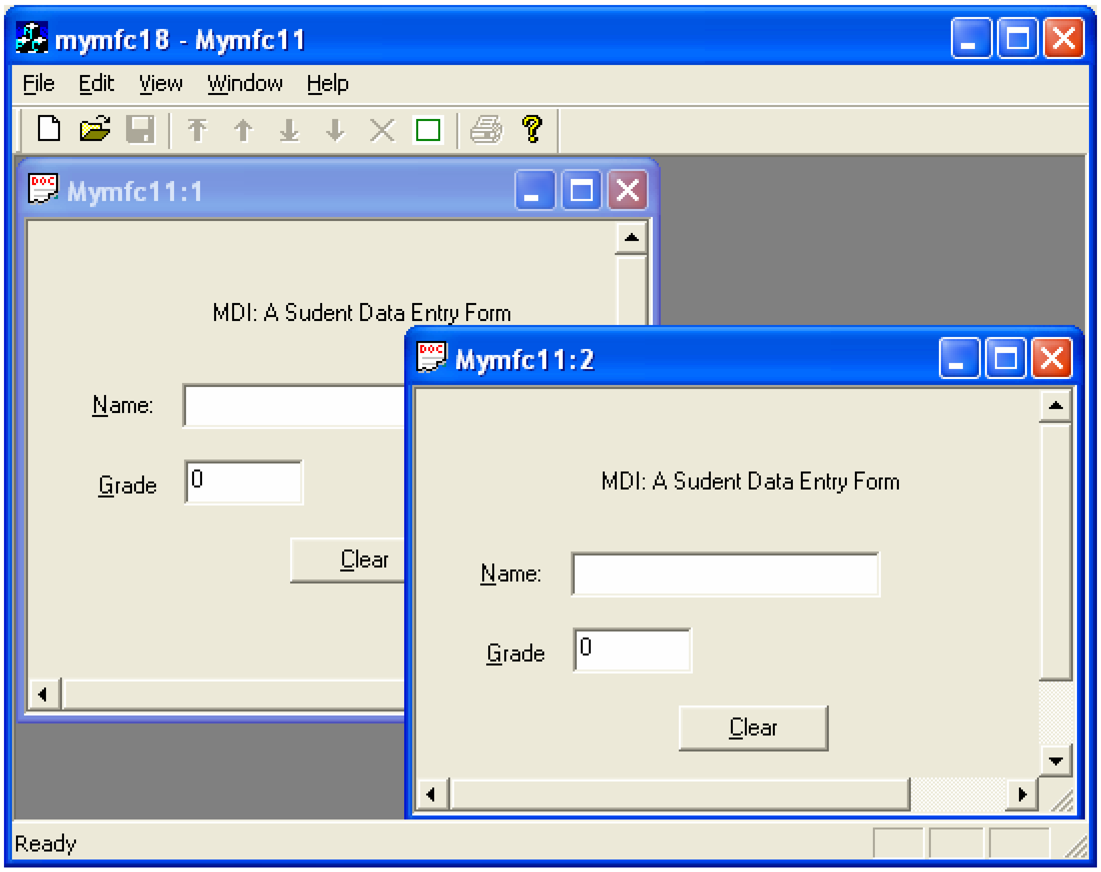


Figure 22: Opening the saved MDI documents.

Now exit the program and start Windows Explorer. The files you created should show up with document icons. Double-click on a document icon and see whether the MYMFC18 program starts up. Now, with both Windows Explorer and MYMFC18 on the desktop, drag a document from Windows Explorer to MYMFC18. Was the file opened?

Further reading and digging:

1. Standard C File Input/Output.
2. Standard C++ File Input/Output.
3. Win32 File Input/Output: Module C, Module D and Module E.
4. MSDN MFC 6.0 class library online documentation - used throughout this Tutorial.
5. MSDN MFC 7.0 class library online documentation - used in .Net framework and also backward compatible with 6.0 class library
6. MSDN Library
7. Windows data type.
8. Win32 programming Tutorial.
9. The best of C/C++, MFC, Windows and other related books.
10. Unicode and Multibyte character set: Story and program examples.