Automation part 3

Program examples compiled using Visual C++ 6.0 compiler on Windows XP Pro machine with Service Pack 2. The Excel version is Excel 2003/Office 11. Topics and sub topics for this tutorial are listed below. Don't forget to read Tenouk's small disclaimer. The supplementary notes for this tutorial are automation, variant and COlevariant class.

Index:

The MYMFC29D Automation Client Example

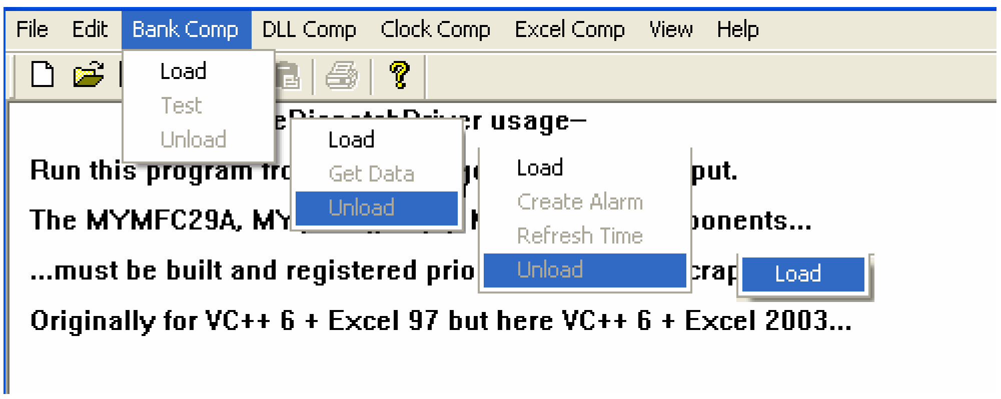
So far, you've seen C++ **Automation component programs**. Now you'll see a C++ **Automation client program** that runs all the previous components and also controls Microsoft Excel. The MYMFC29D program was originally generated by AppWizard, but **without any COM options**. It was easier to add the COM code than it would have been to rip out the component-specific code. If you do use AppWizard to build such an Automation controller, add the following line at the end of **StdAfx.h**:

```
#include <afxdisp.h>
```

Then add this call at the beginning of the application's `InitInstance()` function:

```
AfxOleInit();
```

To prepare MYMFC29D, open the mymfc29D project and do the build. Run the application from the debugger, and you'll see a standard SDI application with a menu structure similar to that shown in Figure 1.
If you have built and registered all the previous components (previous program examples), you can test them from MYMFC29D. Notice that the DLL doesn't have to be copied to the \Winnt\System32 (Windows system) directory because Windows finds it through the Registry. For some components, you'll have to watch the Debug window to verify that the test results are correct. The program is reasonably modular. Menu **commands** and **update command UI** events are mapped to the view class. Each component object has its own C++ controller class and an embedded data member in **mymfc29DView.h**. We'll look at each part separately after we delve into the steps to build MYMFC29D from scratch and type libraries.

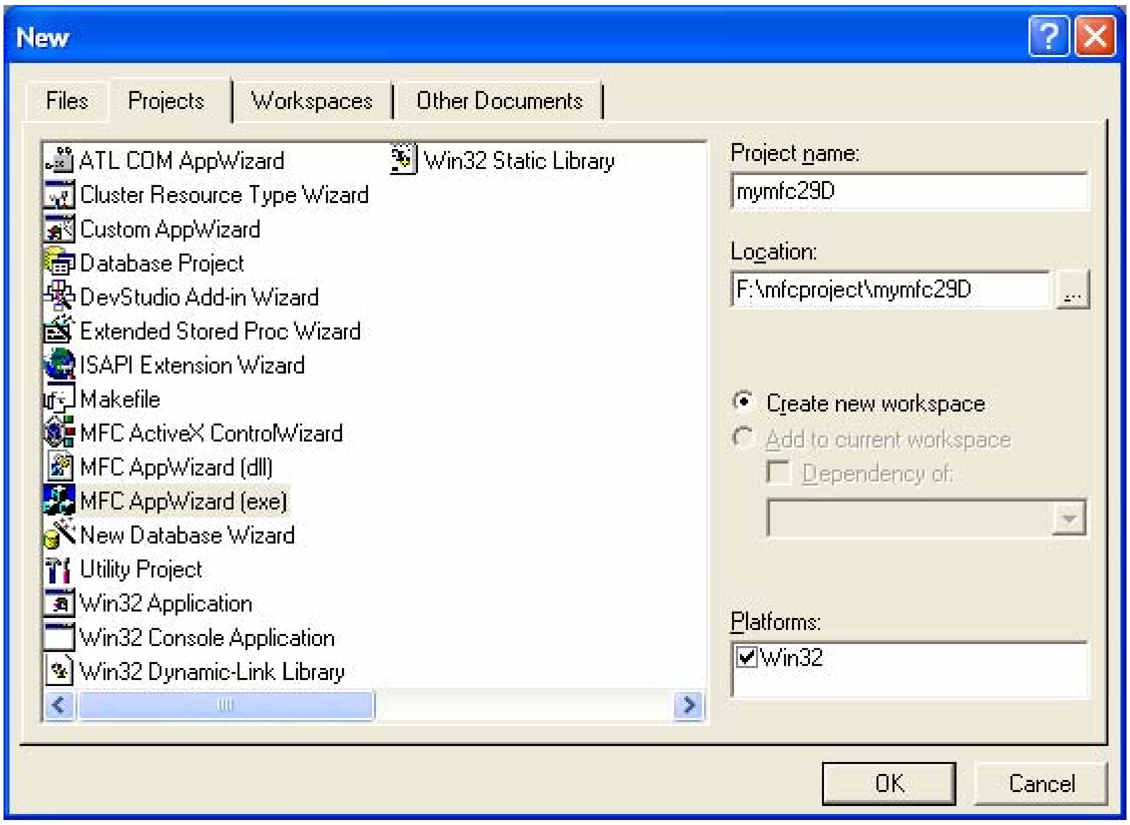


Run this program fro... put.
The MYMFC29A, MY... ponents...
...must be built and registered prio... crap
Originally for VC++ 6 + Excel 97 but here VC++ 6 + Excel 2003...

Figure 1: A sample menu structure for a standard SDI application.

The steps for MYMFC29D from scratch

This is SDI application without Automation support at the beginning.



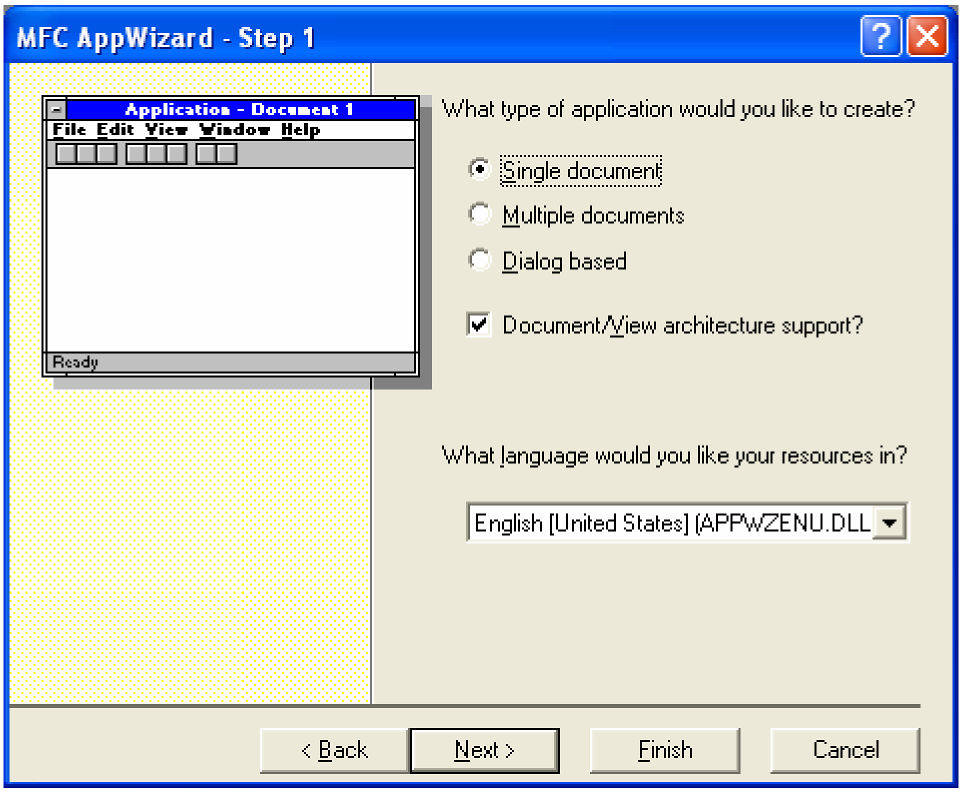
Project name: mymfc29D

Location: F:\mfcproject\mymfc29D

Create new workspace

Platforms: Win32 ✓

Figure 2: MYMFC29D new Visual C++ project dialog.

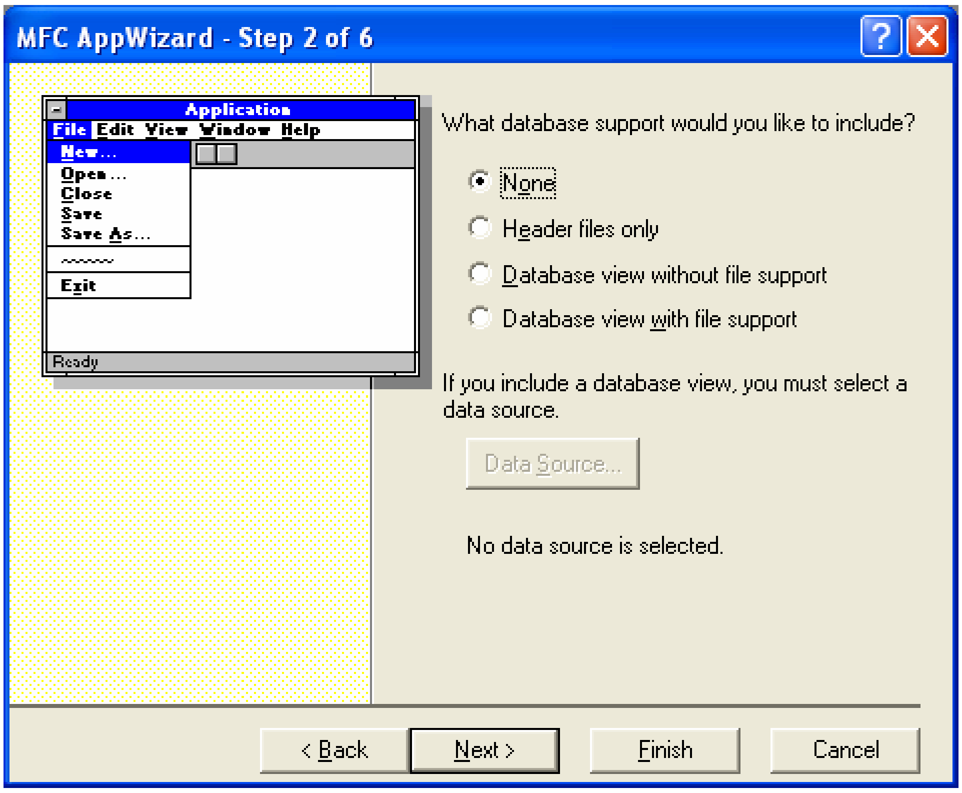


What type of application would you like to create? Single document

Document/View architecture support? ✓

What language would you like your resources in? English [United States] (APPWZENU.DLL

Figure 3: MYMFC29D – AppWizard step 1 of 6, selecting SDI application.



What database support would you like to include?

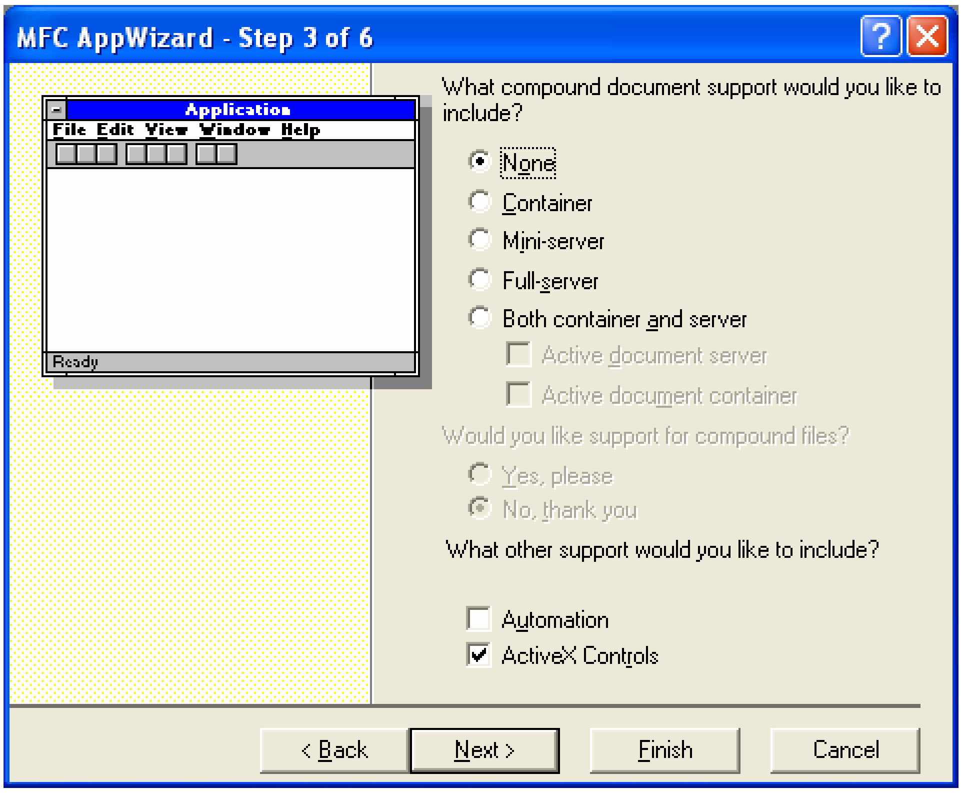
- [✓ None]
- Header files only
- Database view without file support
- Database view with file support

If you include a database view, you must select a data source.

Data Source...

No data source is selected.

Figure 4: MYMFC29D – AppWizard step 2 of 6.



What compound document support would you like to include?

- [✓ None]
- Container
- Mini-server
- Full-server
- Both container and server
  - Active document server
  - Active document container

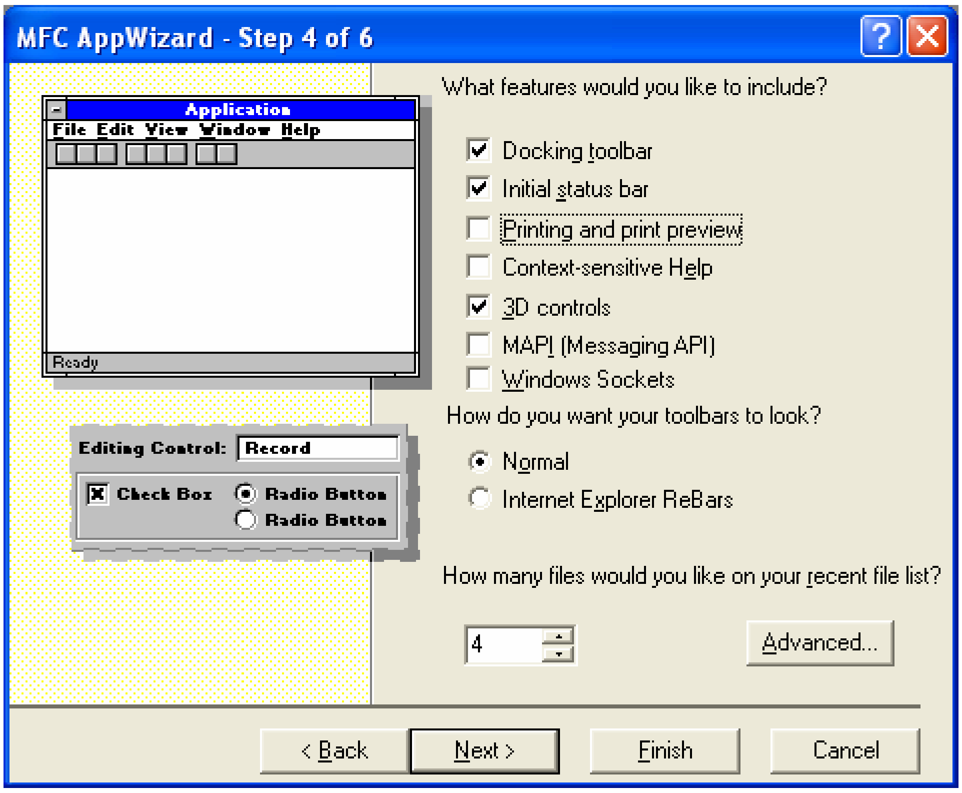
Would you like support for compound files?

- Yes, please
- [✓ No, thank you]

What other support would you like to include?

- Automation
- [✓ ActiveX Controls]

Figure 5: MYMFC29D – AppWizard step 3 of 6, Automation option not selected.



What features would you like to include?

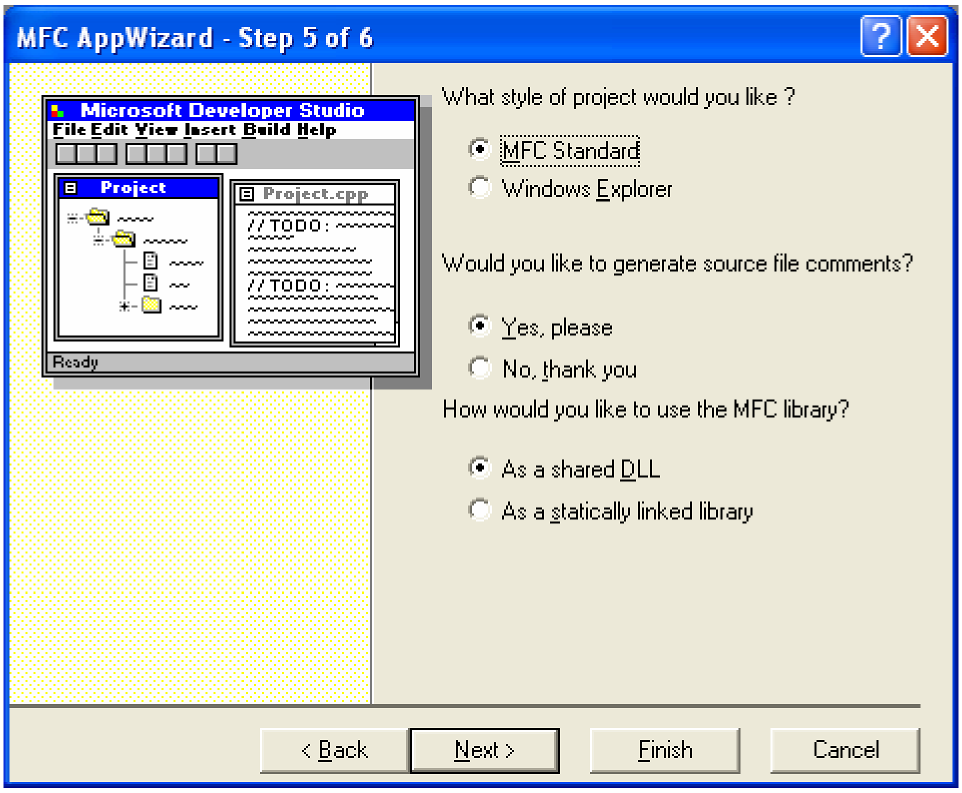
- [✓] Docking toolbar
- [✓] Initial status bar
- [ ] Printing and print preview
- [ ] Context-sensitive Help
- [✓] 3D controls
- [ ] MAPI (Messaging API)
- [ ] Windows Sockets

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 6: MYMFC29D – AppWizard step 4 of 6, deselect the printing services.


MFC AppWizard - Step 5 of 6
Microsoft Developer Studio
File Edit View Insert Build Help
Project
Project.cpp
// TODO:
// TODO:
Ready
< Back
Next >
Finish
Cancel

What style of project would you like ?

[✓ MFC Standard] Windows Explorer

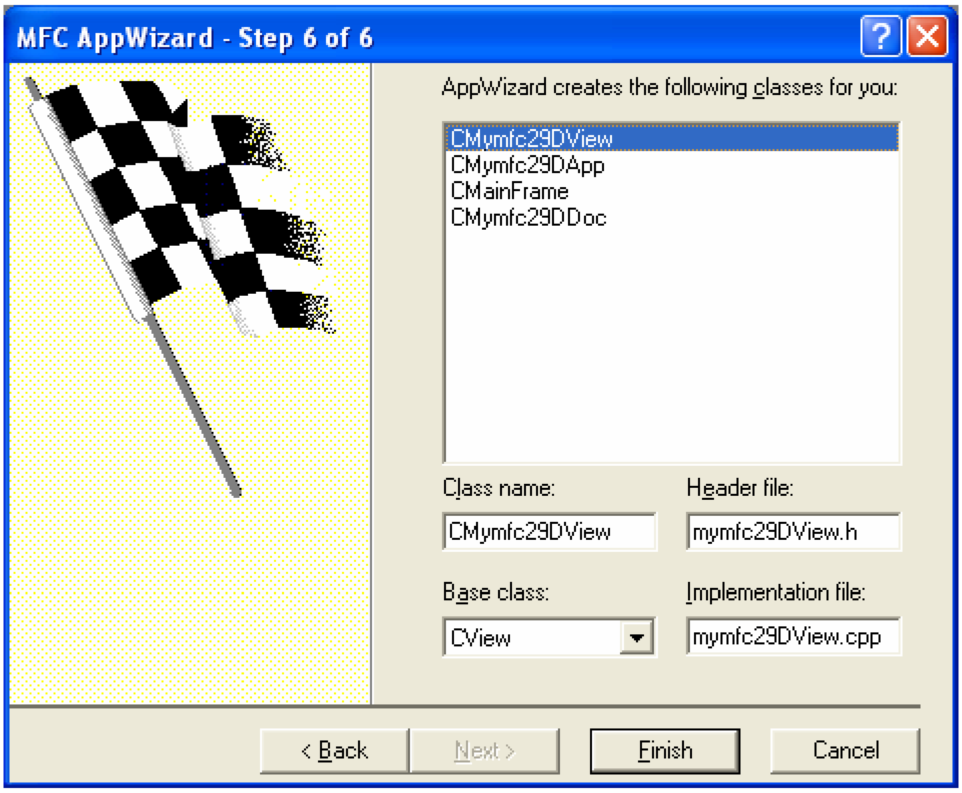
Would you like to generate source file comments?

[✓ Yes, please] No, thank you

How would you like to use the MFC library?

[✓ As a shared DLL] As a statically linked library

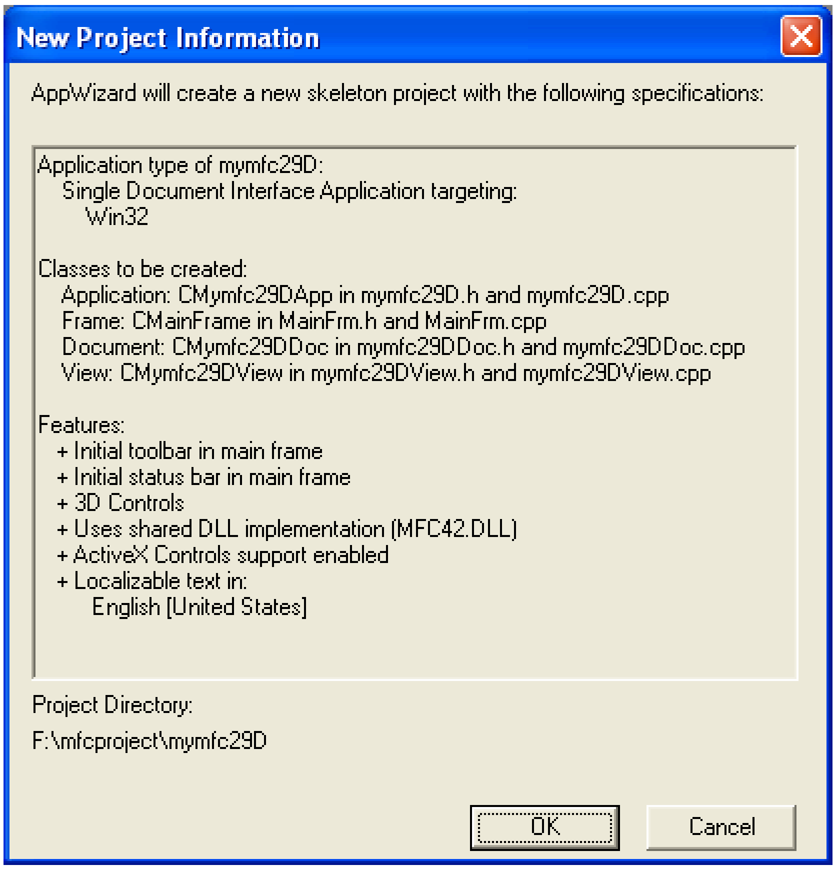
Figure 7: MYMFC29D – AppWizard step 5 of 6.



CMymfc29DView
CMymfc29DApp
CMainFrame
CMymfc29DDoc

Class name: CMymfc29DView
Header file: mymfc29DView.h
Base class: CView
Implementation file: mymfc29DView.cpp

Figure 8: MYMFC29D – AppWizard step 6 of 6.



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

```
Application type of mymfc29D:
    Single Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CMymfc29DApp in mymfc29D.h and mymfc29D.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    Document: CMymfc29DDoc in mymfc29DDoc.h and mymfc29DDoc.cpp
    View: CMymfc29DView in mymfc29DView.h and mymfc29DView.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]
```

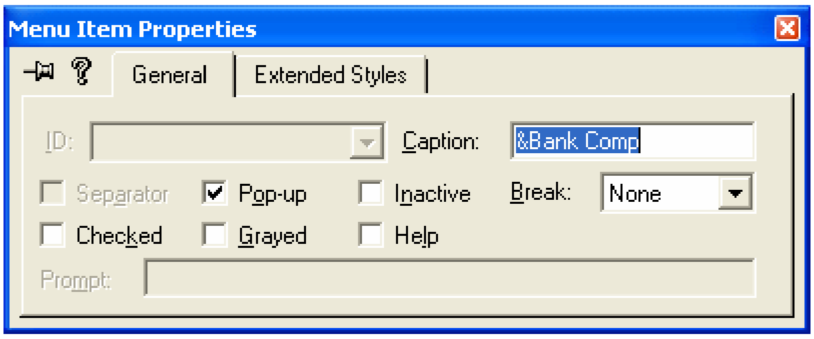
Project Directory:
F:\mfcproject\mymfc29D

Figure 9: MYMFC29D project summary.

Add menus and their items for **Bank Comp**. Use ResourceView to add new menus and their respective items.

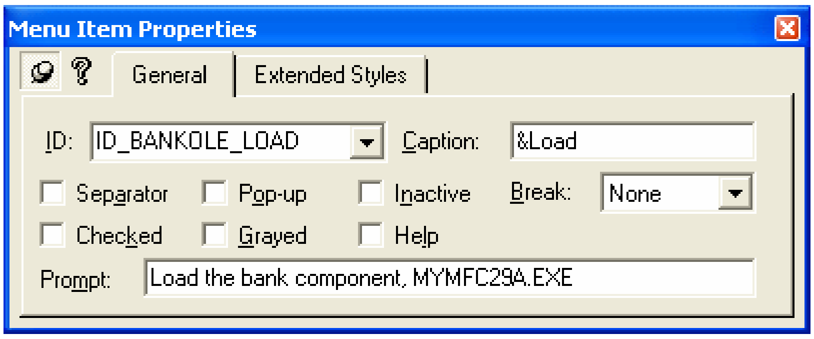
ID	Menu	Prompt
-	Bank Comp	-
ID_BANKOLE_LOAD	Load	Load the bank component, MYMFC29A.EXE
ID_BANKOLE_TEST	Test	Test the bank component
ID_BANKOLE_UNLOAD	Unload	Unload the bank component

Table 1.



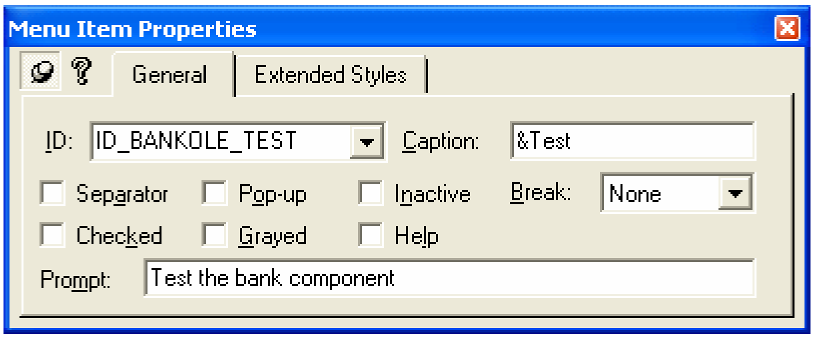
ID:
Caption: &Bank Comp
[ ] Separator [✓] Pop-up [ ] Inactive Break: None
[ ] Checked [ ] Grayed [ ] Help
Prompt:

Figure 10: Adding **Bank Comp** menu.



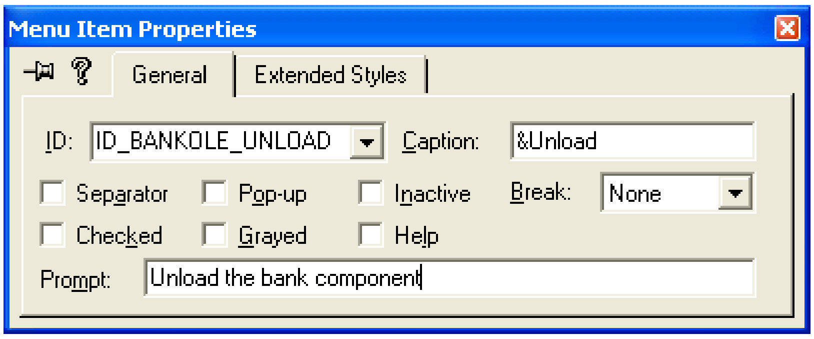
ID: ID_BANKOLE_LOAD
Caption: &Load
[ ] Separator [ ] Pop-up [ ] Inactive Break: None
[ ] Checked [ ] Grayed [ ] Help
Prompt: Load the bank component, MYMFC29A.EXE

Figure 11: Adding **Load** menu item.



ID: ID_BANKOLE_TEST
Caption: &Test
[ ] Separator [ ] Pop-up [ ] Inactive Break: None
[ ] Checked [ ] Grayed [ ] Help
Prompt: Test the bank component

Figure 12: Adding **Test** menu item.


Menu Item Properties

General | Extended Styles

ID: ID_BANKOLE_UNLOAD Caption: &Unload

☐ Separator ☐ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt: Unload the bank component

Figure 13: Adding **Unload** menu item.

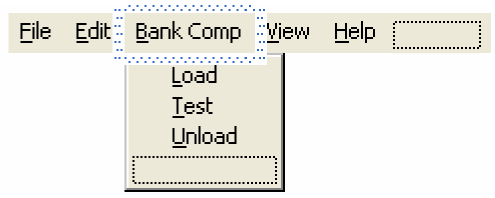
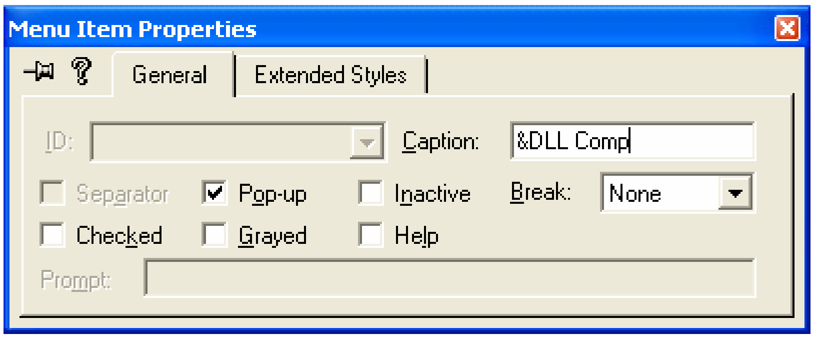


Figure 14: A completed **Bank Comp** menus.

Add menu items for **DLL Comp**.

ID	Menu	Prompt
-	DLL Comp	-
`ID_DLLOLE_LOAD`	Load	Load the DLL component (MYMFC29B.DLL)
`ID_DLLOLE_GETDATA`	Get Data	Get data from the DLL component
`ID_DLLOLE_UNLOAD`	Unload	Unload the DLL component

Table 2.


Menu Item Properties

General | Extended Styles

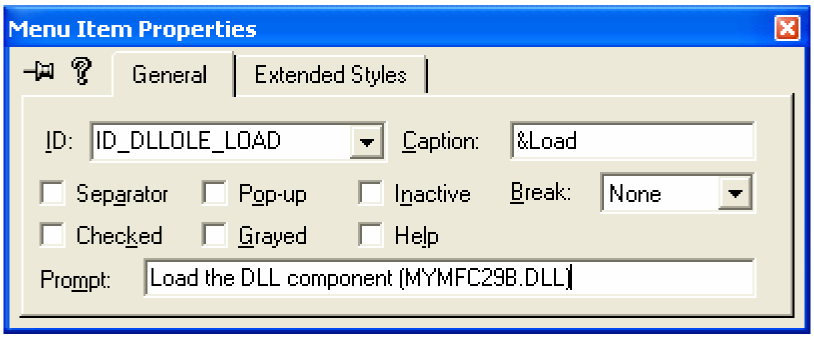
ID: Caption: &DLL Comp

☐ Separator ☑ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt:

Figure 15: Adding **DLL Comp** menu.


Menu Item Properties

General | Extended Styles

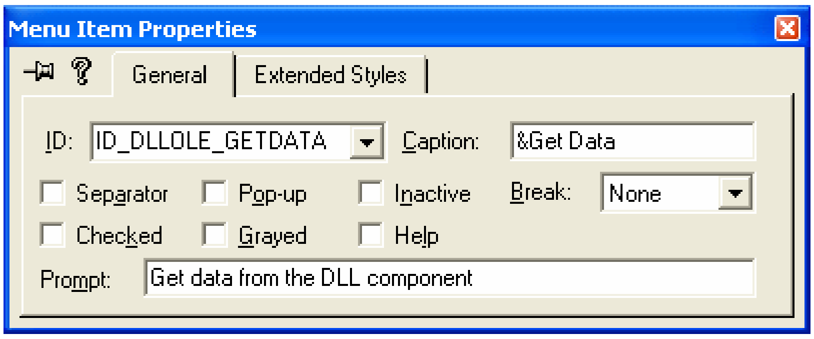
ID: ID_DLLOLE_LOAD Caption: &Load

☐ Separator ☐ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt: Load the DLL component (MYMFC29B.DLL)

Figure 16: Adding **Load** menu item.



Menu Item Properties

General | Extended Styles

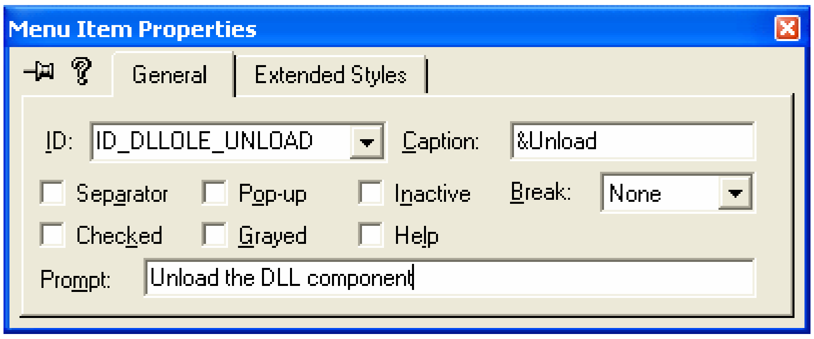
ID: ID_DLLOLE_GETDATA Caption: &Get Data

Separator [ ] Pop-up [ ] Inactive [ ] Break: None

Checked [ ] Grayed [ ] Help [ ]

Prompt: Get data from the DLL component

Figure 17: Adding **Get Data** menu item.



Menu Item Properties

General | Extended Styles

ID: ID_DLLOLE_UNLOAD Caption: &Unload

Separator [ ] Pop-up [ ] Inactive [ ] Break: None

Checked [ ] Grayed [ ] Help [ ]

Prompt: Unload the DLL component

Figure 18: Adding **Unload** menu item.

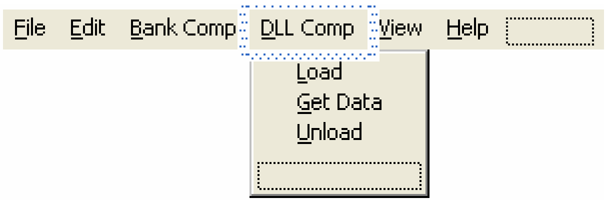
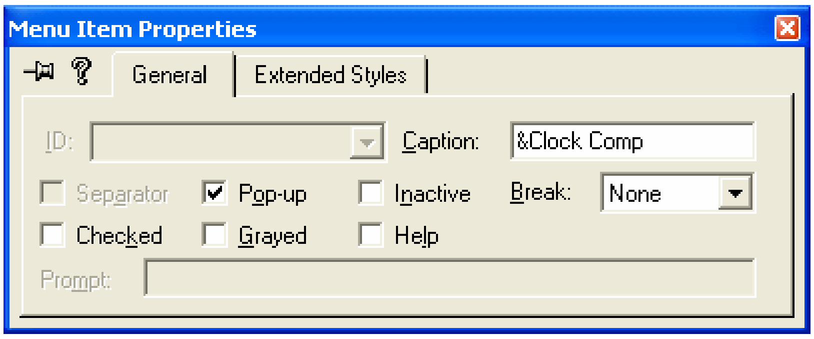


Figure 19: A completed DLL Comp menus.

Add menu items for **Clock Comp**.

ID	Menu	Prompt
-	Clock Comp	-
ID_CLOCKOLE_LOAD	Load	Load the clock component, MYMFC29C.EXE
ID_CLOCKOLE_CREATEALARM	Create Alarm	Create an alarm
ID_CLOCKOLE_REFRESHTIME	Refresh Time	Refresh the time from the system clock
ID_CLOCKOLE_UNLOAD	Unload	Unload the clock component

Table 3.



Menu Item Properties

General | Extended Styles

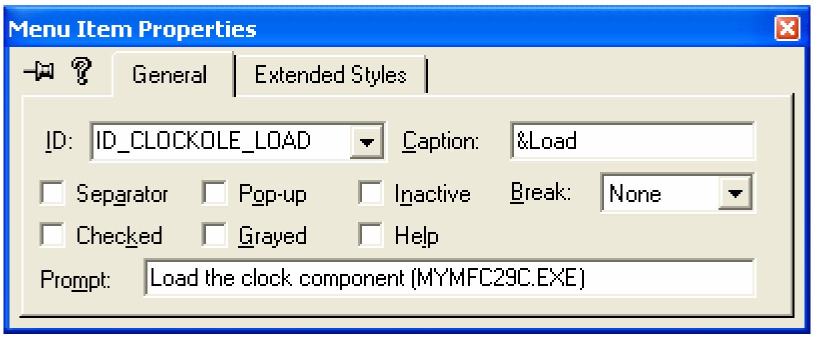
ID: Caption: &Clock Comp

Separator [ ] Pop-up [✓] Inactive [ ] Break: None

Checked [ ] Grayed [ ] Help [ ]

Prompt:

Figure 20: Adding **Clock Comp** menu.


Menu Item Properties

General | Extended Styles

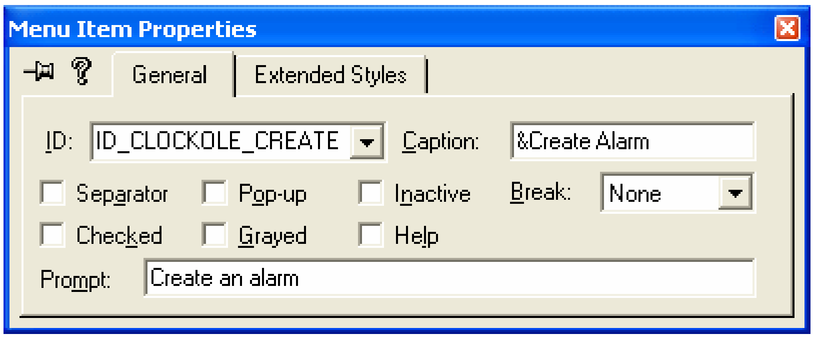
ID: ID_CLOCKOLE_LOAD Caption: &Load

☐ Separator ☐ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt: Load the clock component (MYMFC29C.EXE)

Figure 21: Adding **Load** menu item.


Menu Item Properties

General | Extended Styles

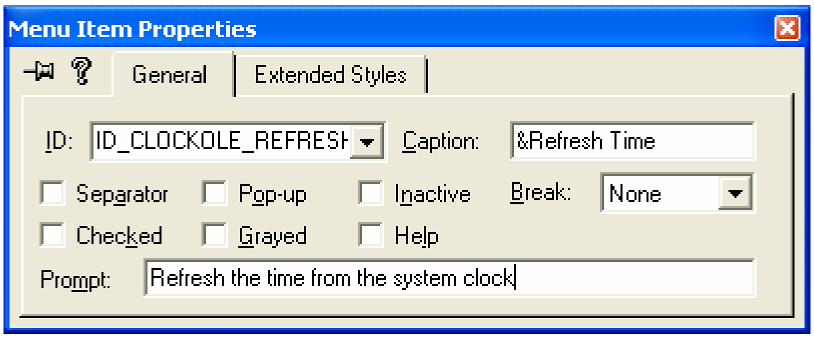
ID: ID_CLOCKOLE_CREATE Caption: &Create Alarm

☐ Separator ☐ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt: Create an alarm

Figure 22: Adding **CreateAlarm** menu item.


Menu Item Properties

General | Extended Styles

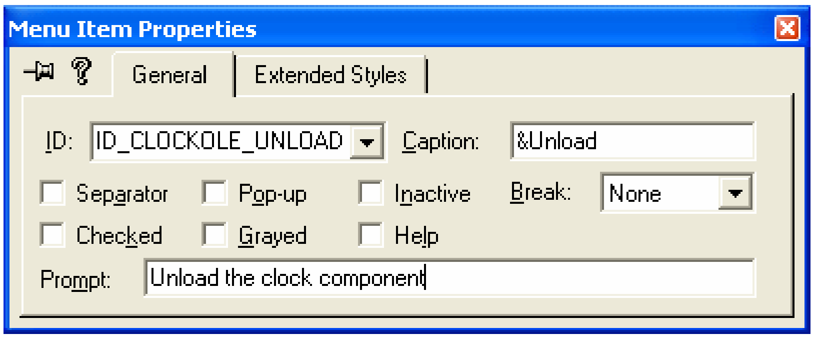
ID: ID_CLOCKOLE_REFRESH Caption: &Refresh Time

☐ Separator ☐ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt: Refresh the time from the system clock

Figure 23: Adding **Refresh Time** menu item.


Menu Item Properties

General | Extended Styles

ID: ID_CLOCKOLE_UNLOAD Caption: &Unload

☐ Separator ☐ Pop-up ☐ Inactive Break: None

☐ Checked ☐ Grayed ☐ Help

Prompt: Unload the clock component

Figure 24: Adding **Unload** menu item.

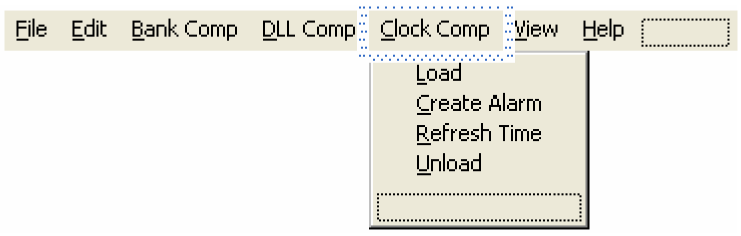
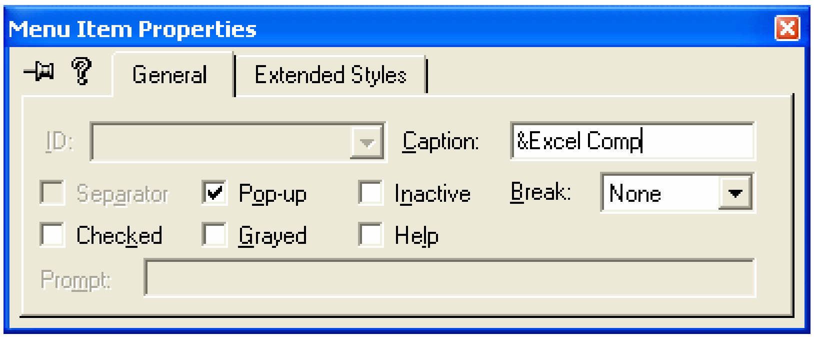


Figure 25: A completed **Clock Comp** menus.

Add menu and the item for **Excel Comp**.

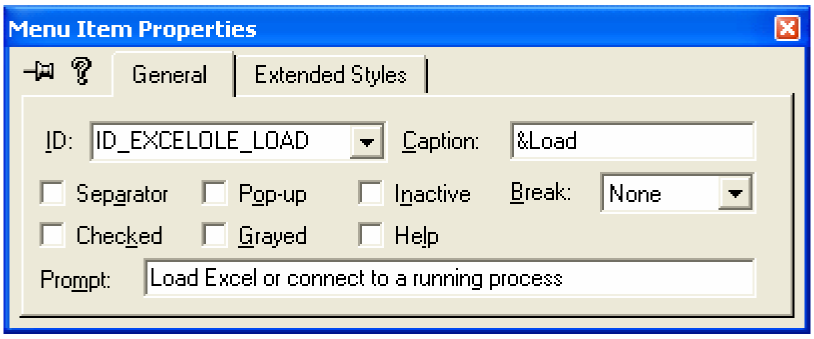
ID	Menu	Prompt
-	Excel Comp	-
ID_EXCELOLE_LOAD	Load	Load Excel or connect to a running process

Table 4.



ID:
Caption: &Excel Comp
☐ Separator ☑ Pop-up ☐ Inactive Break: None
☐ Checked ☐ Grayed ☐ Help
Prompt:

Figure 26: Adding **Excel Comp** menu.



ID: ID_EXCELOLE_LOAD
Caption: &Load
☐ Separator ☐ Pop-up ☐ Inactive Break: None
☐ Checked ☐ Grayed ☐ Help
Prompt: Load Excel or connect to a running process

Figure 27: Adding **Load** menu item.

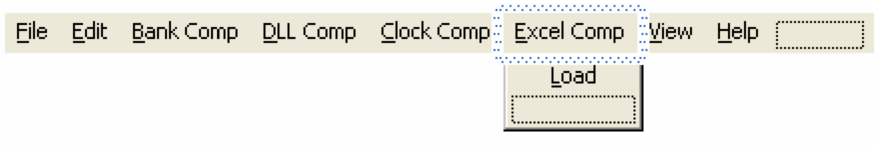


Figure 28: A completed **Excel Comp** menus.

Add a new dialog for the alarm setting. Use ResourceView to add new dialog, use the following information for the dialog and Edit controls. Leave the static texts IDs to the default.

ID	Item
IDD_ALARMDLG	Dialog with **Alarm Dialog** caption
IDC_HOURS	Hours edit box.
IDC_MINUTES	Minutes edit box.

`IDC_SECONDS`	Seconds edit box.

Table 6.

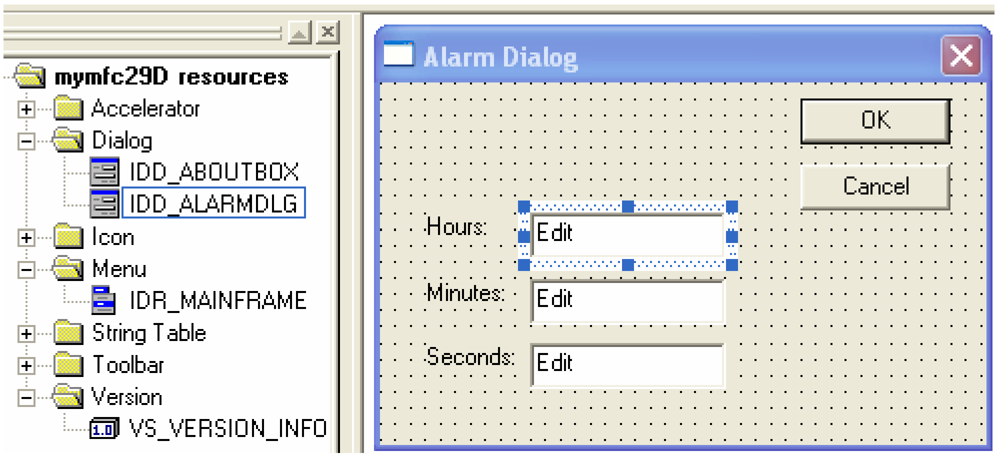
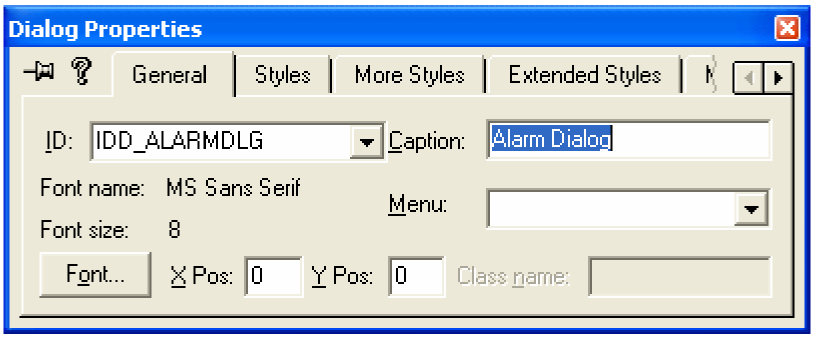
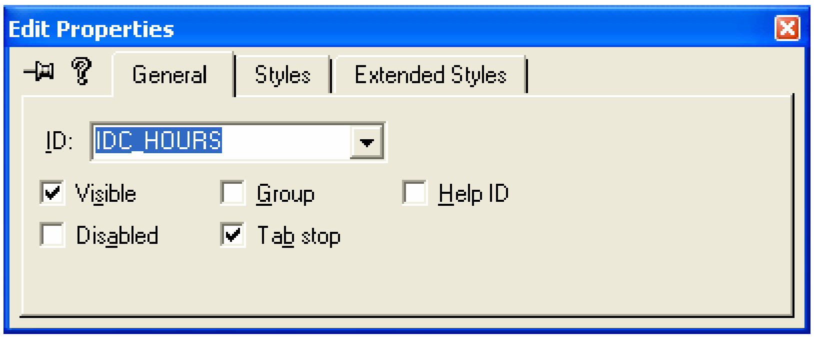


Figure 29: Adding dialog for alarm setting.



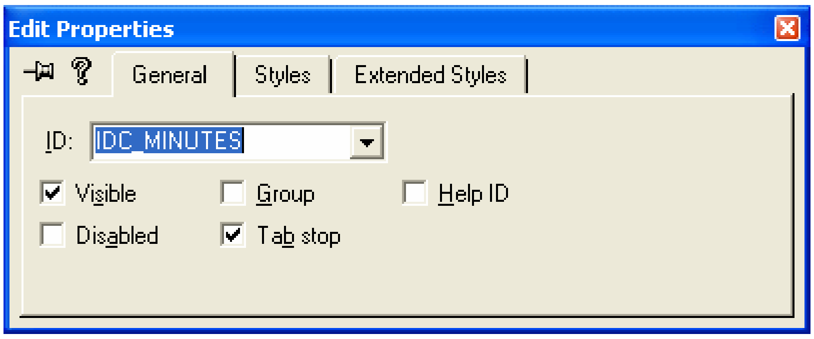
ID: IDD_ALARMDLG
Caption: Alarm Dialog
Font name: MS Sans Serif
Menu:
Font size: 8
X Pos: 0
Y Pos: 0
Class name:

Figure 30: Setting the dialog property.



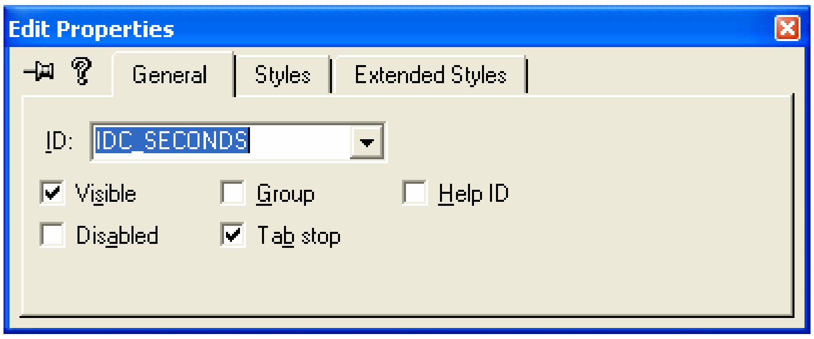
ID: IDC_HOURS
[✓] Visible [ ] Group [ ] Help ID
[ ] Disabled [✓] Tab stop

Figure 31: Setting the Hours Edit control property.



ID: IDC_MINUTES
[✓] Visible [ ] Group [ ] Help ID
[ ] Disabled [✓] Tab stop

Figure 32: Setting the Minutes Edit control property.


Edit Properties

General | Styles | Extended Styles

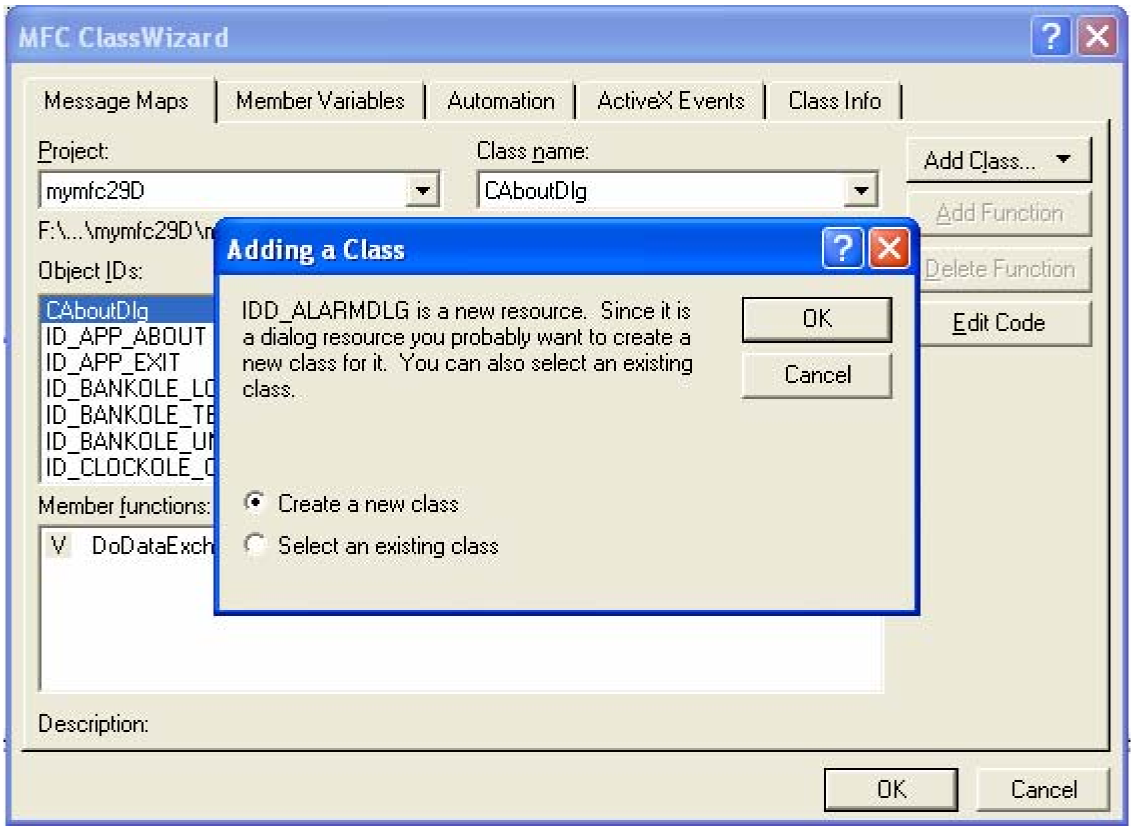
ID: IDC_SECONDS

[✓ Visible] Group Help ID

Disabled [✓ Tab stop]

Figure 33: Setting the Seconds Edit control property.

Launch ClassWizard. Click the **OK** button to create a new class.


MFC ClassWizard

Message Maps | Member Variables | Automation | ActiveX Events | Class Info

Project: mymfc29D

Class name: CAboutDlg

F:\...\mymfc29D\m

Object IDs: CAboutDlg, ID_APP_ABOUT, ID_APP_EXIT, ID_BANKOLE_LO, ID_BANKOLE_TE, ID_BANKOLE_UI, ID_CLOCKOLE_C

Member functions: V DoDataExch

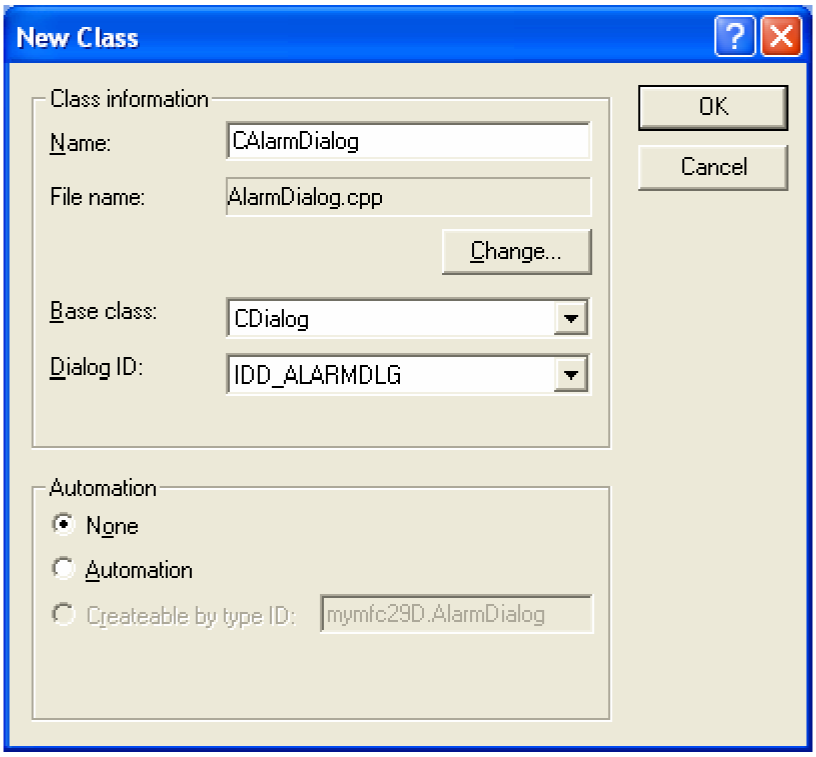
Description:

Adding a Class

IDD_ALARMDLG is a new resource. Since it is a dialog resource you probably want to create a new class for it. You can also select an existing class.

[● Create a new class] Select an existing class

Figure 34: Adding new class, `CAlarmDialog` for the previously created dialog.



Class information

Name: CAlarmDialog

File name: AlarmDialog.cpp

Base class: CDialog

Dialog ID: IDD_ALARMDLG

Automation

[✓ None]

Automation

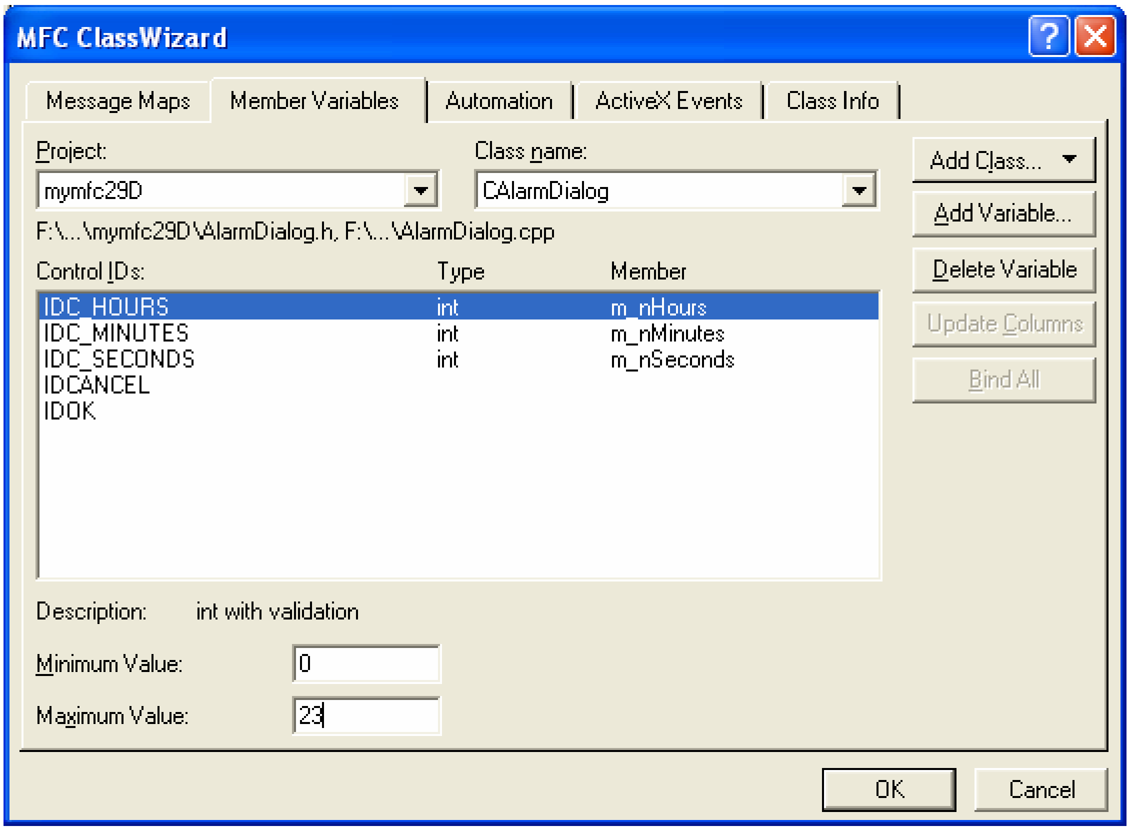
Createable by type ID: mymfc29D.AlarmDialog

Figure 35: Entering `CAlarmDialog` class information, without Automation support.

Add member variables and initialize them. Add the following member variables using **Member Variables** page of ClassWizard.

ID	Type	Member name	Min/Max value
`IDC_HOURS`	`int`	`m_nHours`	0/23
`IDC_MINUTES`	`int`	`m_nMinutes`	0/59
`IDC_SECONDS`	`int`	`m_nSeconds`	0/59

Table 7.



Project: mymfc29D

Class name: CAlarmDialog

F:\...\mymfc29D\AlarmDialog.h, F:\...\AlarmDialog.cpp

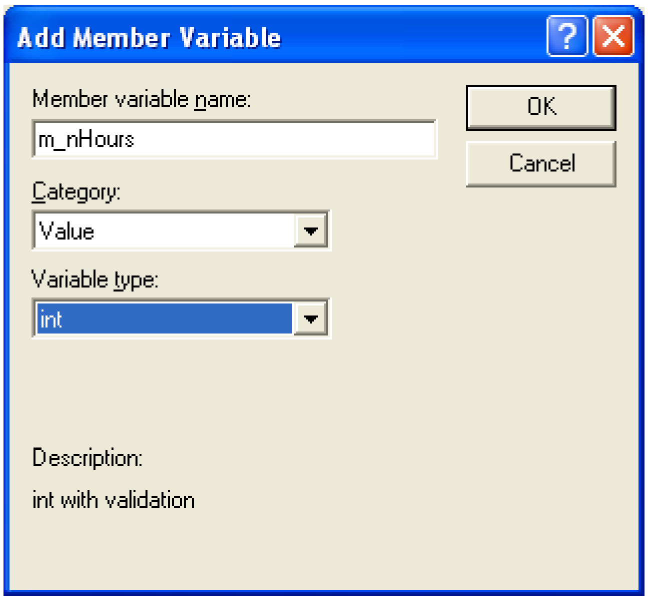
Control IDs:	Type	Member
IDC_HOURS	int	m_nHours
IDC_MINUTES	int	m_nMinutes
IDC_SECONDS	int	m_nSeconds
IDCANCEL		
IDOK		

Description: int with validation

Minimum Value: 0

Maximum Value: 23

Figure 36: Adding member variables to the controls in `CAlarmDialog` class.



Member variable name: m_nHours

Category: Value

Variable type: int

Description:

int with validation

Figure 37: Adding `m_nHours` member variable to `IDC_HOURS` Edit control.

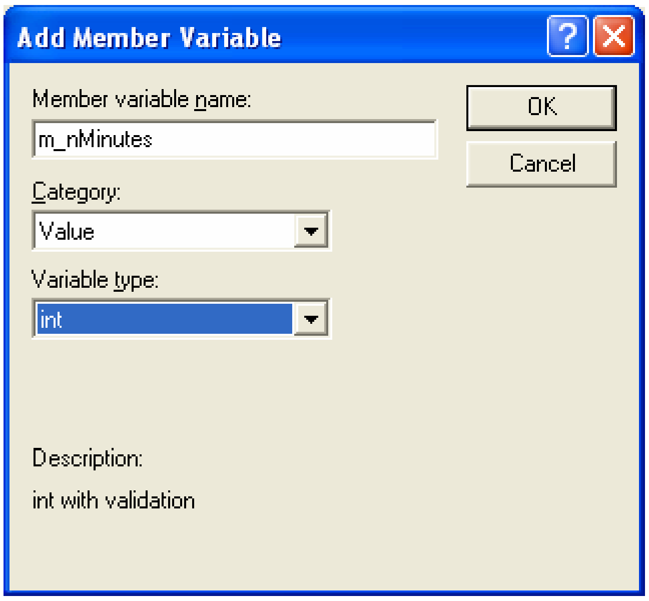


Figure 38: Adding `m_nMinutes` member variable to `IDC_MINUTES` Edit control. Repeat the same step for `IDC_SECONDS`.

Using ClassWizard to generate a driver class from the component project's **TLB** file. Open the ClassWizard and click the **Add Class** button and select the **From a type library** option. Here, we extract and add to project, an already created class from TLB (type library) file of the MYMFC29A project. It is class reusability.

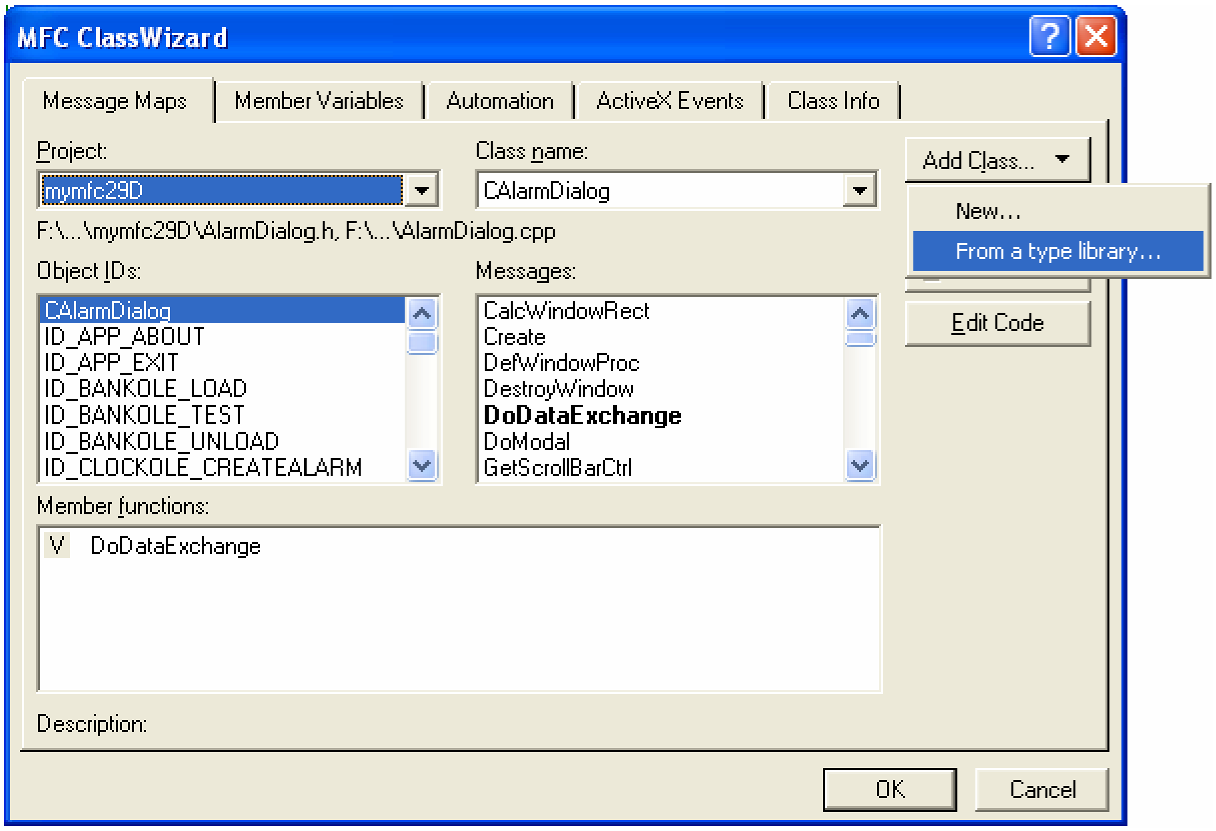
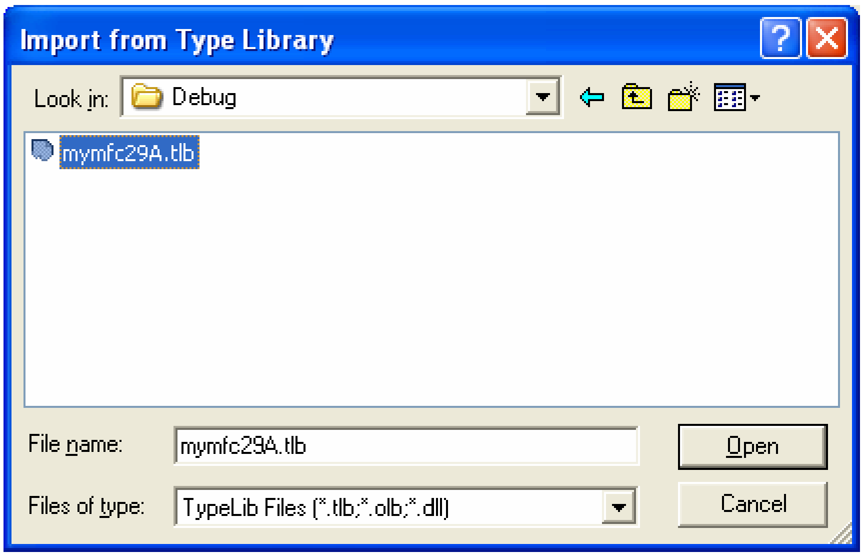


Figure 39: Adding class from TLB file.

Find your **mymfc29A.tlb** file in the Debug directory of the MYMFC29A project directory. Select the file and click the **Open** button.

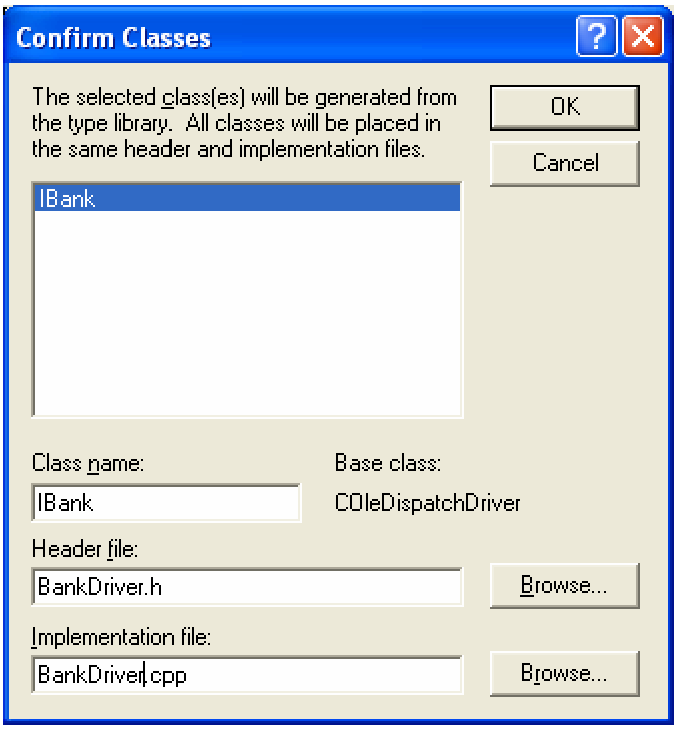


File name: mymfc29A.tlb

Files of type: TypeLib Files (*.tlb;*.olb;*.dll)

Figure 40: Browsing and selecting MYMFC29A's TLB file.

Change the header and implementation files to **BankDriver.h** and **BankDriver.cpp** respectively and click the **OK** button.



The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.

IBank

Class name: IBank

Base class: COleDispatchDriver

Header file: BankDriver.h

Implementation file: BankDriver.cpp

Figure 41: Adding `IBank` component from MYMFC29A project. At the same time, changing the class header and source file names.

Select the `IBank` component, the added files and class can be seen through the ClassView and FileView respectively. Repeat the same steps for TLB files of the MYMFC29B and MYMFC29C projects.

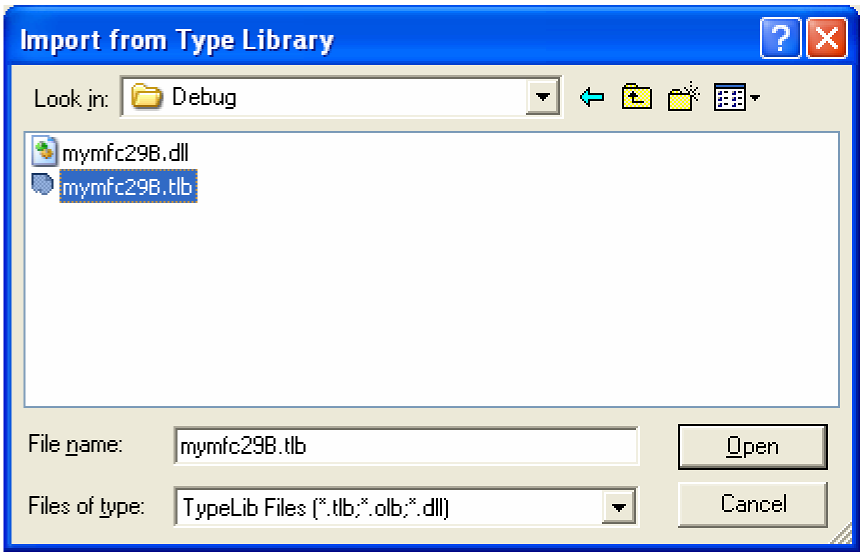


Figure 42: Selecting MYMFC29B's TLB file.

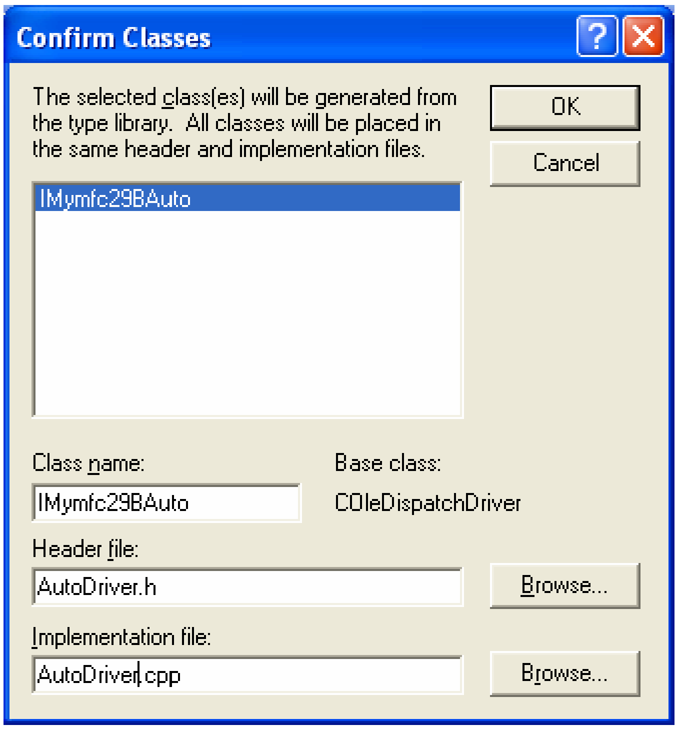
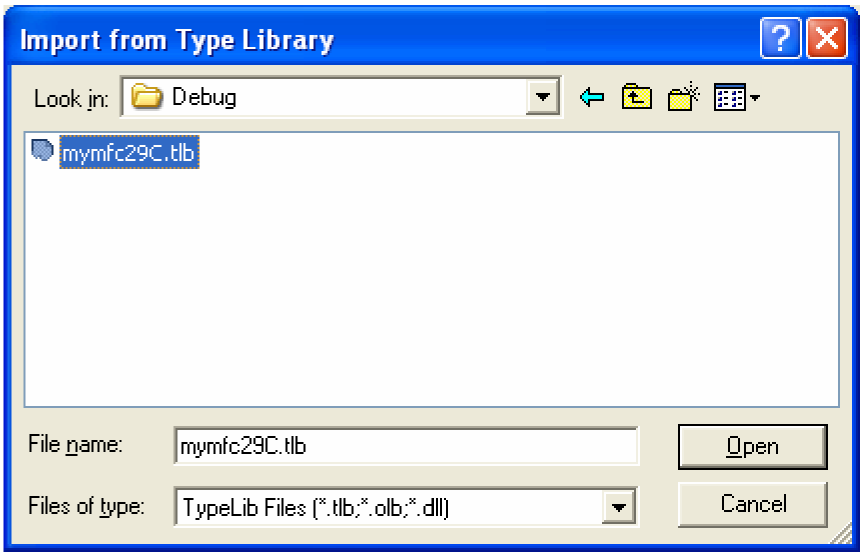

Confirm Classes
The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.
OK
Cancel
IMymfc29BAuto
Class name: IMymfc29BAuto
Base class: COleDispatchDriver
Header file: AutoDriver.h
Implementation file: AutoDriver.cpp
Browse...
Browse...

Figure 43: Adding `IMymfc29BAuto` component to MYMFC29D project.



Import from Type Library

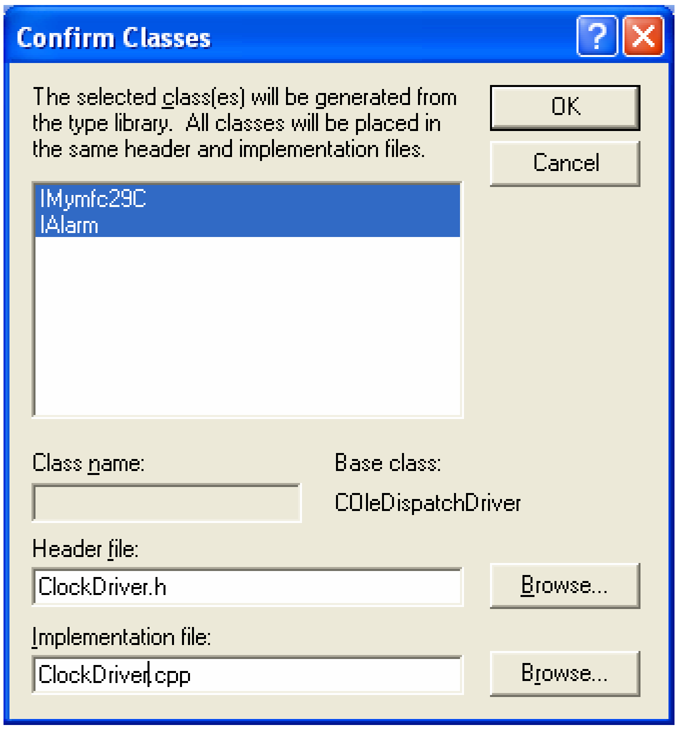
Look in: Debug

mymfc29C.tlb

File name: mymfc29C.tlb

Files of type: TypeLib Files (*.tlb;*.olb;*.dll)

Figure 44: Selecting MYMFC29C's TLB file.



Confirm Classes

The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.

IMymfc29C
IAlarm

Class name:

Base class: COleDispatchDriver

Header file: ClockDriver.h

Implementation file: ClockDriver.cpp

Figure 45: Adding `IMymfc29C` and `IAlarm` components to MYMFC29D project.

The Coding Part

Add the automation support manually. Add the following code in **StdAfx.h**.

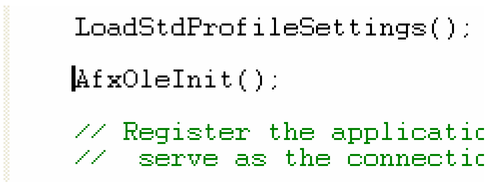
```
#include <afxdisp.h>
```

```
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxdisp.h> // Automation support
//{{AFX_INSERT_LOCATION}}
```

Listing 1.

Then add this call at the beginning of the application's `InitInstance()` function in **mymfc29D.cpp**.

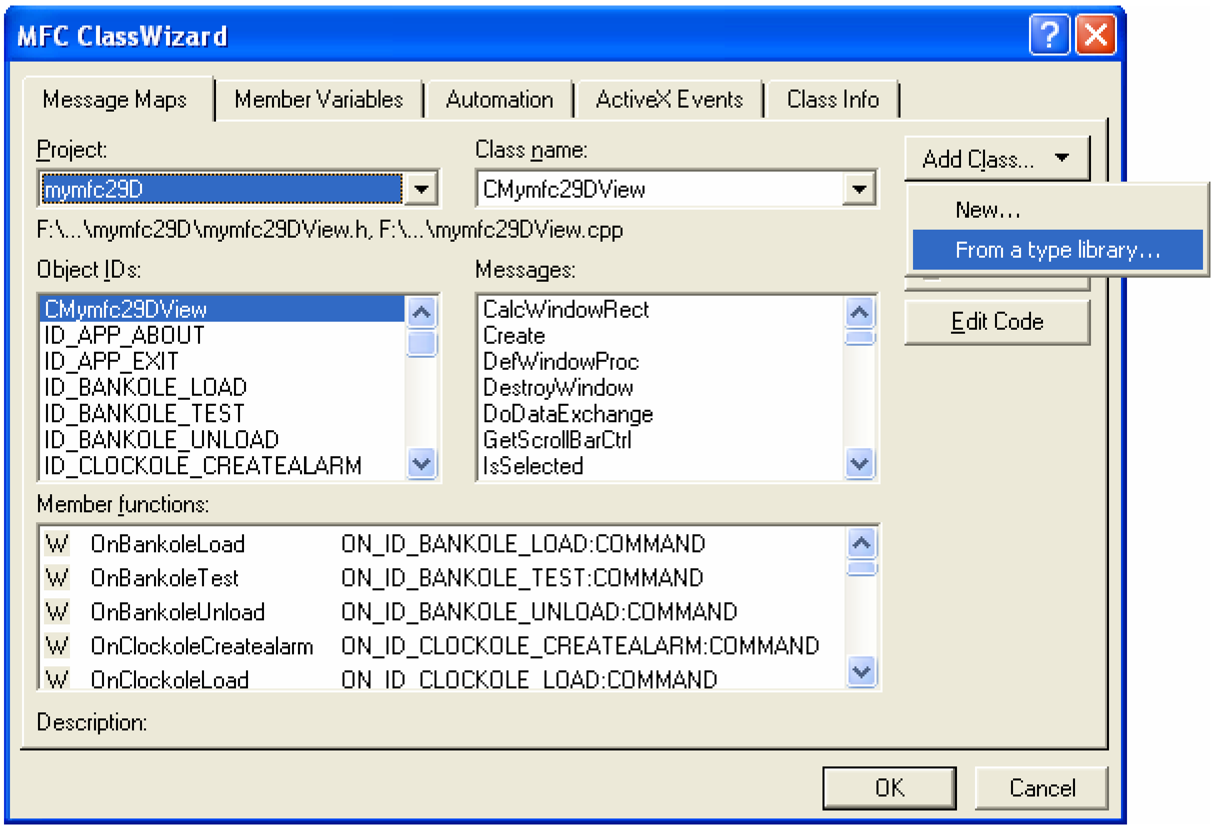
```
        AfxOleInit();
```



```
    LoadStdProfileSettings();
    AfxOleInit();

    // Register the applicatic
    //  serve as the connectic
```

Listing 2.

Add classes from Excel type library. Again, add classes from Excel's type library.



Project: mymfc29D
Class name: CMymfc29DView
F:\...\mymfc29D\mymfc29DView.h, F:\...\mymfc29DView.cpp

Object IDs:
CMymfc29DView
ID_APP_ABOUT
ID_APP_EXIT
ID_BANKOLE_LOAD
ID_BANKOLE_TEST
ID_BANKOLE_UNLOAD
ID_CLOCKOLE_CREATEALARM

Messages:
CalcWindowRect
Create
DefWindowProc
DestroyWindow
DoDataExchange
GetScrollBarCtrl
IsSelected

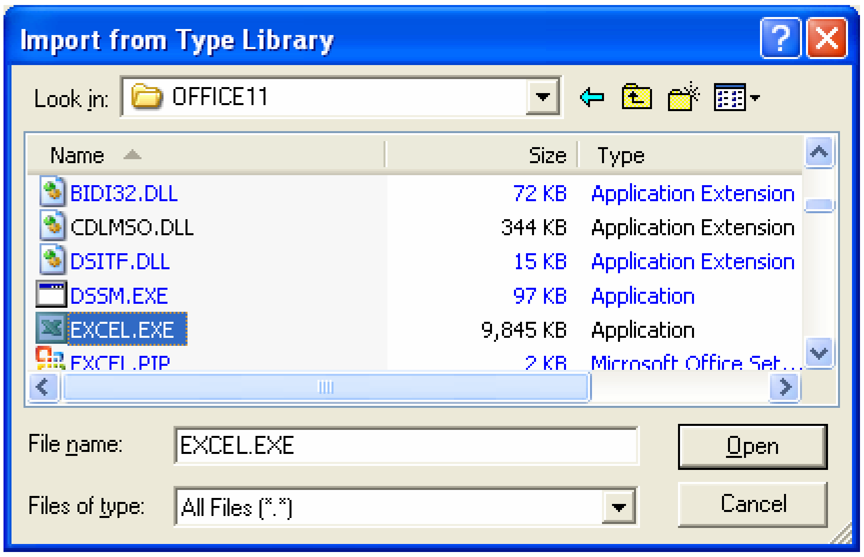
Member functions:

```
W  OnBankoleLoad          ON_ID_BANKOLE_LOAD:COMMAND
W  OnBankoleTest          ON_ID_BANKOLE_TEST:COMMAND
W  OnBankoleUnload        ON_ID_BANKOLE_UNLOAD:COMMAND
W  OnClockoleCreatealarm  ON_ID_CLOCKOLE_CREATEALARM:COMMAND
W  OnClockoleLoad         ON_ID_CLOCKOLE_LOAD:COMMAND
```

Description:

Figure 46: Adding new classes from type library.

For Tenouk's Office 2003 English version (Office 11) the OLB/TLB file is **excel.exe**. From the previous steps, we can see that the type library file extensions may be .TLB, .OLB, .DLL or .EXE.



Name	Size	Type
BIDI32.DLL	72 KB	Application Extension
CDLMSO.DLL	344 KB	Application Extension
DSITF.DLL	15 KB	Application Extension
DSSM.EXE	97 KB	Application
EXCEL.EXE	9,845 KB	Application
EXCEL.PIP	2 KB	Microsoft Office Set...

File name: EXCEL.EXE

Figure 47: Selecting the Excel type library file. Take note that it is Excel executable file.

Some Notes

Once you have added the classes from a type library to your project, you will notice that many classes have been added to the project. In ClassView, you can double-click a class to see the member functions of that class and then double-click the member function to view the definition of that function in the Excel.cpp implementation file.
You need to go to the definition of a member function if you wish to verify a return type or if you need to change a function's implementation. Any time you change a function definition, remember to change the declaration in the **Excel8.h** file. When doing so, be sure that you change the correct function declaration; sometimes, the same name is given to member functions of multiple classes, `GetApplication()` is one such example.
Although the steps above illustrate how to automate Microsoft Excel, you can apply the same ideas to automating other applications. The list below contains the file names for the type libraries of the Microsoft Office applications:

Application	Type Library
Microsoft Access 97	**Msacc8.olb**
Microsoft Jet Database 3.5	**DAO350.dll**
Microsoft Binder 97	**Msbdr8.olb**
Microsoft Excel 97	**Excel8.olb**
Microsoft Graph 97	**Graph8.olb**
Microsoft Office 97	**Mso97.dll**
Microsoft Outlook 97	**Msoutl97.olb**
Microsoft PowerPoint 97	**Msppt8.olb**
Microsoft Word 97	**Msword8.olb**
Microsoft Access 2000	**Msacc9.olb**
Microsoft Jet Database 3.51	**DAO360.dll**
Microsoft Binder 2000	**Msbdr9.olb**
Microsoft Excel 2000	**Excel9.olb**
Microsoft Graph 2000	**Graph9.olb**
Microsoft Office 2000	**Mso9.dll**
Microsoft Outlook 2000	**Msoutl9.olb**
Microsoft PowerPoint 2000	**Msppt9.olb**
Microsoft Word 2000	**Msword9.olb**
Microsoft Access 2002	**Msacc.olb**
Microsoft Excel 2002	**Excel.exe**
Microsoft Graph 2002	**Graph.exe**
Microsoft Office 2002	**MSO.dll**

Microsoft Outlook 2002	**MSOutl.olb**
Microsoft PowerPoint 2002	**MSPpt.olb**
Microsoft Word 2002	**MSWord.olb**
Microsoft Office Access 2003	**Msacc.olb**
Microsoft Office Excel 2003	**Excel.exe**
Microsoft Graph 2003	**Graph.exe**
Microsoft Office 2003	**MSO.dll**
Microsoft Office Outlook 2003	**MSOutl.olb**
Microsoft Office PowerPoint 2003	**MSPpt.olb**
Microsoft Office Word 2003	**MSWord.olb**

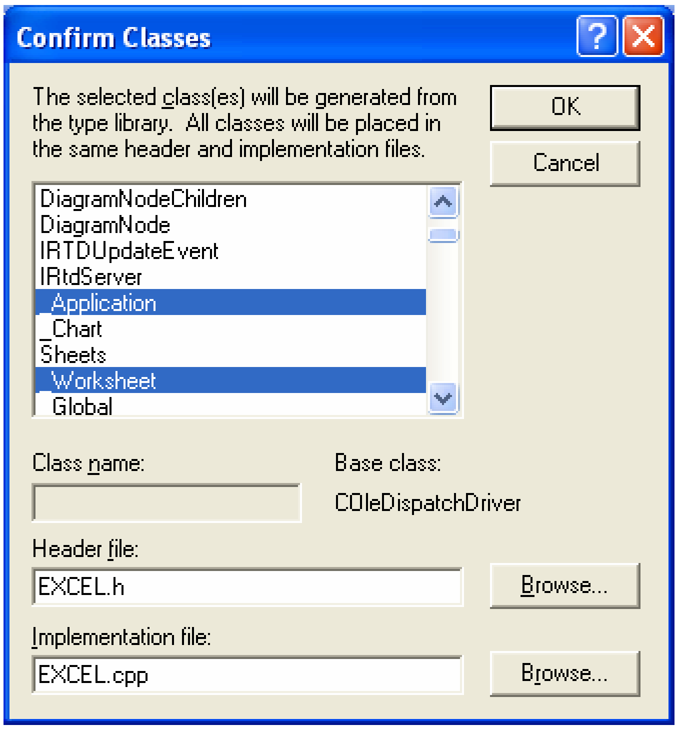
Table 8.

The default location for these type libraries is **C:\Program Files\Microsoft Office\Office** for Office 2002 the path is **C:\...\Office10** and for Office 2003 the path is **C:\...\Office11**, except for **Dao350.dll** or **Dao360.dll**, and Microsoft Office 10(**MSO.dll**). The default location for **Dao350.dll/Dao360.dll** is **C:\Program Files\Common Files\Microsoft Shared\Dao**. The default location for **MSO.dll** is **C:\Program Files\Common Files\Microsoft Shared\Office10** for Office 2002 and **C:\Program Files\Common Files\Microsoft Shared\Office11** for Office 2003.

Select the following classes by holding down the **Ctrl** button while selecting the classes with your mouse click. It seems that different versions having slightly different class names and similar class name may have different functionalities for different versions.

Class
_Application
_Workbook
Workbooks
_Worksheet
Worksheets
Range

Table 9.



The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.

DiagramNodeChildren
DiagramNode
IRTDUpdateEvent
IRtdServer
_Application
_Chart
Sheets
_Worksheet
Global

Class name:

Base class: COleDispatchDriver

Header file: EXCEL.h

Implementation file: EXCEL.cpp

Figure 48: Selecting classes from Excel 2003 type library.

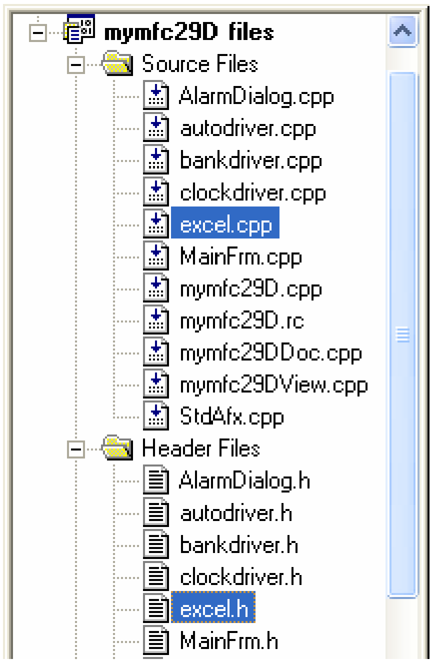


Figure 49: The added header and source files seen through FileView.

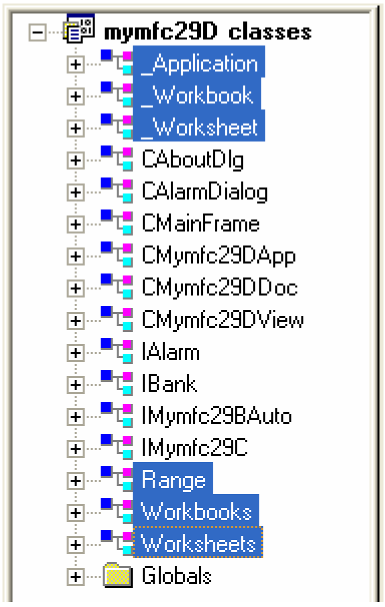


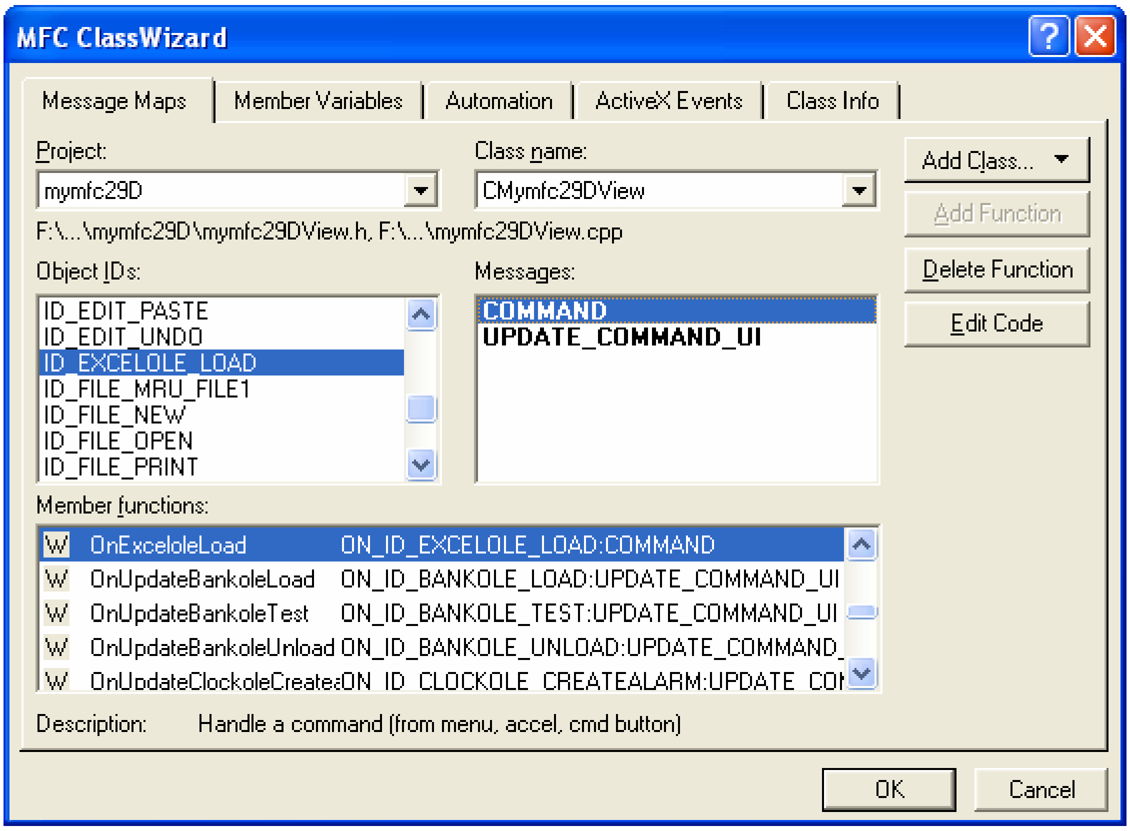
Figure 50: The added classes seen through ClassView.

Next, Using ClassWizard, add the menu **commands** and **update command UI** events as listed in the following Table to the `CMymfc29DView` class for all the menu items created previously.

ID
ID_BANKOLE_LOAD
ID_BANKOLE_TEST
ID_BANKOLE_UNLOAD
ID_DLLOLE_LOAD
ID_DLLOLE_GETDATA
ID_DLLOLE_UNLOAD
ID_CLOCKOLE_LOAD

ID_CLOCKOLE_CREATEALARM
ID_CLOCKOLE_REFRESHTIME
ID_CLOCKOLE_UNLOAD
ID_EXCELOLE_LOAD

Table 10.



Project: mymfc29D

Class name: CMymfc29DView

F:\...\mymfc29D\mymfc29DView.h, F:\...\mymfc29DView.cpp

Object IDs:

```
ID_EDIT_PASTE
ID_EDIT_UNDO
ID_EXCELOLE_LOAD
ID_FILE_MRU_FILE1
ID_FILE_NEW
ID_FILE_OPEN
ID_FILE_PRINT
```

Messages:

```
COMMAND
UPDATE_COMMAND_UI
```

Member functions:

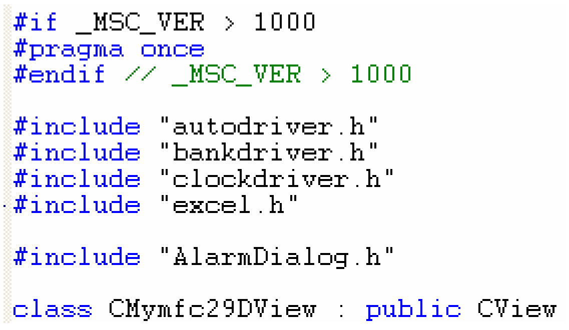
```
W  OnExceloleLoad          ON_ID_EXCELOLE_LOAD:COMMAND
W  OnUpdateBankoleLoad     ON_ID_BANKOLE_LOAD:UPDATE_COMMAND_UI
W  OnUpdateBankoleTest     ON_ID_BANKOLE_TEST:UPDATE_COMMAND_UI
W  OnUpdateBankoleUnload   ON_ID_BANKOLE_UNLOAD:UPDATE_COMMAND_
W  OnUpdateClockoleCreate  ON_ID_CLOCKOLE_CREATEALARM:UPDATE_CO
```

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

Figure 51: Adding Commands and Update Commands to `CMymfc29DView` class.

Add the following `#include` statements in **mymfc29DView.h**.

```
#include "autodriver.h"
#include "bankdriver.h"
#include "clockdriver.h"
#include "excel.h"

#include "AlarmDialog.h"
```

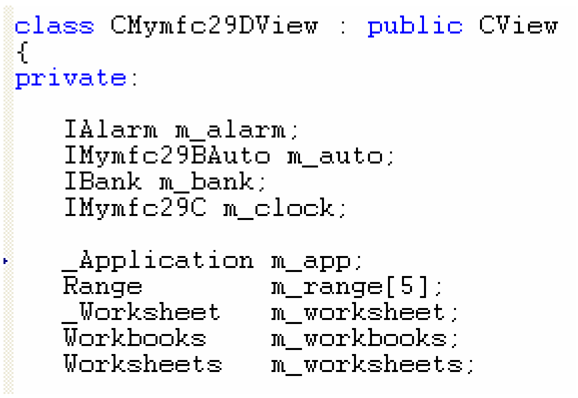


```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "autodriver.h"
#include "bankdriver.h"
#include "clockdriver.h"
#include "excel.h"

#include "AlarmDialog.h"

class CMymfc29DView : public CView
```

Listing 3.

Then, add the following member variables also in the **mymfc29Dview.h**.

```
private:

   IAlarm m_alarm;
   IMymfc29BAuto m_auto;
   IBank m_bank;
   IMymfc29C m_clock;

   _Application m_app;
   Range        m_range[5];
   _Worksheet   m_worksheet;
   Workbooks    m_workbooks;
   Worksheets   m_worksheets;
```

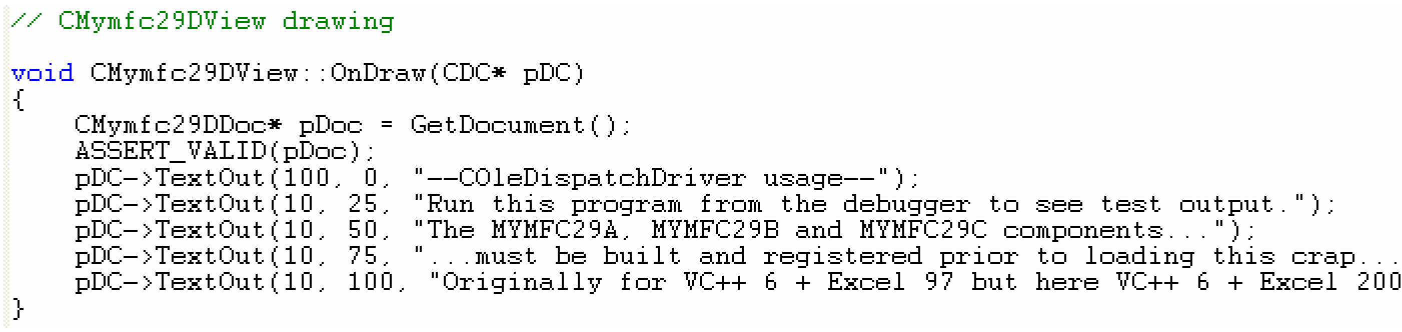


```
class CMymfc29DView : public CView
{
private:

    IAlarm m_alarm;
    IMymfc29BAuto m_auto;
    IBank m_bank;
    IMymfc29C m_clock;

    _Application m_app;
    Range        m_range[5];
    _Worksheet   m_worksheet;
    Workbooks    m_workbooks;
    Worksheets   m_worksheets;
```

Listing 4.

Add/edit code for `CMymfc29DView::OnDraw()` member function in **mymfc29DView.cpp**.

```
void CMymfc29DView::OnDraw(CDC* pDC)
{
    CMymfc29DDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(100, 0, "--COleDispatchDriver usage--");
    pDC->TextOut(10, 25, "Run this program from the debugger to see test
output.");
    pDC->TextOut(10, 50, "The MYMFC29A, MYMFC29B and MYMFC29C
components...");
    pDC->TextOut(10, 75, "...must be built and registered prior to loading
this crap...");
    pDC->TextOut(10, 100, "Originally for VC++ 6 + Excel 97 but here VC++ 6 +
Excel 2003...");
}
```

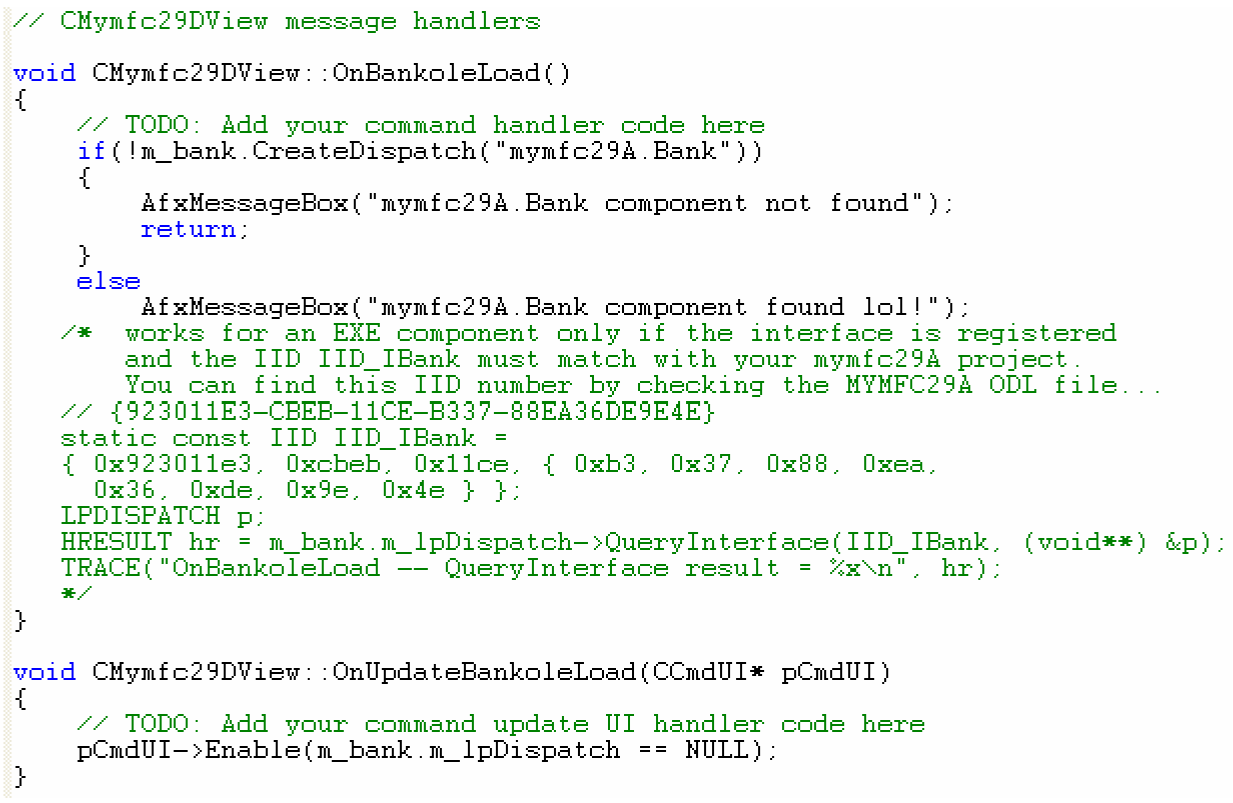


```
// CMymfc29DView drawing

void CMymfc29DView::OnDraw(CDC* pDC)
{
    CMymfc29DDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(100, 0, "--COleDispatchDriver usage--");
    pDC->TextOut(10, 25, "Run this program from the debugger to see test output.");
    pDC->TextOut(10, 50, "The MYMFC29A, MYMFC29B and MYMFC29C components...");
    pDC->TextOut(10, 75, "...must be built and registered prior to loading this crap...
    pDC->TextOut(10, 100, "Originally for VC++ 6 + Excel 97 but here VC++ 6 + Excel 200
}
```

Listing 5.

Finally, add codes for the **commands** and **update command UI** events of the `CMymfc29DView` class.

```
void CMymfc29DView::OnBankoleLoad()
{
	// TODO: Add your command handler code here
	if(!m_bank.CreateDispatch("mymfc29A.Bank"))
	{
		AfxMessageBox("mymfc29A.Bank component not found");
		return;
	}
	else
		AfxMessageBox("mymfc29A.Bank component found lol!");
   /*  works for an EXE component only if the interface is registered
       and the IID IID_IBank must match with your mymfc29A project.
       You can find this IID number by checking the MYMFC29A ODL file...
   // {923011E3-CBEB-11CE-B337-88EA36DE9E4E}
   static const IID IID_IBank =
   { 0x923011e3, 0xcbeb, 0x11ce, { 0xb3, 0x37, 0x88, 0xea,
     0x36, 0xde, 0x9e, 0x4e } };
   LPDISPATCH p;
   HRESULT hr = m_bank.m_lpDispatch->QueryInterface(IID_IBank, (void**) &p);
   TRACE("OnBankoleLoad -- QueryInterface result = %x\n", hr);
   */
}

void CMymfc29DView::OnUpdateBankoleLoad(CCmdUI* pCmdUI)
{
	// TODO: Add your command update UI handler code here
	pCmdUI->Enable(m_bank.m_lpDispatch == NULL);
}
```

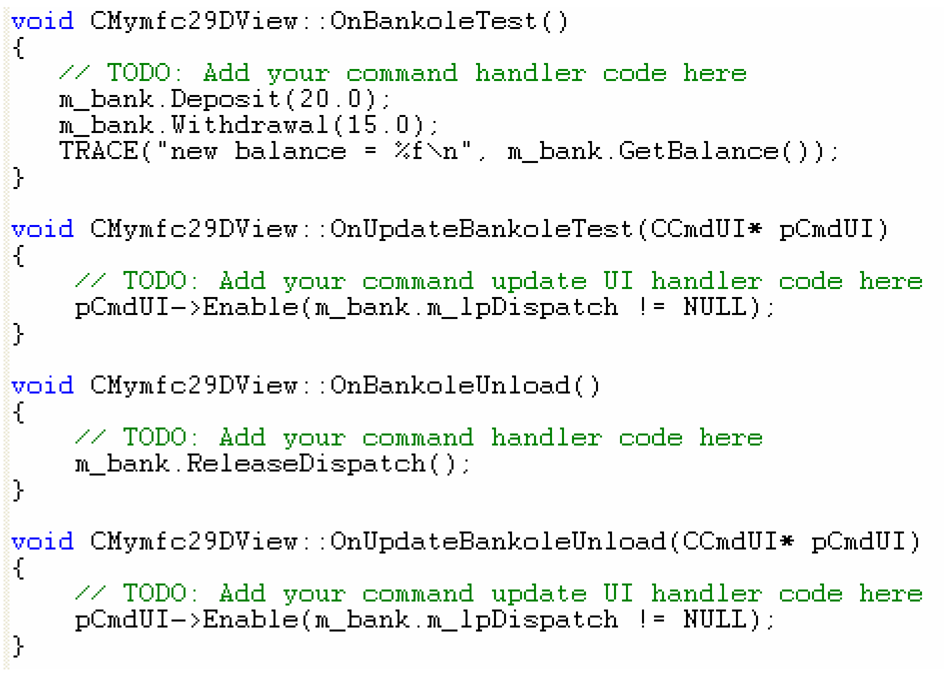


```
// CMymfc29DView message handlers

void CMymfc29DView::OnBankoleLoad()
{
    // TODO: Add your command handler code here
    if(!m_bank.CreateDispatch("mymfc29A.Bank"))
    {
        AfxMessageBox("mymfc29A.Bank component not found");
        return;
    }
    else
        AfxMessageBox("mymfc29A.Bank component found lol!");
   /*  works for an EXE component only if the interface is registered
       and the IID IID_IBank must match with your mymfc29A project.
       You can find this IID number by checking the MYMFC29A ODL file...
   // {923011E3-CBEB-11CE-B337-88EA36DE9E4E}
   static const IID IID_IBank =
   { 0x923011e3, 0xcbeb, 0x11ce, { 0xb3, 0x37, 0x88, 0xea,
     0x36, 0xde, 0x9e, 0x4e } };
   LPDISPATCH p;
   HRESULT hr = m_bank.m_lpDispatch->QueryInterface(IID_IBank, (void**) &p);
   TRACE("OnBankoleLoad -- QueryInterface result = %x\n", hr);
   */
}

void CMymfc29DView::OnUpdateBankoleLoad(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_bank.m_lpDispatch == NULL);
}
```

Listing 6.

```
void CMymfc29DView::OnBankoleTest()
```

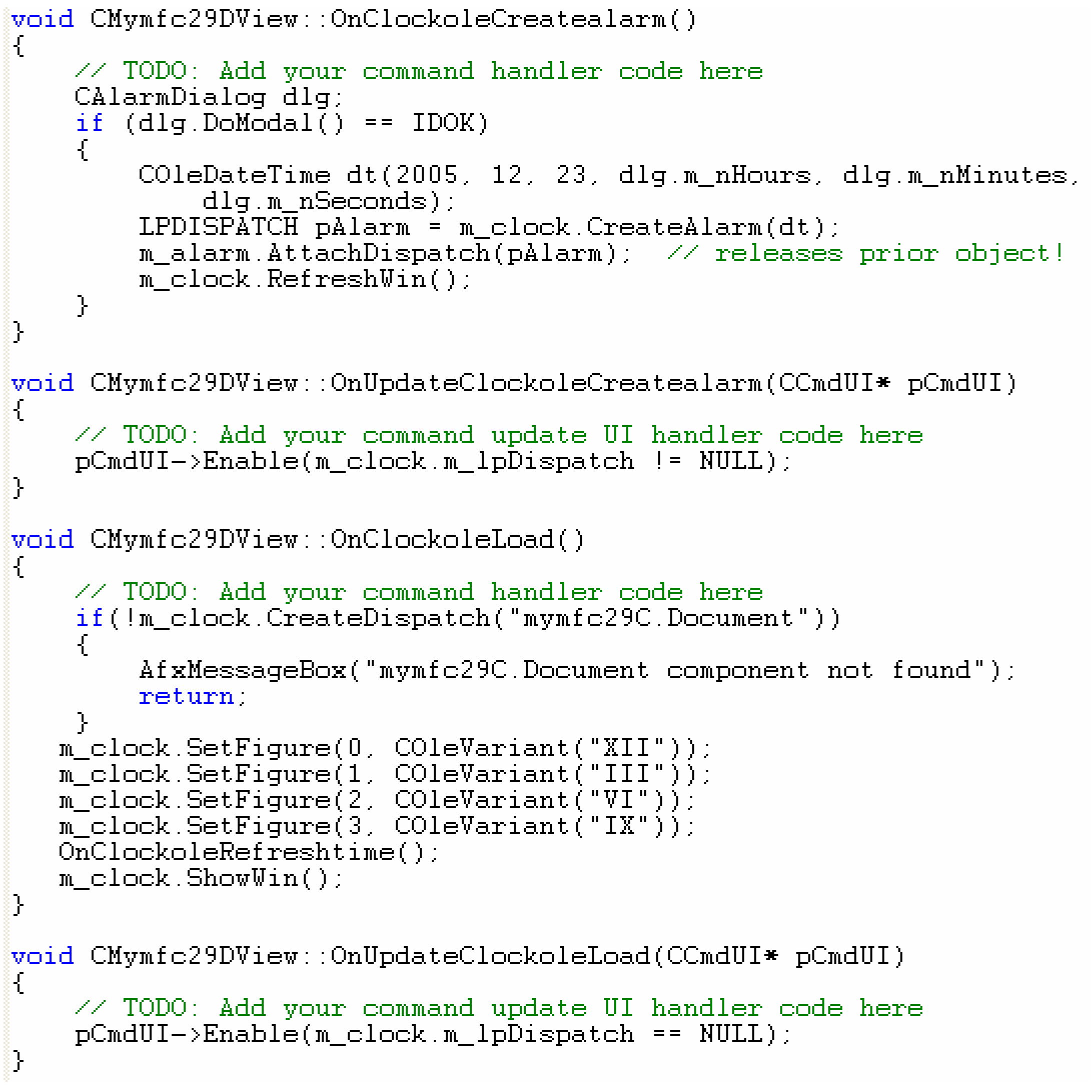
```
{
   // TODO: Add your command handler code here
   m_bank.Deposit(20.0);
   m_bank.Withdrawal(15.0);
   TRACE("new balance = %f\n", m_bank.GetBalance());
}

void CMymfc29DView::OnUpdateBankoleTest(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_bank.m_lpDispatch != NULL);
}

void CMymfc29DView::OnBankoleUnload()
{
      // TODO: Add your command handler code here
      m_bank.ReleaseDispatch();
}

void CMymfc29DView::OnUpdateBankoleUnload(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_bank.m_lpDispatch != NULL);
}
```



```
void CMymfc29DView::OnBankoleTest()
{
   // TODO: Add your command handler code here
   m_bank.Deposit(20.0);
   m_bank.Withdrawal(15.0);
   TRACE("new balance = %f\n", m_bank.GetBalance());
}

void CMymfc29DView::OnUpdateBankoleTest(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_bank.m_lpDispatch != NULL);
}

void CMymfc29DView::OnBankoleUnload()
{
    // TODO: Add your command handler code here
    m_bank.ReleaseDispatch();
}

void CMymfc29DView::OnUpdateBankoleUnload(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_bank.m_lpDispatch != NULL);
}
```

Listing 7.

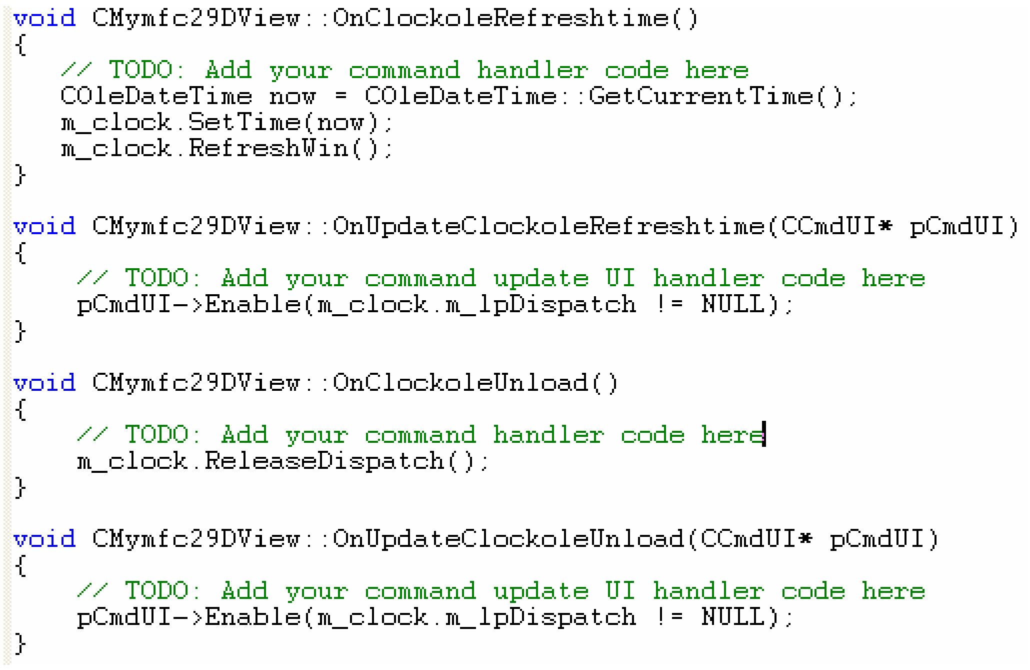
```
void CMymfc29DView::OnClockoleCreatealarm()
{
      // TODO: Add your command handler code here
      CAlarmDialog dlg;
      if (dlg.DoModal() == IDOK)
      {
            COleDateTime dt(2005, 12, 23, dlg.m_nHours, dlg.m_nMinutes,
                  dlg.m_nSeconds);
            LPDISPATCH pAlarm = m_clock.CreateAlarm(dt);
            m_alarm.AttachDispatch(pAlarm);  // releases prior object!
            m_clock.RefreshWin();
      }
}

void CMymfc29DView::OnUpdateClockoleCreatealarm(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}

void CMymfc29DView::OnClockoleLoad()
{
      // TODO: Add your command handler code here
      if(!m_clock.CreateDispatch("mymfc29C.Document"))
      {
            AfxMessageBox("mymfc29C.Document component not found");
            return;
       }
   m_clock.SetFigure(0, COleVariant("XII"));
   m_clock.SetFigure(1, COleVariant("III"));
   m_clock.SetFigure(2, COleVariant("VI"));
   m_clock.SetFigure(3, COleVariant("IX"));
   OnClockoleRefreshtime();
   m_clock.ShowWin();
}

void CMymfc29DView::OnUpdateClockoleLoad(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_clock.m_lpDispatch == NULL);
}
```



```cpp
void CMymfc29DView::OnClockoleCreatealarm()
{
    // TODO: Add your command handler code here
    CAlarmDialog dlg;
    if (dlg.DoModal() == IDOK)
    {
        COleDateTime dt(2005, 12, 23, dlg.m_nHours, dlg.m_nMinutes,
            dlg.m_nSeconds);
        LPDISPATCH pAlarm = m_clock.CreateAlarm(dt);
        m_alarm.AttachDispatch(pAlarm);  // releases prior object!
        m_clock.RefreshWin();
    }
}

void CMymfc29DView::OnUpdateClockoleCreatealarm(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}

void CMymfc29DView::OnClockoleLoad()
{
    // TODO: Add your command handler code here
    if(!m_clock.CreateDispatch("mymfc29C.Document"))
    {
        AfxMessageBox("mymfc29C.Document component not found");
        return;
    }
   m_clock.SetFigure(0, COleVariant("XII"));
   m_clock.SetFigure(1, COleVariant("III"));
   m_clock.SetFigure(2, COleVariant("VI"));
   m_clock.SetFigure(3, COleVariant("IX"));
   OnClockoleRefreshtime();
   m_clock.ShowWin();
}

void CMymfc29DView::OnUpdateClockoleLoad(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_clock.m_lpDispatch == NULL);
}
```

Listing 8.

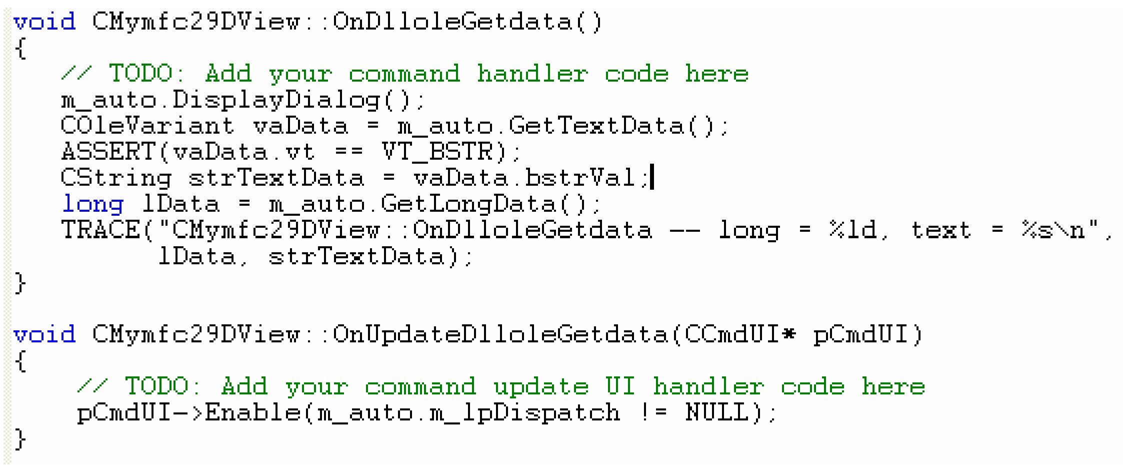
```cpp
void CMymfc29DView::OnClockoleRefreshtime()
{
   // TODO: Add your command handler code here
   COleDateTime now = COleDateTime::GetCurrentTime();
   m_clock.SetTime(now);
   m_clock.RefreshWin();
}

void CMymfc29DView::OnUpdateClockoleRefreshtime(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}

void CMymfc29DView::OnClockoleUnload()
{
      // TODO: Add your command handler code here
      m_clock.ReleaseDispatch();
}

void CMymfc29DView::OnUpdateClockoleUnload(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}
```



```
void CMymfc29DView::OnClockoleRefreshtime()
{
   // TODO: Add your command handler code here
   COleDateTime now = COleDateTime::GetCurrentTime();
   m_clock.SetTime(now);
   m_clock.RefreshWin();
}

void CMymfc29DView::OnUpdateClockoleRefreshtime(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}

void CMymfc29DView::OnClockoleUnload()
{
    // TODO: Add your command handler code here
    m_clock.ReleaseDispatch();
}

void CMymfc29DView::OnUpdateClockoleUnload(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}
```

Listing 9.

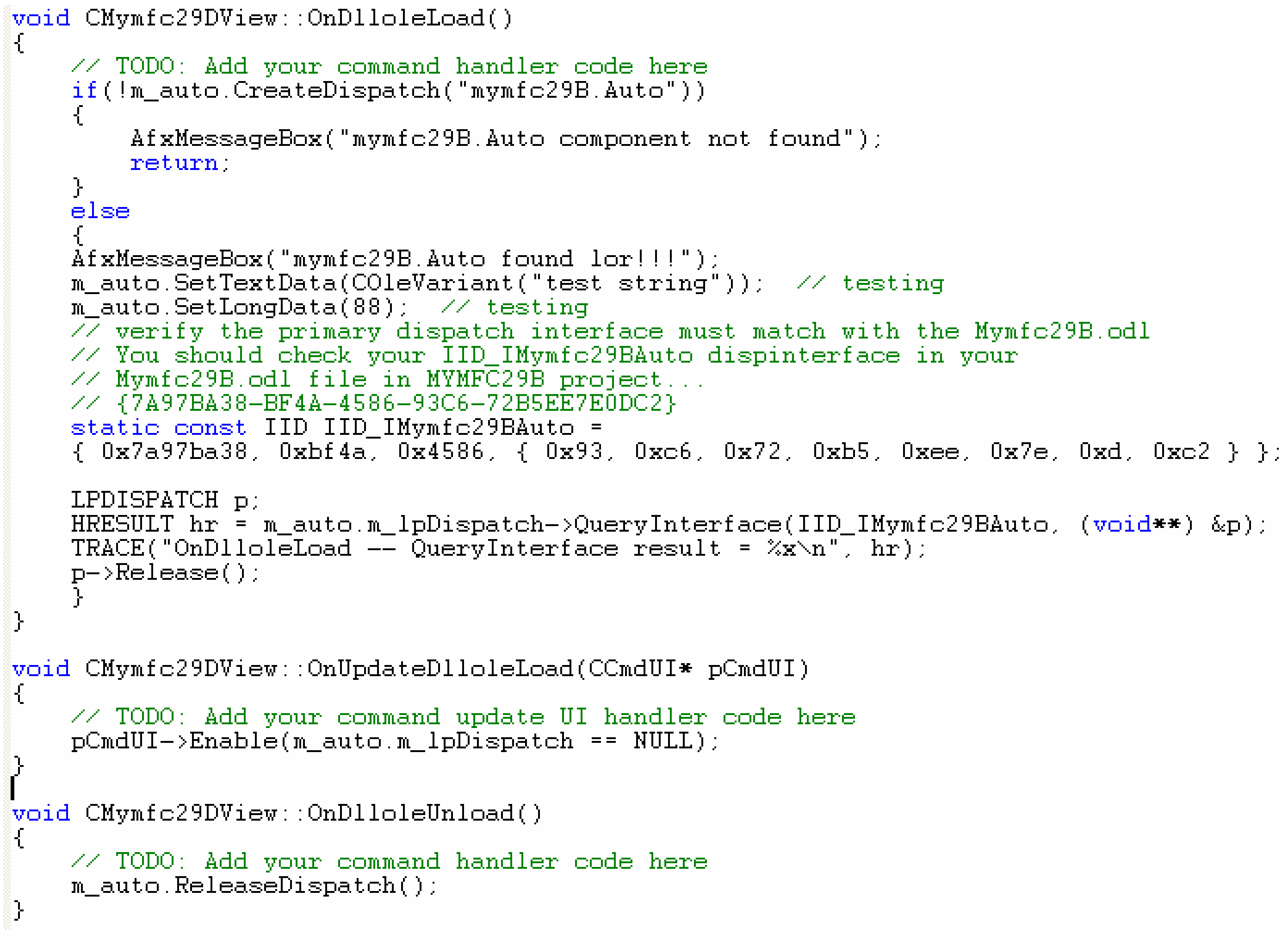
```
void CMymfc29DView::OnDlloleGetdata()
{
   // TODO: Add your command handler code here
   m_auto.DisplayDialog();
   COleVariant vaData = m_auto.GetTextData();
   ASSERT(vaData.vt == VT_BSTR);
   CString strTextData = vaData.bstrVal;
   long lData = m_auto.GetLongData();
   TRACE("CMymfc29DView::OnDlloleGetdata -- long = %ld, text = %s\n",
         lData, strTextData);
}

void CMymfc29DView::OnUpdateDlloleGetdata(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_auto.m_lpDispatch != NULL);
}
```



```
void CMymfc29DView::OnDlloleGetdata()
{
    // TODO: Add your command handler code here
    m_auto.DisplayDialog();
    COleVariant vaData = m_auto.GetTextData();
    ASSERT(vaData.vt == VT_BSTR);
    CString strTextData = vaData.bstrVal;
    long lData = m_auto.GetLongData();
    TRACE("CMymfc29DView::OnDlloleGetdata -- long = %ld, text = %s\n",
          lData, strTextData);
}

void CMymfc29DView::OnUpdateDlloleGetdata(CCmdUI* pCmdUI)
{
     // TODO: Add your command update UI handler code here
     pCmdUI->Enable(m_auto.m_lpDispatch != NULL);
}
```

Listing 10.

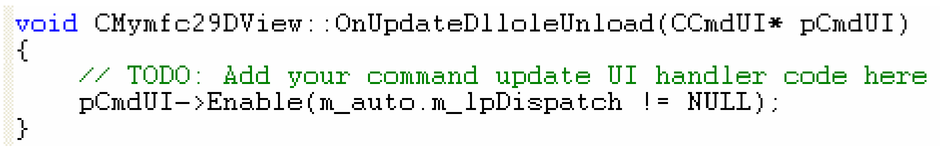
```
void CMymfc29DView::OnDlloleLoad()
{
      // TODO: Add your command handler code here
      if(!m_auto.CreateDispatch("mymfc29B.Auto"))
      {
            AfxMessageBox("mymfc29B.Auto component not found");
            return;
      }
      else
      {
      AfxMessageBox("mymfc29B.Auto found lor!!!");
      m_auto.SetTextData(COleVariant("test string"));  // testing
      m_auto.SetLongData(88);  // testing
      // verify the primary dispatch interface must match with the
Mymfc29B.odl
      // You should check your IID_IMymfc29BAuto dispinterface in your
      // Mymfc29B.odl file in MYMFC29B project...
      // {7A97BA38-BF4A-4586-93C6-72B5EE7E0DC2}
      static const IID IID_IMymfc29BAuto =
      { 0x7a97ba38, 0xbf4a, 0x4586, { 0x93, 0xc6, 0x72, 0xb5, 0xee, 0x7e,
0xd, 0xc2 } };

      LPDISPATCH p;
      HRESULT hr = m_auto.m_lpDispatch->QueryInterface(IID_IMymfc29BAuto,
(void**) &p);
      TRACE("OnDlloleLoad -- QueryInterface result = %x\n", hr);
      p->Release();
      }
}

void CMymfc29DView::OnUpdateDlloleLoad(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_auto.m_lpDispatch == NULL);
}

void CMymfc29DView::OnDlloleUnload()
{
      // TODO: Add your command handler code here
      m_auto.ReleaseDispatch();
}
```



```
void CMymfc29DView::OnDlloleLoad()
{
    // TODO: Add your command handler code here
    if(!m_auto.CreateDispatch("mymfc29B.Auto"))
    {
        AfxMessageBox("mymfc29B.Auto component not found");
        return;
    }
    else
    {
    AfxMessageBox("mymfc29B.Auto found lor!!!");
    m_auto.SetTextData(COleVariant("test string"));  // testing
    m_auto.SetLongData(88);  // testing
    // verify the primary dispatch interface must match with the Mymfc29B.odl
    // You should check your IID_IMymfc29BAuto dispinterface in your
    // Mymfc29B.odl file in MYMFC29B project...
    // {7A97BA38-BF4A-4586-93C6-72B5EE7E0DC2}
    static const IID IID_IMymfc29BAuto =
    { 0x7a97ba38, 0xbf4a, 0x4586, { 0x93, 0xc6, 0x72, 0xb5, 0xee, 0x7e, 0xd, 0xc2 } };

    LPDISPATCH p;
    HRESULT hr = m_auto.m_lpDispatch->QueryInterface(IID_IMymfc29BAuto, (void**) &p);
    TRACE("OnDlloleLoad -- QueryInterface result = %x\n", hr);
    p->Release();
    }
}

void CMymfc29DView::OnUpdateDlloleLoad(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_auto.m_lpDispatch == NULL);
}

void CMymfc29DView::OnDlloleUnload()
{
    // TODO: Add your command handler code here
    m_auto.ReleaseDispatch();
}
```

Listing 11.

```
void CMymfc29DView::OnUpdateDlloleUnload(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_auto.m_lpDispatch != NULL);
}
```



```
void CMymfc29DView::OnUpdateDlloleUnload(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_auto.m_lpDispatch != NULL);
}
```

Listing 12.

------------------------Warning----------------------------

```
// The following code seems not working for Excel 10/11 Office 2000/2003
// because of the obsolete classes. At least two functions will generate error,
// m_app.GetWorkbooks() and m_workbooks.Add()...
// It is for Excel8.olb/Office 97...and the menu item
// for the Excel Comp Execute also has been omitted for this example...
// You have to dig the Microsoft Office documentation...hohohohoho...:-)

/*
LPDISPATCH pRange, pWorkbooks;

   CWnd* pWnd = CWnd::FindWindow("XLMAIN", NULL);
```

```
if (pWnd != NULL) {
  TRACE("Excel window found\n");
  pWnd->ShowWindow(SW_SHOWNORMAL);
  pWnd->UpdateWindow();
  pWnd->BringWindowToTop();
}

m_app.SetSheetsInNewWorkbook(1);

VERIFY(pWorkbooks = m_app.GetWorkbooks());
m_workbooks.AttachDispatch(pWorkbooks);

LPDISPATCH pWorkbook = NULL;
if (m_workbooks.GetCount() == 0) {
   // Add returns a Workbook pointer, but we
   //  don't have a Workbook class
   pWorkbook = m_workbooks.Add(); // Save the pointer for
                                  //  later release
}
LPDISPATCH pWorksheets = m_app.GetWorksheets();
ASSERT(pWorksheets != NULL);
m_worksheets.AttachDispatch(pWorksheets);
LPDISPATCH pWorksheet = m_worksheets.GetItem(COleVariant((short) 1));

m_worksheet.AttachDispatch(pWorksheet);
m_worksheet.Select();

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A1")));
m_range[0].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A2")));
m_range[1].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A3")));
m_range[2].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A3"), COleVariant("C5")));
m_range[3].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A6")));
m_range[4].AttachDispatch(pRange);

m_range[4].SetValue(COleVariant(COleDateTime(2005, 4, 24, 15, 47, 8)));
// retrieve the stored date and print it as a string
COleVariant vaTimeDate = m_range[4].GetValue();
TRACE("returned date type = %d\n", vaTimeDate.vt);
COleVariant vaTemp;
vaTemp.ChangeType(VT_BSTR, &vaTimeDate);
CString str = vaTemp.bstrVal;
TRACE("date = %s\n", (const char*) str);

m_range[0].SetValue(COleVariant("test string"));

COleVariant vaResult0 = m_range[0].GetValue();
if (vaResult0.vt == VT_BSTR) {
  CString str = vaResult0.bstrVal;
  TRACE("vaResult0 = %s\n", (const char*) str);
}

m_range[1].SetValue(COleVariant(3.14159));

COleVariant vaResult1 = m_range[1].GetValue();
if (vaResult1.vt == VT_R8) {
  TRACE("vaResult1 = %f\n", vaResult1.dblVal);
}

m_range[2].SetFormula(COleVariant("=$A2*2.0"));
```

```
   COleVariant vaResult2 = m_range[2].GetValue();
   if (vaResult2.vt == VT_R8) {
     TRACE("vaResult2 = %f\n", vaResult2.dblVal);
   }

   COleVariant vaResult2a = m_range[2].GetFormula();
   if (vaResult2a.vt == VT_BSTR) {
     CString str = vaResult2a.bstrVal;
     TRACE("vaResult2a = %s\n", (const char*) str);
   }

   m_range[3].FillRight();
   m_range[3].FillDown();

    // cleanup
    if (pWorkbook != NULL) {
           pWorkbook->Release();
   }
}

void CMymfc29DView::OnUpdateExceloleExecute(CCmdUI* pCmdUI)
{
       // TODO: Add your command update UI handler code here
       pCmdUI->Enable(m_app.m_lpDispatch != NULL);
}*/
```

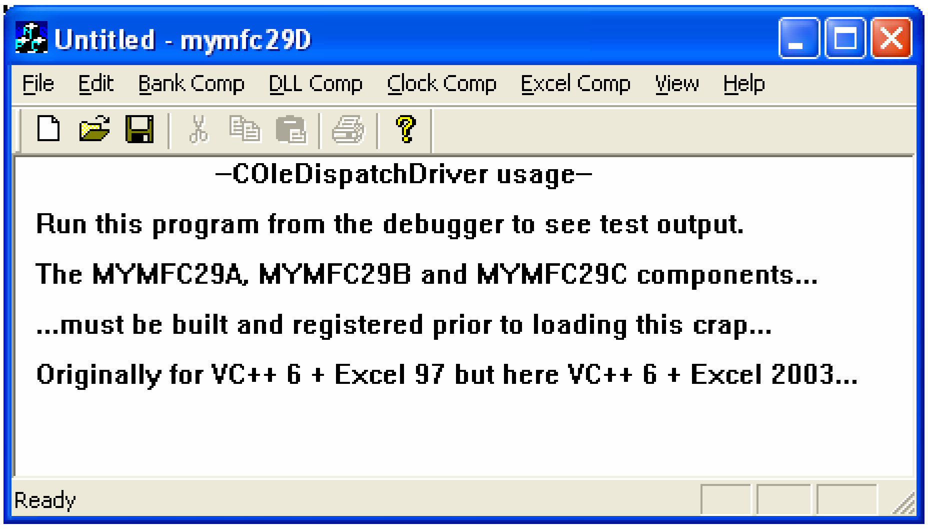
---------------End of Warning----------------

```
// The following code should be ok but do nothing, just load the Excel OLE
void CMymfc29DView::OnExceloleLoad()
{
   // TODO: Add your command handler code here
   // if Excel is already running, attach to it, otherwise start it
   LPDISPATCH pDisp;
   LPUNKNOWN pUnk;
   CLSID clsid;
   TRACE("Entering CMymfc29DView::OnExcelLoad\n");
   BeginWaitCursor();
   // Excel for Office 2003. Adjust accordingly for your Excel version
   ::CLSIDFromProgID(L"Excel.Application.11", &clsid); // from registry
   if(::GetActiveObject(clsid, NULL, &pUnk) == S_OK)
   {
      VERIFY(pUnk->QueryInterface(IID_IDispatch, (void**) &pDisp) == S_OK);
      m_app.AttachDispatch(pDisp);
      pUnk->Release();
      TRACE(" attach complete\n");
      // you can use SPY++/Windows Task Manager to verify
      // your Excel process is attached...
   }
   else
   {
          // Excel for Office 2003. Adjust accordingly for your Excel version
          if(!m_app.CreateDispatch("Excel.Application.11"))
          {
              AfxMessageBox("Excel program not found");
          }
       TRACE(" create complete\n");
   }
   EndWaitCursor();
}

void CMymfc29DView::OnUpdateExceloleLoad(CCmdUI* pCmdUI)
{
       // TODO: Add your command update UI handler code here
       pCmdUI->Enable(m_app.m_lpDispatch == NULL);
}
```

```
// The following code should be ok but do nothing, just load the Excel OLE
void CMymfc29DView::OnExceloleLoad()
{
   // TODO: Add your command handler code here
   // if Excel is already running, attach to it, otherwise start it
   LPDISPATCH pDisp;
   LPUNKNOWN pUnk;
   CLSID clsid;
   TRACE("Entering CMymfc29DView::OnExcelLoad\n");
   BeginWaitCursor();
   // Excel for Office 2003. Adjust accordingly for your Excel version
   ::CLSIDFromProgID(L"Excel.Application.11", &clsid); // from registry
   if(::GetActiveObject(clsid, NULL, &pUnk) == S_OK)
   {
      VERIFY(pUnk->QueryInterface(IID_IDispatch, (void**) &pDisp) == S_OK);
      m_app.AttachDispatch(pDisp);
      pUnk->Release();
      TRACE(" attach complete\n");
      // you can use SPY++ to verify your Excel process is attached...
    }
   else
   {
       // Excel for Office 2003. Adjust accordingly for your Excel version
       if(!m_app.CreateDispatch("Excel.Application.11"))
       {
            AfxMessageBox("Excel program not found");
       }
       TRACE(" create complete\n");
   }
   EndWaitCursor();
}

void CMymfc29DView::OnUpdateExceloleLoad(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_app.m_lpDispatch == NULL);
}
```

Listing 13.

Let play around with our program. Make sure you have already build/run/registered the MYMFC29A, MYMFC29B and MYMFC29C components in order to test MYMFC29D program. Build and run MYMFC29D, the output is shown below. Test all the components through the menu items.


Untitled - mymfc29D
File Edit Bank Comp DLL Comp Clock Comp Excel Comp View Help
Ready

-COleDispatchDriver usage-

Run this program from the debugger to see test output.

The MYMFC29A, MYMFC29B and MYMFC29C components...

...must be built and registered prior to loading this crap...

Originally for VC++ 6 + Excel 97 but here VC++ 6 + Excel 2003...

Figure 52: MYMFC29D output.

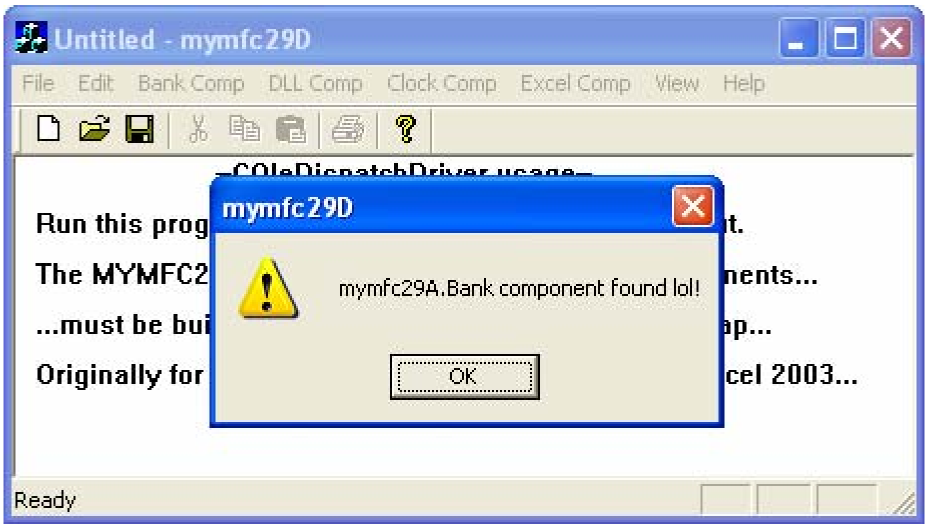

mymfc29A.Bank component found lol!

Figure 53: MYMFC29D – testing the **Bank Comp Load** menu.

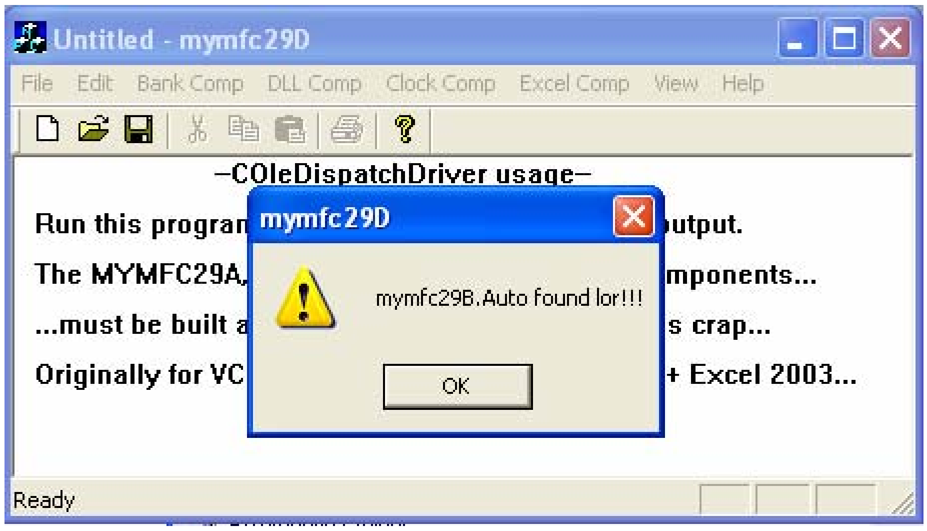

mymfc29B.Auto found lor!!!

Figure 54: MYMFC29D – testing the **DLL Comp Load** menu.

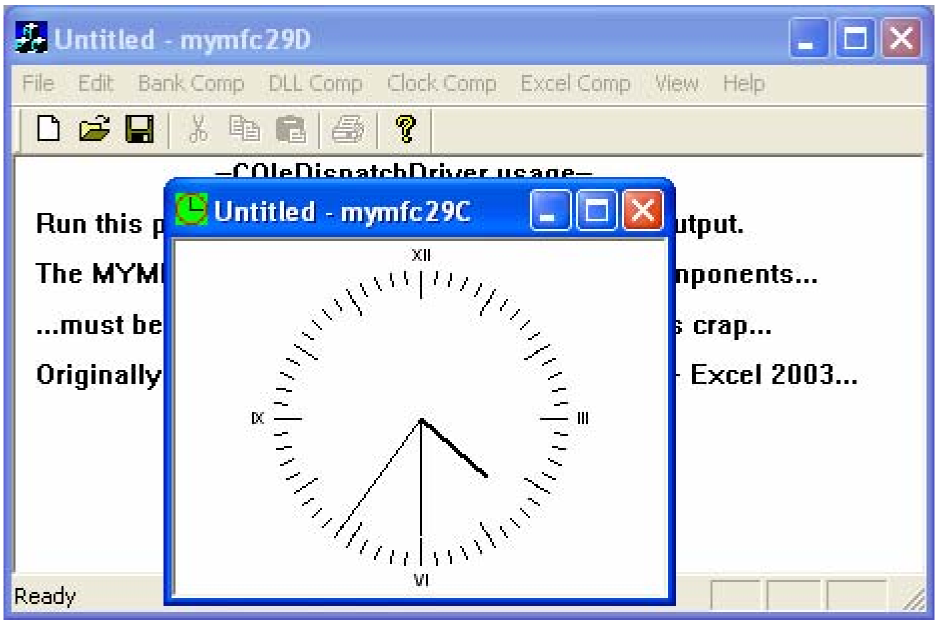
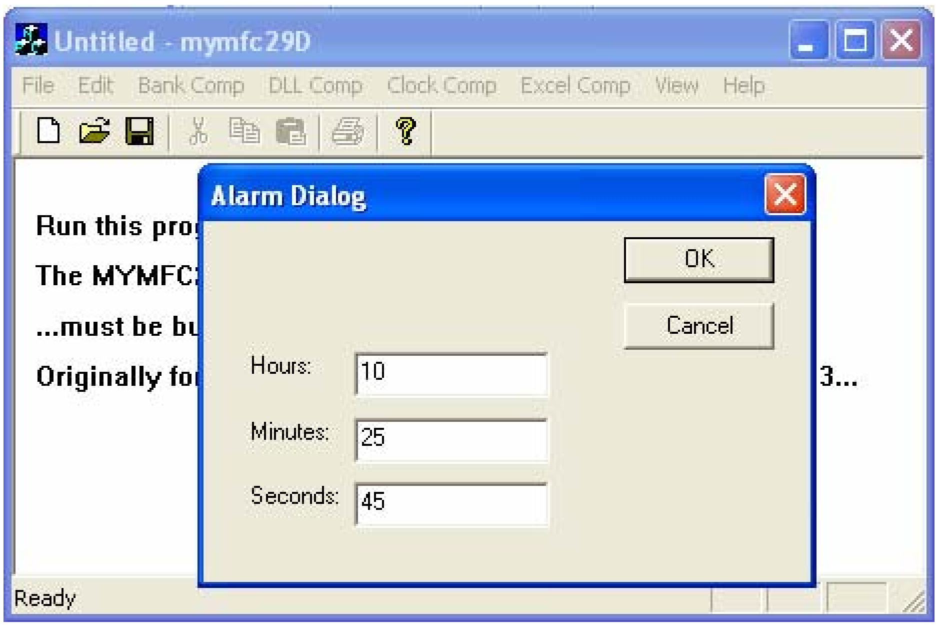


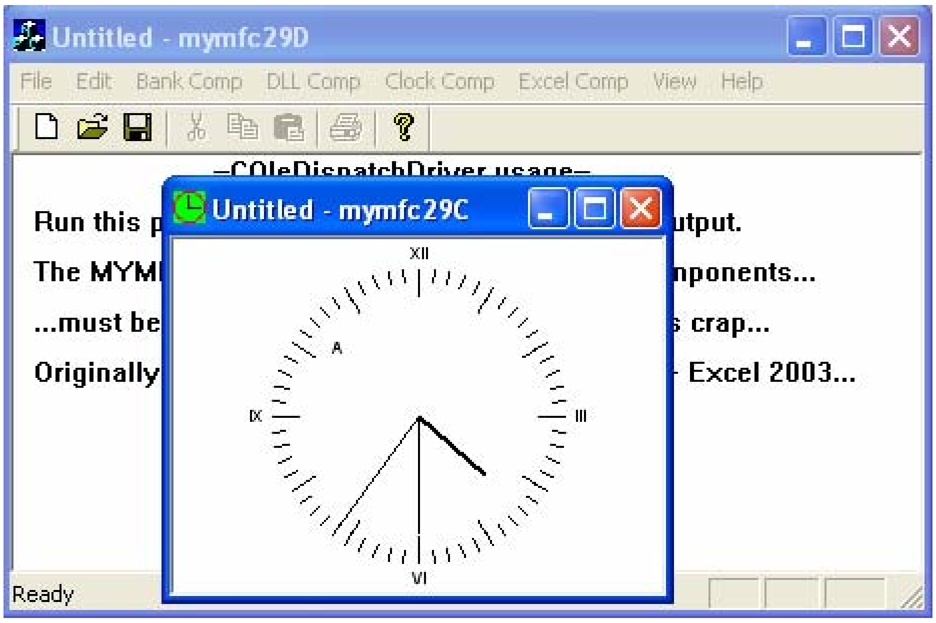
Figure 55: MYMFC29D – testing the **Clock Comp Load** menu.


Untitled - mymfc29D
File Edit Bank Comp DLL Comp Clock Comp Excel Comp View Help
Alarm Dialog
OK
Cancel
Ready

Run this pro
The MYMFC
...must be bu
Originally fo
3...

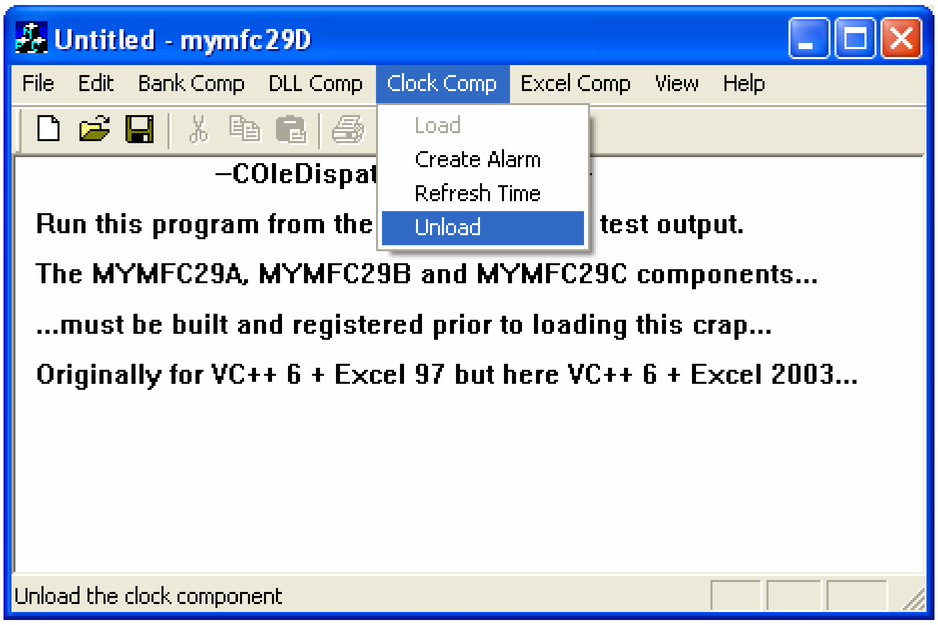
Hours: 10
Minutes: 25
Seconds: 45

Figure 56: MYMFC29D – testing the **Clock Comp Create Alarm** menu. Enter the alarm values.


Untitled - mymfc29D
File Edit Bank Comp DLL Comp Clock Comp Excel Comp View Help
Untitled - mymfc29C
XII
III
VI
IX
A
Ready

–COleDispatchDriver usage–
Run this p utput.
The MYM nponents...
...must be s crap...
Originally - Excel 2003...

Figure 57: MYMFC29D – the alarm was set.


Untitled - mymfc29D
File Edit Bank Comp DLL Comp Clock Comp Excel Comp View Help
Load
Create Alarm
Refresh Time
Unload

–COleDispat

Run this program from the test output.

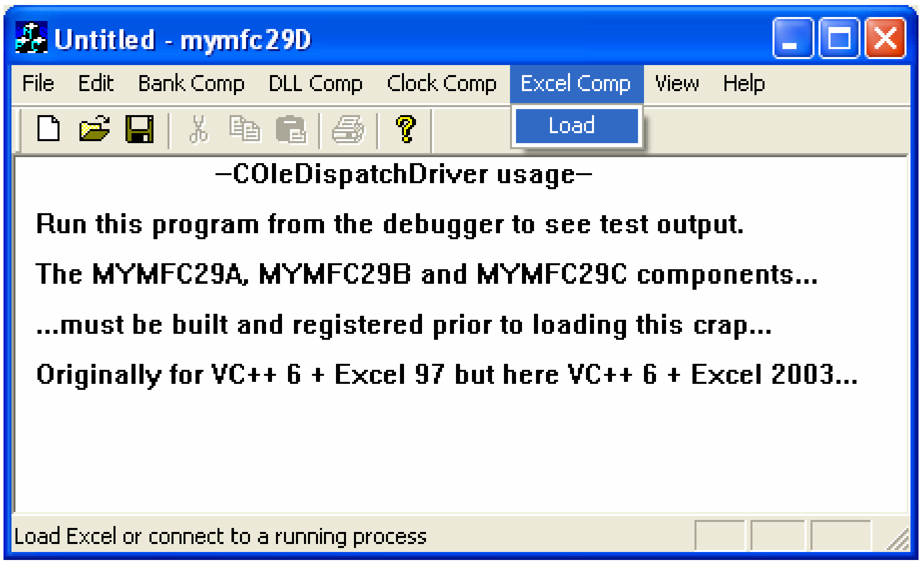
The MYMFC29A, MYMFC29B and MYMFC29C components...

...must be built and registered prior to loading this crap...

Originally for VC++ 6 + Excel 97 but here VC++ 6 + Excel 2003...

Unload the clock component

Figure 58: MYMFC29D – testing the **Clock Comp Unload** menu, unloading the clock.


Untitled - mymfc29D
File Edit Bank Comp DLL Comp Clock Comp Excel Comp View Help
Load

–COleDispatchDriver usage–

Run this program from the debugger to see test output.

The MYMFC29A, MYMFC29B and MYMFC29C components...

...must be built and registered prior to loading this crap...

Originally for VC++ 6 + Excel 97 but here VC++ 6 + Excel 2003...

Load Excel or connect to a running process

Figure 59: MYMFC29D – testing the **Excel Comp Load** menu.

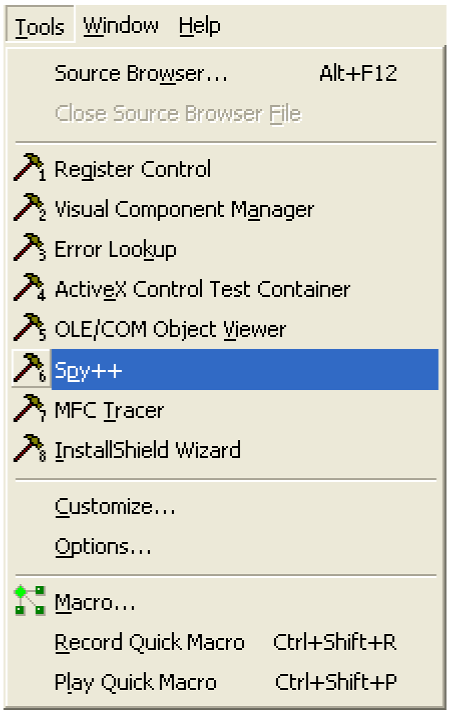
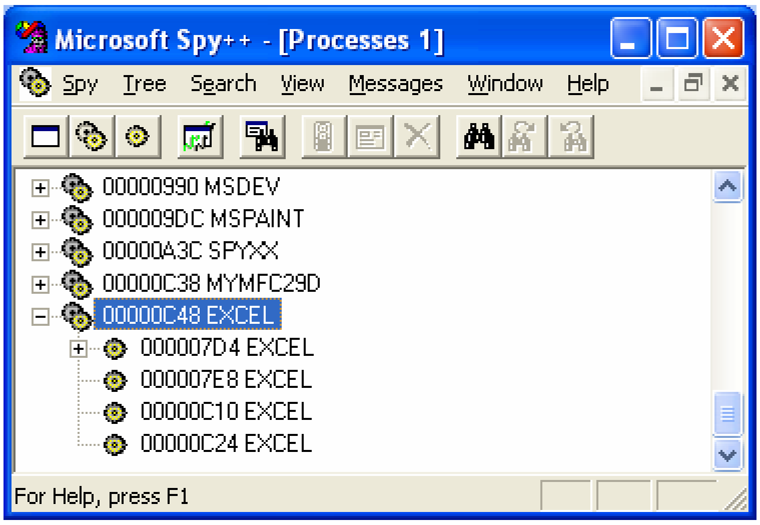


Figure 60: Viewing the loaded Excel processes using SPY++.



```
00000990 MSDEV
000009DC MSPAINT
00000A3C SPYXX
00000C38 MYMFC29D
00000C48 EXCEL
    000007D4 EXCEL
    000007E8 EXCEL
    00000C10 EXCEL
    00000C24 EXCEL
```

Figure 61: The Excel program was loaded.

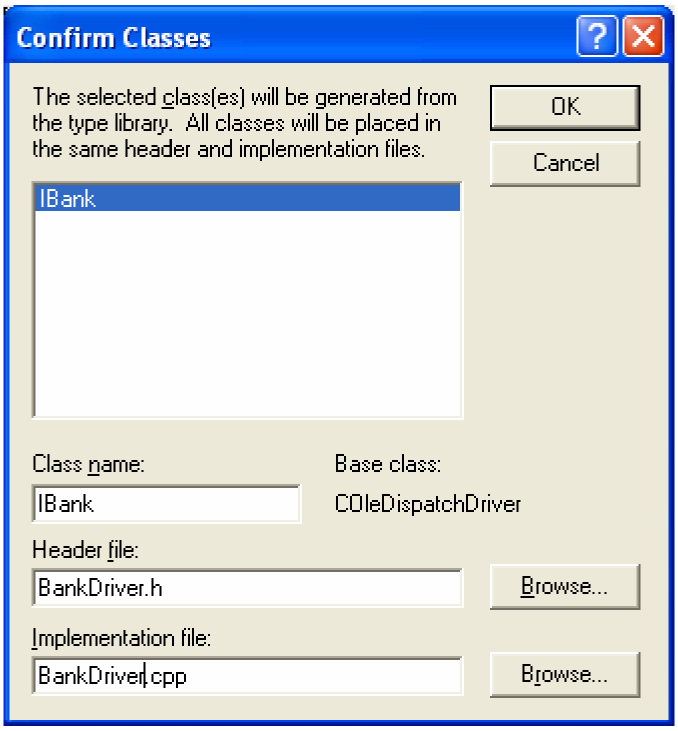
The Story: Type Libraries and ODL Files etc.

We've told you that **type libraries** aren't necessary for the MFC `IDispatch` implementation, but Visual C++ has been quietly generating and updating type libraries for all your components. What good are these type libraries? VBA can use a type library to browse your component's methods and properties, and it can use the type library for improved access to properties and methods, a process called **early binding** described later in this module. But we're building a C++ client program here, not a VBA program. It so happens that ClassWizard can read a component's type library and use the information to generate C++ code for the client to use to "drive" an Automation component. AppWizard initializes a project's **Object Description Language** (ODL) file when you first create it. ClassWizard edits this file each time you generate a new Automation component class or add properties and methods to an existing class. Unlike it does with the ClassWizard (CLW) file, ClassWizard can't rebuild an ODL file from the contents of your source files. If you mess up your ODL file, you'll have to re-create it manually.

When you were adding properties and methods to your component classes, ClassWizard was updating the project's ODL file. This file is a **text file** that describes the component in an ODL. (Your `GUID` will be different if you used AppWizard to generate this project.) Here's the ODL file for the bank component:

```
// Originally just a manually created odl template

[ uuid(40980C17-DB2B-498F-85AC-BE88F2B549CF), version(1.0) ]
library mymfc29A
{
	importlib("stdole32.tlb");
	importlib("stdole2.tlb");

	//  Primary dispatch interface for CBank

	[ uuid(122FA935-F2E8-4EB0-B974-D0BF4C59E53D) ]
	dispinterface IBank
	{
		properties:
			// NOTE - ClassWizard will maintain property information
here.
			//    Use extreme caution when editing this section.
			//{{AFX_ODL_PROP(CBank)
			[id(1)] double Balance;
			//}}AFX_ODL_PROP

		methods:
			// NOTE - ClassWizard will maintain method information
here.
			//    Use extreme caution when editing this section.
			//{{AFX_ODL_METHOD(CBank)
			[id(2)] double Withdrawal(double dAmount);
			[id(3)] void Deposit(double dAmount);
			//}}AFX_ODL_METHOD

	};

	//  Class information for CBank

	[ uuid(58E7A2CE-C23D-4A09-A1AD-E4710AE28E4D) ]
	coclass Bank
	{
		[default] dispinterface IBank;
	};

	//{{AFX_APPEND_ODL}}
};
```

The **ODL** file has a unique `GUID` type library identifier, `40980C17-DB2B-498F-85AC-BE88F2B549CF`, and it completely describes the bank component's properties and methods under a dispinterface named `IBank`. In addition, it specifies the dispinterface `GUID`, `122FA935-F2E8-4EB0-B974-D0BF4C59E53D`, which is the same `GUID` that's in the interface map of the `CBank` class. You'll see the significance of this GUID when you read the "VBA Early Binding" section near the end of this module. The `CLSID`, `58E7A2CE-C23D-4A09-A1AD-E4710AE28E4D`, is what a VBA browser can actually use to load your component.
Anyway, when you build your component project, Visual C++ invokes the **MIDL** utility, which reads the ODL file and generates a binary TLB file in your project's debug or release subdirectory. Now when you develop a C++ client program, you can ask ClassWizard to generate a driver class from the component project's TLB file.
The **MIDL** utility generates the type library in a stand-alone **TLB** file, and that's what Automation controllers such as Excel look for. ActiveX controls have their type libraries bound into their resources.
To actually do this, you click the ClassWizard **Add Class** button and then select **From A Type Library** from the drop-down list. You navigate to the component project's **TLB** file, and then ClassWizard shows you a dialog similar to the illustration below.



The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.

IBank

Class name: IBank

Base class: COleDispatchDriver

Header file: BankDriver.h

Implementation file: BankDriver.cpp

Figure 62: The `IBank` class extracted from type library.

`IBank` is the dispinterface specified in the ODL file. You can keep this name for the class if you want, and you can specify the H and CPP filenames. If a type library contains several interfaces you can make multiple selections. You'll see the generated controller classes in the sections that follow.

The Controller Class for mymfc29A.exe

ClassWizard generated the `IBank` class (derived from `COleDispatchDriver`) listed in Listing 14. Look closely at the member function implementations. Note the first parameters of the `GetProperty()`, `SetProperty()`, and `InvokeHelper()` function calls. These are hard-coded `DISPID`s for the component's properties and methods, as determined by the component's dispatch map sequence.
If you use ClassWizard to delete a property and then add the property back, you'll probably change the component's dispatch IDs. That means that you'll have to **regenerate** or **edit** the controller class so that the IDs match.

```
BANKDRIVER.H

// Machine generated IDispatch wrapper class(es) created with ClassWizard
/////////////////////////////////////////////////////////////////////////////
// IBank wrapper class

class IBank : public COleDispatchDriver
{
public:
	IBank() {}		// Calls COleDispatchDriver default constructor
	IBank(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
	IBank(const IBank& dispatchSrc) : COleDispatchDriver(dispatchSrc) {}

// Attributes
public:
	double GetBalance();
	void SetBalance(double);

// Operations
public:
	double Withdrawal(double dAmount);
	void Deposit(double dAmount);
};
```

```
BANKDRIVER.CPP
// Machine generated IDispatch wrapper class(es) created with ClassWizard

#include "stdafx.h"
#include "bankdriver.h"

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



/////////////////////////////////////////////////////////////////////////////
// IBank properties

double IBank::GetBalance()
{
	double result;
	GetProperty(0x1, VT_R8, (void*)&result);
	return result;
}

void IBank::SetBalance(double propVal)
{
	SetProperty(0x1, VT_R8, propVal);
}

/////////////////////////////////////////////////////////////////////////////
// IBank operations

double IBank::Withdrawal(double dAmount)
{
	double result;
	static BYTE parms[] =
		VTS_R8;
	InvokeHelper(0x2, DISPATCH_METHOD, VT_R8, (void*)&result, parms, dAmount);
	return result;
}

void IBank::Deposit(double dAmount)
{
	static BYTE parms[] =
		VTS_R8;
	InvokeHelper(0x3, DISPATCH_METHOD, VT_EMPTY, NULL, parms, dAmount);
}
```

Listing 14: The `IBank` class listing.

The `CMymfc29DView` class has a data member `m_bank` of class `IBank`. The `CMymfc29DView` member functions for the **Mymfc29A.Bank** component are listed below. They are hooked up to options on the **MYMFC29D** main menu. Of particular interest is the `OnBankoleLoad()` function. The `COleDispatchDriver::CreateDispatch` function loads the component program (by calling `CoGetClassObject()` and `IClassFactory::CreateInstance`) and then calls `QueryInterface()` to get an `IDispatch` pointer, which it stores in the object's `m_lpDispatch` data member. The `COleDispatchDriver::ReleaseDispatch` function, called in `OnBankoleUnload()`, calls `Release()` on the pointer.

```
void CMymfc29DView::OnBankoleLoad()
{
   if(!m_bank.CreateDispatch("Mymfc29A.Bank")) {
        AfxMessageBox("Mymfc29A.Bank component not found");
```

```
            return;
        }
    }

    void CMymfc29DView::OnUpdateBankoleLoad(CCmdUI* pCmdUI)
    {
        pCmdUI->Enable(m_bank.m_lpDispatch == NULL);
    }

    void CMymfc29DView::OnBankoleTest()
    {
        m_bank.Deposit(20.0);
        m_bank.Withdrawal(15.0);
        TRACE("new balance = %f\n",
        m_bank.GetBalance());
    }

    void CMymfc29DView::OnUpdateBankoleTest(CCmdUI* pCmdUI)
    {
        pCmdUI->Enable(m_bank.m_lpDispatch != NULL);
    }

    void CMymfc29DView::OnBankoleUnload()
    {
        m_bank.ReleaseDispatch();
    }

    void CMymfc29DView::OnUpdateBankoleUnload(CCmdUI* pCmdUI)
    {
        pCmdUI->Enable(m_bank.m_lpDispatch != NULL);
    }
```

The Controller Class for mymfc29B.dll

Listing 15 shows the class header file generated by ClassWizard.

```
AUTODRIVER.H

// Machine generated IDispatch wrapper class(es) created with ClassWizard
/////////////////////////////////////////////////////////////////////////////
// IMymfc29BAuto wrapper class

class IMymfc29BAuto : public COleDispatchDriver
{
public:
       IMymfc29BAuto() {}         // Calls COleDispatchDriver default
constructor
       IMymfc29BAuto(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
       IMymfc29BAuto(const IMymfc29BAuto& dispatchSrc) :
COleDispatchDriver(dispatchSrc) {}

// Attributes
public:
       long GetLongData();
       void SetLongData(long);
       VARIANT GetTextData();
       void SetTextData(const VARIANT&);

// Operations
public:
       BOOL DisplayDialog();
};
```

Listing 15: The `Mymfc29BAuto` class header file.

Notice that each property requires separate Get and Set functions in the client class, even though a data member in the component represents the property.
The view class header has a data member `m_auto` of class `IMymfc29BAuto`. Here are some DLL-related command handler member functions from **mymfc29DView.cpp**:

```
void CMymfc29DView::OnDlloleGetdata()
{
    m_auto.DisplayDialog();
    COleVariant vaData = m_auto.GetTextData();
    ASSERT(vaData.vt == VT_BSTR);
    CString strTextData = vaData.bstrVal;
    long lData = m_auto.GetLongData();
    TRACE("CMymfc29DView::OnDlloleGetdata – long = %ld, text = %s\n",
          lData, strTextData);
}

void CMymfc29DView::OnUpdateDlloleGetdata(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_auto.m_lpDispatch != NULL);
}

void CMymfc29DView::OnDlloleLoad()
{
    if(!m_auto.CreateDispatch("Mymfc29B.Auto")) {
        AfxMessageBox("Mymfc29B.Auto component not found");
        return;
    }
    m_auto.SetTextData(COleVariant("test"));  // testing
    m_auto.SetLongData(79);  // testing
    // verify dispatch interface
    //  {A9515AD7-5B85-11D0-848F-00400526305B}
    static const IID IID_IMymfc29BAuto = { 0xa9515ad7, 0x5b85, 0x11d0, { 0x84,
0x8f, 0x0, 0x40, 0x5, 0x26, 0x30, 0x5b } };
    LPDISPATCH p;
    HRESULT hr = m_auto.m_lpDispatch->QueryInterface(IID_IMymfc29BAuto,
(void**) &p);
    TRACE("OnDlloleLoad – QueryInterface result = %x\n", hr);
    p->Release();
}

void CMymfc29DView::OnUpdateDlloleLoad(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_auto.m_lpDispatch == NULL);
}

void CMymfc29DView::OnDlloleUnload()
{
    m_auto.ReleaseDispatch();
}

void CMymfc29DView::OnUpdateDlloleUnload(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_auto.m_lpDispatch != NULL);
}
```

The Controller Class for mymfc29C.exe

Listing 16 shows the headers for the `IMymfc29C` and `IAlarm` classes, which drive the MYMFC29C Automation component.

```
CLOCKDRIVER.H

// Machine generated IDispatch wrapper class(es) created with ClassWizard
/////////////////////////////////////////////////////////////////////////////
// IMymfc29C wrapper class
```

```
class IMymfc29C : public COleDispatchDriver
{
public:
        IMymfc29C() {}             // Calls COleDispatchDriver default
constructor
        IMymfc29C(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
        IMymfc29C(const IMymfc29C& dispatchSrc) :
COleDispatchDriver(dispatchSrc) {}

// Attributes
public:
        DATE GetTime();
        void SetTime(DATE);

// Operations
public:
        VARIANT GetFigure(short n);
        void SetFigure(short n, const VARIANT& newValue);
        void RefreshWin();
        void ShowWin();
        LPDISPATCH CreateAlarm(DATE Time);
};
/////////////////////////////////////////////////////////////////////////////
// IAlarm wrapper class

class IAlarm : public COleDispatchDriver
{
public:
        IAlarm() {}          // Calls COleDispatchDriver default constructor
        IAlarm(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
        IAlarm(const IAlarm& dispatchSrc) : COleDispatchDriver(dispatchSrc) {}

// Attributes
public:
        DATE GetTime();
        void SetTime(DATE);

// Operations
public:
};
```

Listing 16: The `IMymfc29C` and `IAlarm` class header files.

Of particular interest is the `IMymfc29C::CreateAlarm` member function in **ClockDriver.cpp**:

```
LPDISPATCH IMymfc29C::CreateAlarm(DATE time)
{
    LPDISPATCH result;
    static BYTE parms[] = VTS_DATE;
    InvokeHelper(0x3, DISPATCH_METHOD, VT_DISPATCH, (void*)&result,
        parms, time);
    return result;
}
```

This function can be called only after the clock object (document) has been constructed. It causes the MYMFC29C component to construct an alarm object and return an `IDispatch` pointer with a reference count of 1. The `COleDispatchDriver::AttachDispatch` function connects that pointer to the client's `m_alarm` object, but if that object already has a dispatch pointer, the old pointer is released. That's why, if you watch the Debug window, you'll see that the old MYMFC29C instance exits immediately after you ask for a new instance. You'll have to test this behavior with the Excel driver because MYMFC29D disables the **Load** menu option when the clock is running. The view class has the data members, `m_clock` and `m_alarm`. Here are the view class command handlers:

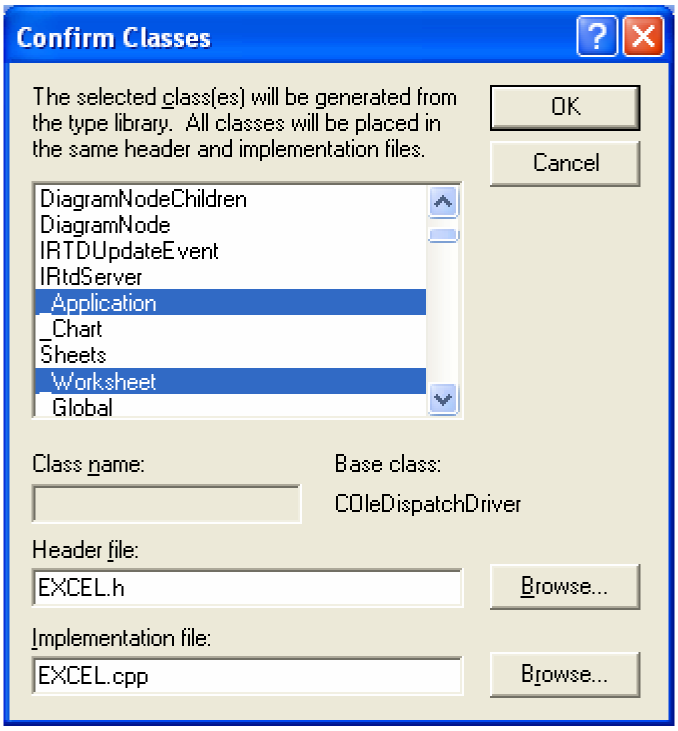
```
void CMymfc29DView::OnClockoleCreatealarm()
{
    CAlarmDialog dlg;
    if (dlg.DoModal() == IDOK)
    {
        COleDateTime dt(1995, 12, 23, dlg.m_nHours, dlg.m_nMinutes,
            dlg.m_nSeconds);
        LPDISPATCH pAlarm = m_clock.CreateAlarm(dt);
        m_alarm.AttachDispatch(pAlarm);  // releases prior object!
        m_clock.RefreshWin();
    }
}

void CMymfc29DView::OnUpdateClockoleCreatealarm(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}

void CMymfc29DView::OnClockoleLoad()
{
   if(!m_clock.CreateDispatch("Mymfc29C.Document"))
   {
        AfxMessageBox("Mymfc29C.Document component not found");
        return;
   }
    m_clock.SetFigure(0, COleVariant("XII"));
    m_clock.SetFigure(1, COleVariant("III"));
    m_clock.SetFigure(2, COleVariant("VI"));
    m_clock.SetFigure(3, COleVariant("IX"));
    OnClockoleRefreshtime();
    m_clock.ShowWin();
}

void CMymfc29DView::OnUpdateClockoleLoad(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.m_lpDispatch == NULL);
}

void CMymfc29DView::OnClockoleRefreshtime()
{
    COleDateTime now = COleDateTime::GetCurrentTime();
    m_clock.SetTime(now);
    m_clock.RefreshWin();
}

void CMymfc29DView::OnUpdateClockoleRefreshtime(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}

void CMymfc29DView::OnClockoleUnload()
{
    m_clock.ReleaseDispatch();
}

void CMymfc29DView::OnUpdateClockoleUnload(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.m_lpDispatch != NULL);
}
```

Controlling Microsoft Excel

The MYMFC29D program contains code that loads Excel, creates a workbook, and reads from and writes to cells from the active worksheet. Controlling Excel is exactly like controlling an **MFC Automation component**, but you need to know about a few Excel peculiarities.

If you study Excel VBA, you'll notice that you can use **more than 100 "objects"** in your programs. All of these objects are accessible through **Automation**, but if you write an MFC Automation client program, you'll need to know about the objects' properties and methods. Ideally, you'd like a C++ class for each object, with member functions coded to the proper dispatch IDs.

Excel has its own **type library**, found in the file **Excel8.olb** (different file name with same OLB extension for Excel of the Office10 and Office11), usually in the \Program Files\Microsoft Office\Office or \Program Files\Microsoft Office\Office10 – Office 2000 or \Program Files\Microsoft Office\Office11 – Office 2003 directory. ClassWizard can read this file, exactly as it reads TLB files, to create C++ driver classes for individual Excel objects. It makes sense to select the objects you need and then combine the classes into a single pair of files, as shown in Figure 63.


Confirm Classes

The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.

DiagramNodeChildren
DiagramNode
IRTDUpdateEvent
IRtdServer
_Application
_Chart
Sheets
_Worksheet
_Global

Class name:

Base class: COleDispatchDriver

Header file: EXCEL.h

Implementation file: EXCEL.cpp

Figure 63: ClassWizard can create C++ classes for the Excel objects listed in **Excel8.olb**.

You might need to edit the generated code to suit your needs. Let's look at an example. If you use ClassWizard to generate a driver class for the Worksheet object, you get a `GetRange()` member function, as shown here:

```
LPDISPATCH _Worksheet::GetRange(const VARIANT& Cell1, const VARIANT& Cell2)
{
    LPDISPATCH result;
    static BYTE parms[] = VTS_VARIANT VTS_VARIANT;
    InvokeHelper(0xc5, DISPATCH_PROPERTYGET, VT_DISPATCH,
        (void*)&result, parms, &Cell1, &Cell2);
    return result;
}
```

You know (from the Excel VBA documentation) that you can call the method with either a single cell (one parameter) or a rectangular area specified by two cells (two parameters). Remember: you can omit optional parameters in a call to `InvokeHelper()`. Now it makes sense to add a second overloaded `GetRange()` function with a single cell parameter like this:

```
LPDISPATCH _Worksheet::GetRange(const VARIANT& Cell1) // added
{
    LPDISPATCH result;
    static BYTE parms[] = VTS_VARIANT;
    InvokeHelper(0xc5, DISPATCH_PROPERTYGET, VT_DISPATCH,
        (void*)&result, parms, &Cell1);
    return result;
}
```

How do you know which functions to fix up? They're the functions you decide to use in your program. You'll have to read the Excel VBA reference manual to figure out the required parameters and return values.
The MYMFC29D program uses the Excel objects and contains the corresponding classes shown in the table below. All the code for these objects is contained in the files **excel8.h** and **excel8.cpp**.

Object/Class	View Class Data Member
_Application	m_app
Range	m_range[5]
_Worksheet	m_worksheet
Workbooks	m_workbooks
Worksheets	m_worksheets

Table 11.

The following view member function, `OnExceloleLoad()`, handles the **Excel Comp Load** menu command. This function must work if the user already has Excel running on the desktop. The COM `GetActiveObject()` function tries to return an `IUnknown` pointer for Excel. `GetActiveObject()` requires a class ID, so we must first call `CLSIDFromProgID()`. If `GetActiveObject()` is successful, we call `QueryInterface()` to get an `IDispatch` pointer and we attach it to the view's `m_app` controller object of class `_Application`. If `GetActiveObject()` is unsuccessful, we call `COleDispatchDriver::CreateDispatch`, as we did for the other components.

```
void CMymfc29DView::OnExceloleLoad()
{   // If Excel is already running, attach to it; otherwise, start it
    LPDISPATCH pDisp;
    LPUNKNOWN pUnk;
    CLSID clsid;
    TRACE("Entering CMymfc29DView::OnExcelLoad\n");
    BeginWaitCursor();
    ::CLSIDFromProgID(L"Excel.Application.11", &clsid); // from Registry
    if(::GetActiveObject(clsid, NULL, &pUnk) == S_OK) {
        VERIFY(pUnk->QueryInterface(IID_IDispatch, (void**) &pDisp) == S_OK);
        m_app.AttachDispatch(pDisp);
        pUnk->Release();
        TRACE(" attach complete\n");
    }
    else
    {
        if(!m_app.CreateDispatch("Excel.Application.11"))
        {
            AfxMessageBox("Excel 2003 program not found");
        }
        TRACE(" create complete\n");
    }
    EndWaitCursor();
}
```

`OnExceloleExecute()` is the command handler for the **Execute** item in the **Excel Comp** menu. Its first task is to find the Excel main window and bring it to the top. We must write some Windows code here because a method for this purpose couldn't be found. We must also create a workbook if no workbook is currently open.
We have to watch our method return values closely. The Workbooks `Add` method, for example, returns an `IDispatch` pointer for a Workbook object and, of course, increments the reference count. If we generated a class for Workbook, we could call `COleDispatchDriver::AttachDispatch` so that `Release()` would be called when the Workbook object was destroyed. Because we don't need a Workbook class, we'll simply release the pointer at the end of the function. If we don't properly clean up our pointers, we might get memory-leak messages from the Debug version of MFC.
The rest of the `OnExceloleExecute()` function accesses the cells in the worksheet. It's easy to get and set numbers, dates, strings, and formulas. The C++ code is similar to the VBA code you would write to do the same job.

```
void CMymfc29DView::OnExceloleExecute()
{
   LPDISPATCH pRange, pWorkbooks;

   CWnd* pWnd = CWnd::FindWindow("XLMAIN", NULL);
   if (pWnd != NULL) {
     TRACE("Excel window found\n");
```

```
  pWnd->ShowWindow(SW_SHOWNORMAL);
  pWnd->UpdateWindow();
  pWnd->BringWindowToTop();
}

m_app.SetSheetsInNewWorkbook(1);

VERIFY(pWorkbooks = m_app.GetWorkbooks());
m_workbooks.AttachDispatch(pWorkbooks);

LPDISPATCH pWorkbook = NULL;
if (m_workbooks.GetCount() == 0) {
   // Add returns a Workbook pointer, but we
   //  don't have a Workbook class
   pWorkbook = m_workbooks.Add(); // Save the pointer for
                                  //  later release
}
LPDISPATCH pWorksheets = m_app.GetWorksheets();
ASSERT(pWorksheets != NULL);
m_worksheets.AttachDispatch(pWorksheets);
LPDISPATCH pWorksheet = m_worksheets.GetItem(COleVariant((short) 1));

m_worksheet.AttachDispatch(pWorksheet);
m_worksheet.Select();

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A1")));
m_range[0].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A2")));
m_range[1].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A3")));
m_range[2].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A3"), COleVariant("C5")));
m_range[3].AttachDispatch(pRange);

VERIFY(pRange = m_worksheet.GetRange(COleVariant("A6")));
 m_range[4].AttachDispatch(pRange);

 m_range[4].SetValue(COleVariant(COleDateTime(2005, 4, 24, 15, 47, 8)));
 // Retrieve the stored date and print it as a string
 COleVariant vaTimeDate = m_range[4].GetValue();
 TRACE("returned date type = %d\n", vaTimeDate.vt);
 COleVariant vaTemp;
 vaTemp.ChangeType(VT_BSTR, &vaTimeDate);
 CString str = vaTemp.bstrVal;
 TRACE("date = %s\n", (const char*) str);

 m_range[0].SetValue(COleVariant("test string"));

 COleVariant vaResult0 = m_range[0].GetValue();
 if (vaResult0.vt == VT_BSTR) {
     CString str = vaResult0.bstrVal;
     TRACE("vaResult0 = %s\n", (const char*) str);
 }

 m_range[1].SetValue(COleVariant(3.14159));

 COleVariant vaResult1 = m_range[1].GetValue();
 if (vaResult1.vt == VT_R8) {
     TRACE("vaResult1 = %f\n", vaResult1.dblVal);
 }

 m_range[2].SetFormula(COleVariant("=$A2*2.0"));

 COleVariant vaResult2 = m_range[2].GetValue();
 if (vaResult2.vt == VT_R8) {
        TRACE("vaResult2 = %f\n", vaResult2.dblVal);
    }

    COleVariant vaResult2a = m_range[2].GetFormula();
    if (vaResult2a.vt == VT_BSTR) {
        CString str = vaResult2a.bstrVal;
        TRACE("vaResult2a = %s\n", (const char*) str);
    }

    m_range[3].FillRight();
    m_range[3].FillDown();

    // cleanup
    if (pWorkbook != NULL) {
        pWorkbook->Release();
    }
}
```

Continue on next module...

----------------------End Automation part 3---------------------------

Further reading and digging:

1. DCOM at MSDN.
2. COM+ at MSDN.
3. COM at MSDN.
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.