Automation part 4

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 MYMFC29E Automation Client Example

This program uses the new `#import` directive to generate smart pointers. It behaves just like MYMFC29D except that it doesn't run Excel. The `#import` statements are in the **StdAfx.h** file to minimize the number of times the compiler has to generate the driver classes.

The MYMFC29E Steps From Scratch

As usual start with AppWizard, just follow the shown steps and option settings. These monotonous steps provide a very good practice for you!

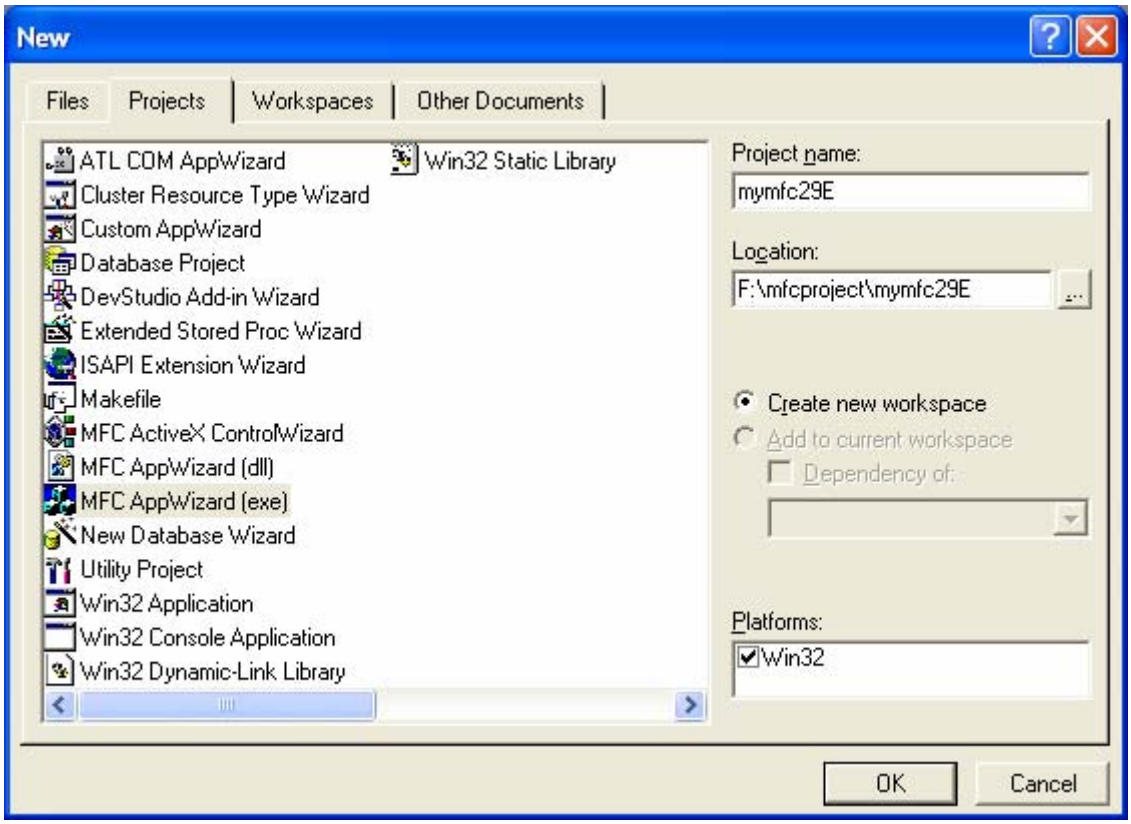

Figure 1: MYMFC29E – Visual C++ new project dialog.

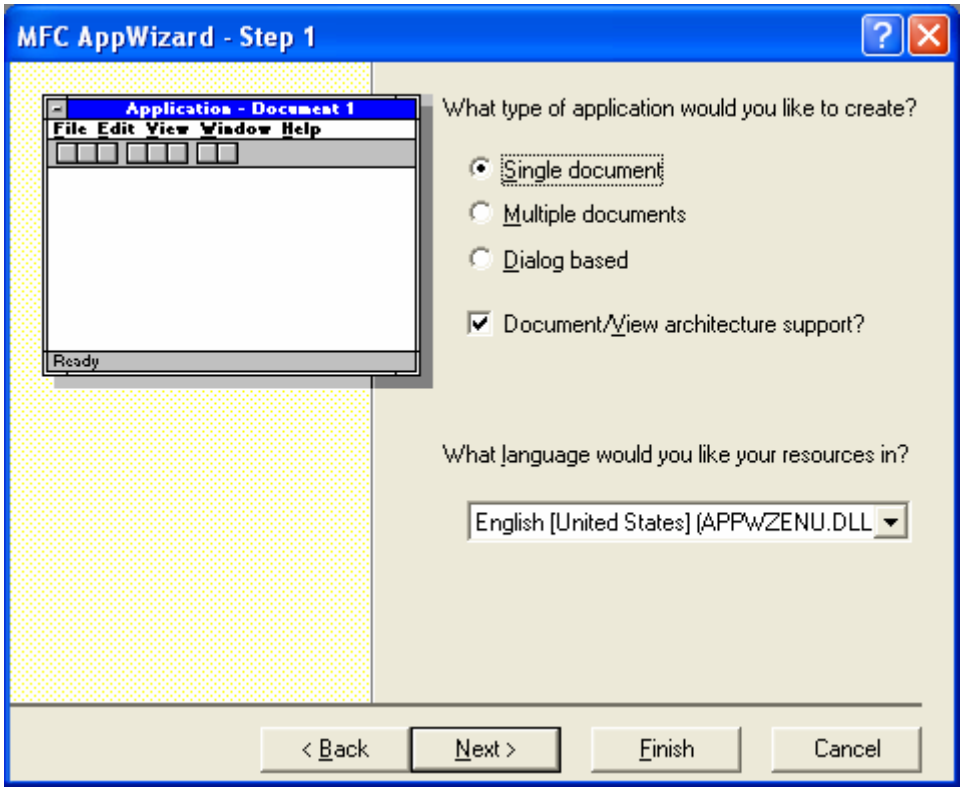

Figure 2: MYMFC29E – AppWizard step 1 of 6, an SDI application.

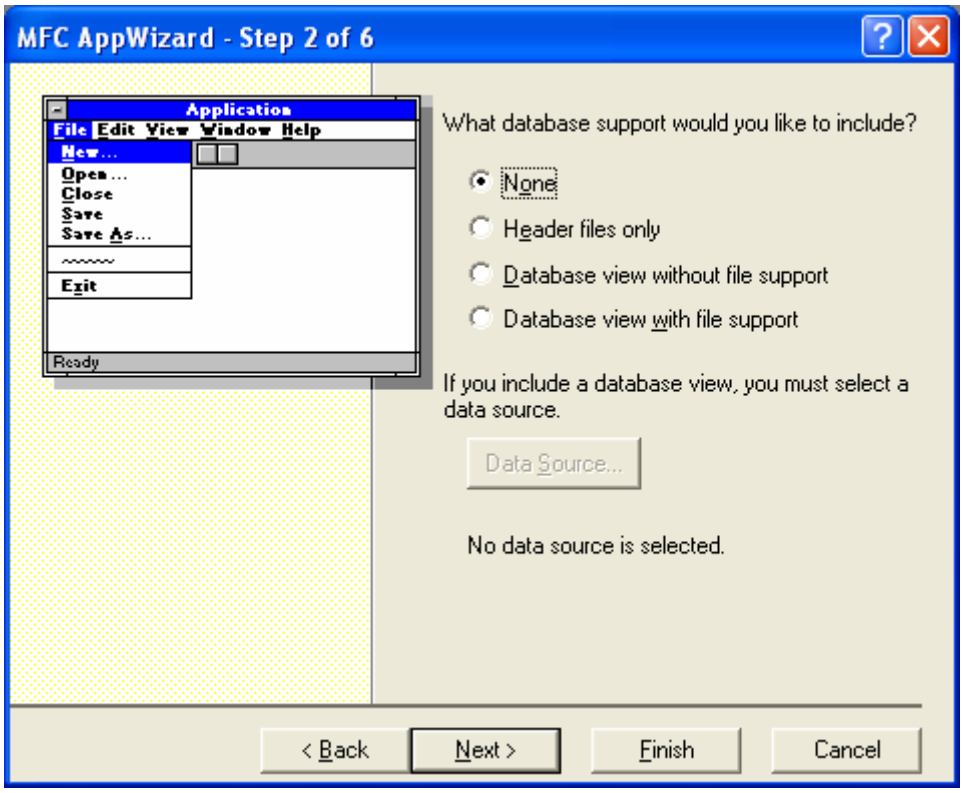

Figure 3: MYMFC29E – AppWizard step 2 of 6.

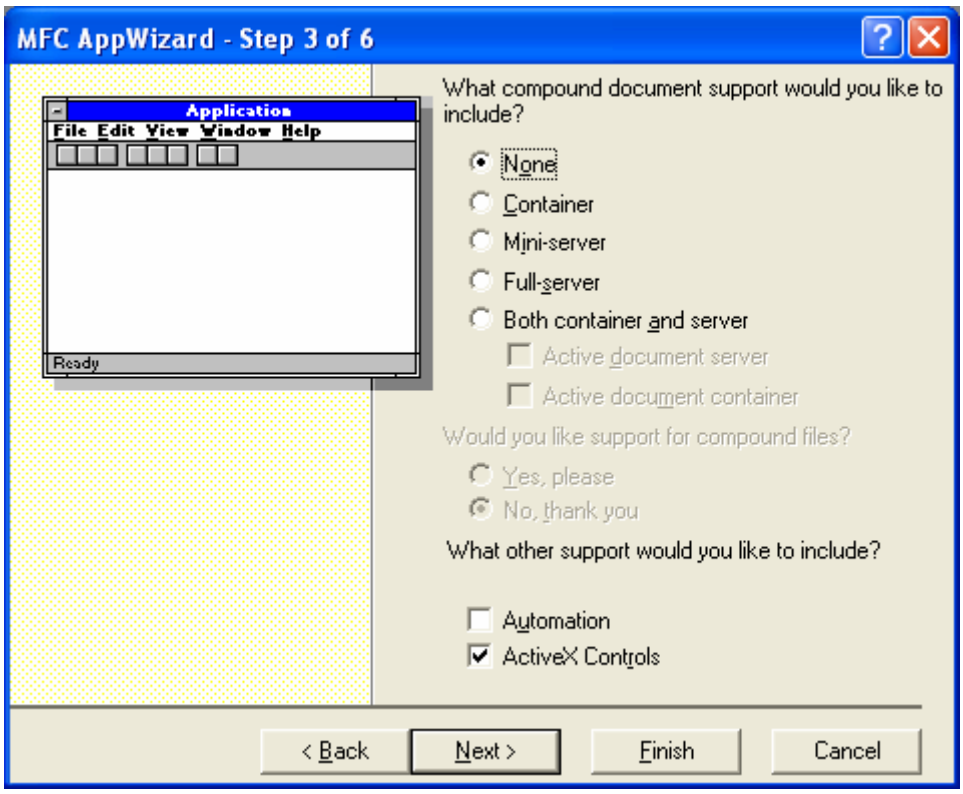

Figure 4: MYMFC29E – AppWizard step 3 of 6.

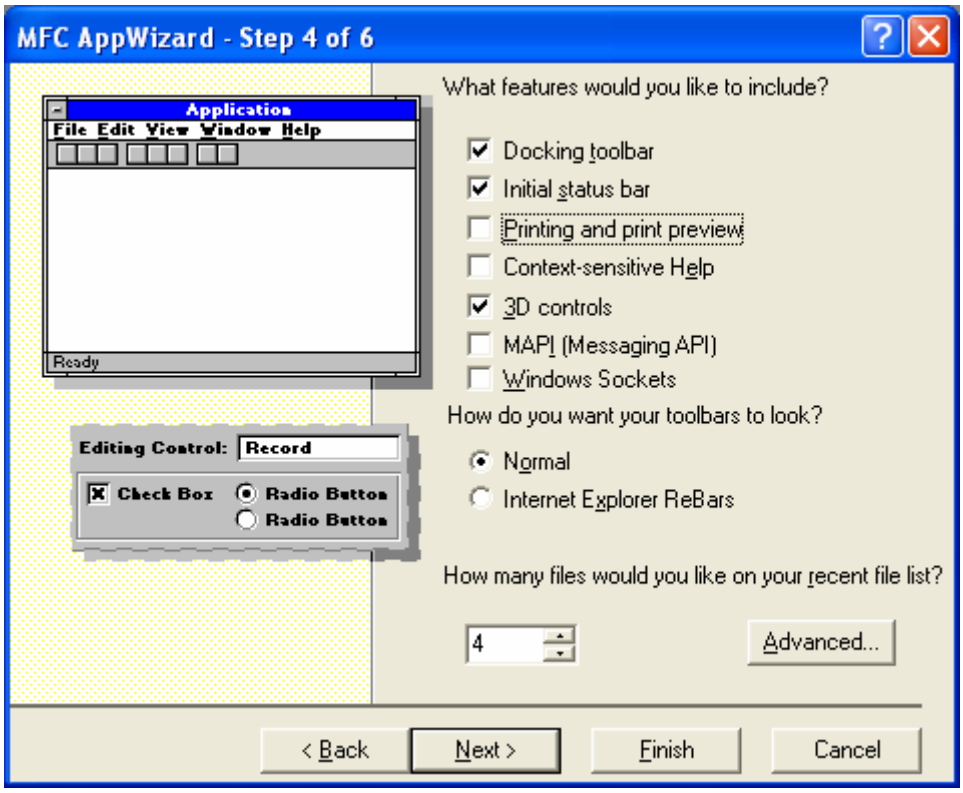

Figure 5: MYMFC29E – AppWizard step 4 of 6, deselecting the printing services.

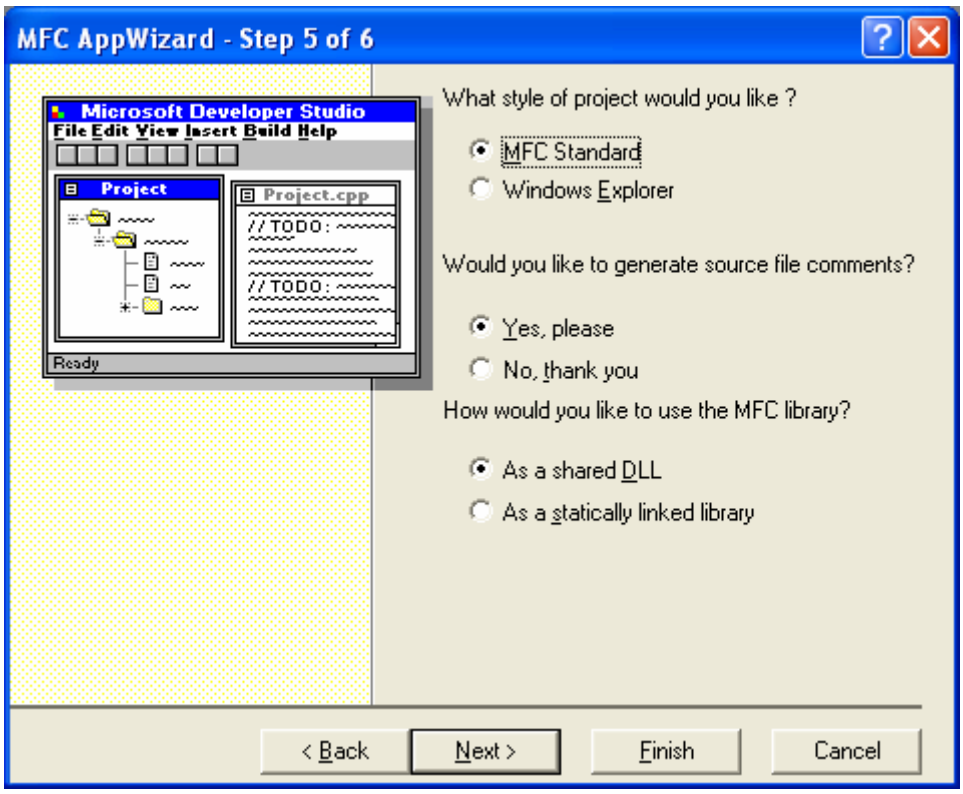

Figure 6: MYMFC29E – AppWizard step 5 of 6.

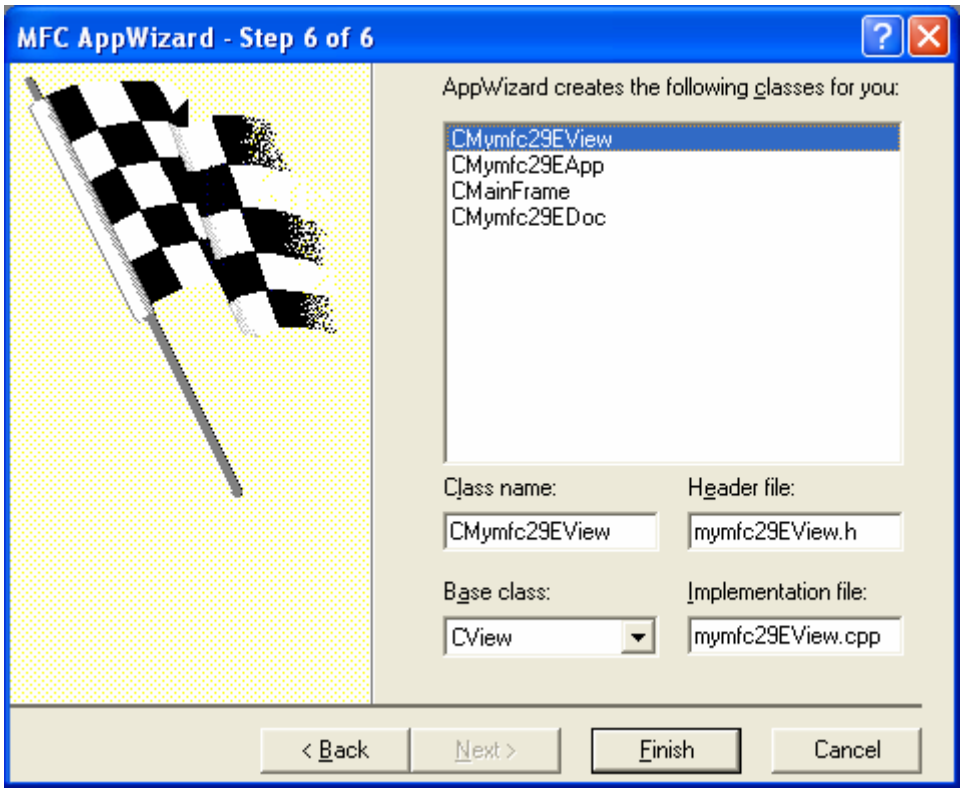

Figure 7: MYMFC29E – AppWizard step 6 of 6.

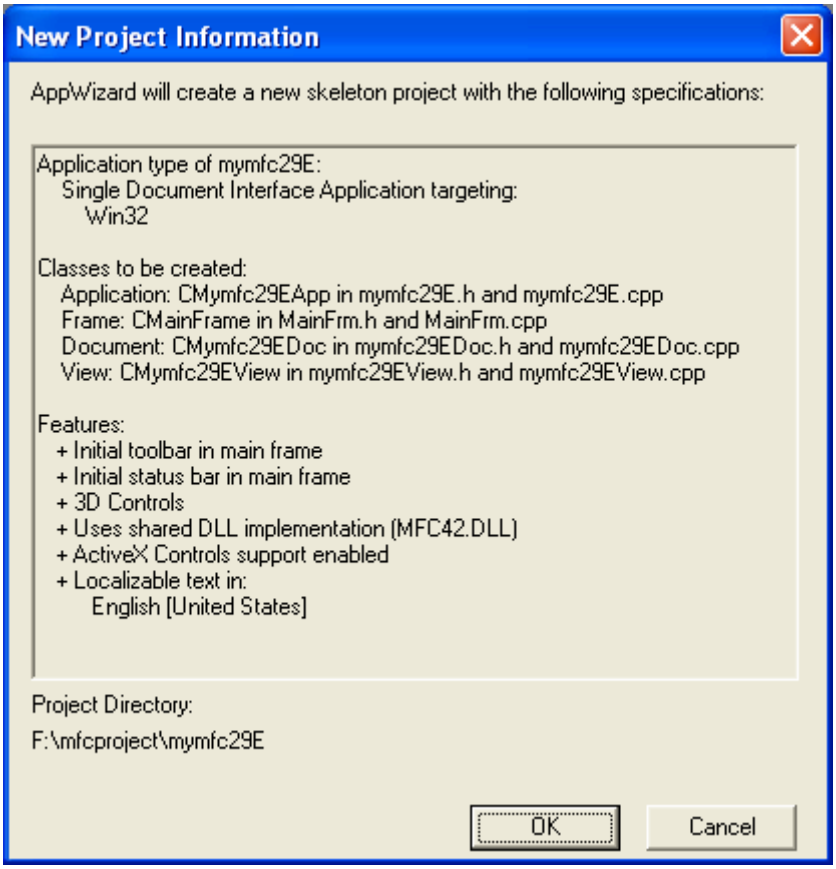

Figure 8: MYMFC29E project summary.

Add a new dialog. Insert new dialog and use the following information for the dialog, static text and edit controls. Leave the static texts IDs to the default.

ID	Item
IDD_ALARMDLG	Dialog with '**29E Alarm Dialog**' caption
IDC_HOURS	Hours edit box
IDC_MINUTES	Minutes edit box
IDC_SECONDS	Seconds edit box

Table 1.

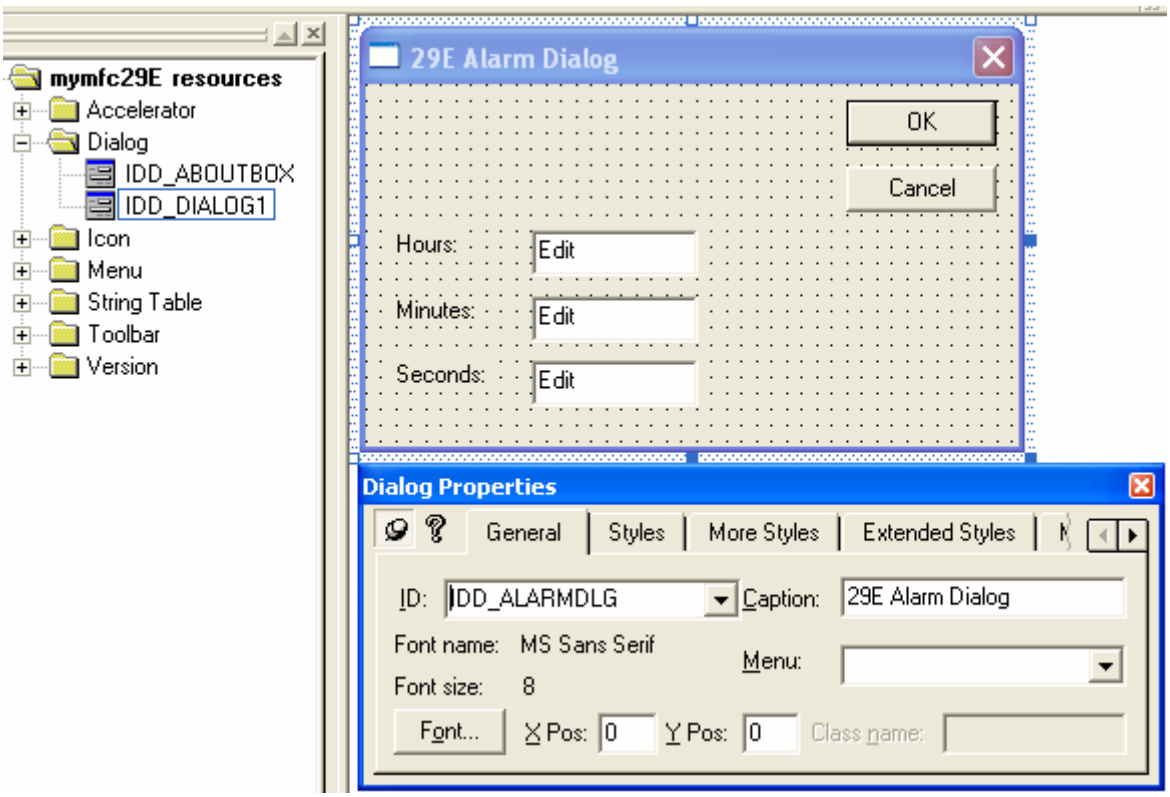

Figure 9: Adding new dialog and editing its property.

Next, use ClassWizard to create `CAlarmDialog`, a new class for the dialog.

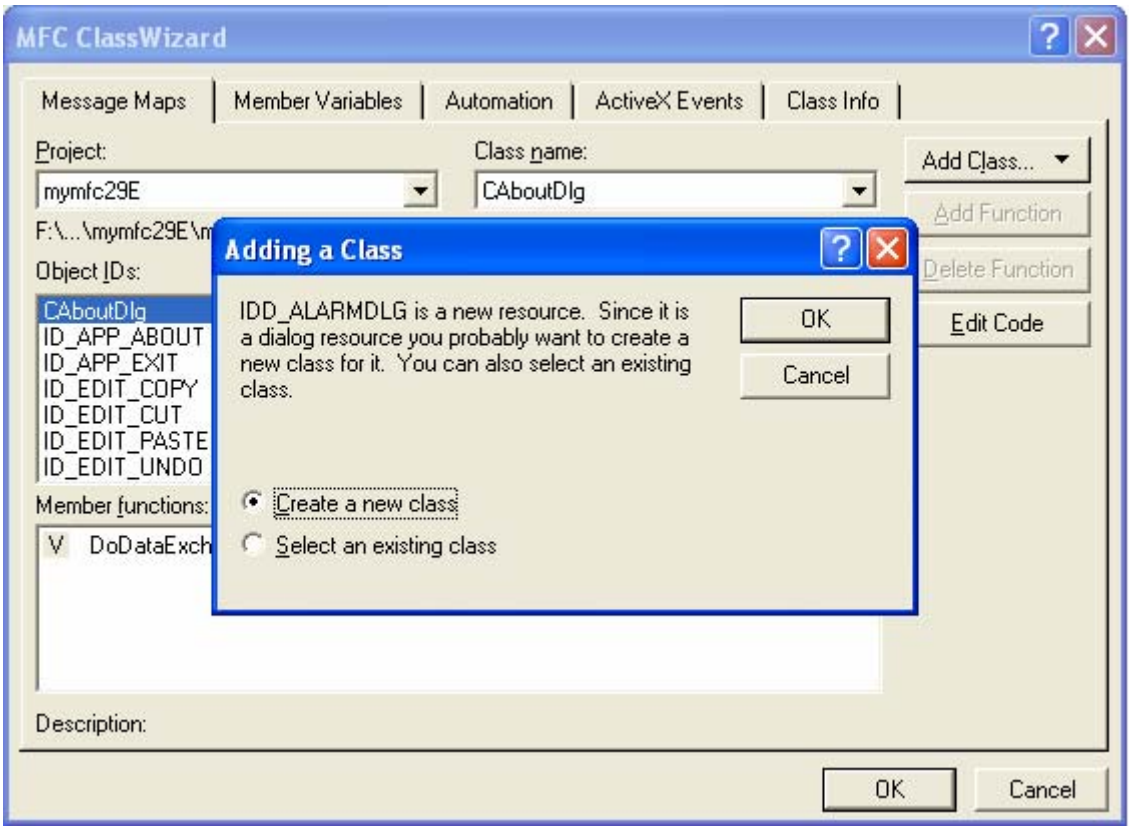

Figure 10: Adding new class, `CAlarmDialog` for the dialog.

Fill in the `CAlarmDialog` class information.

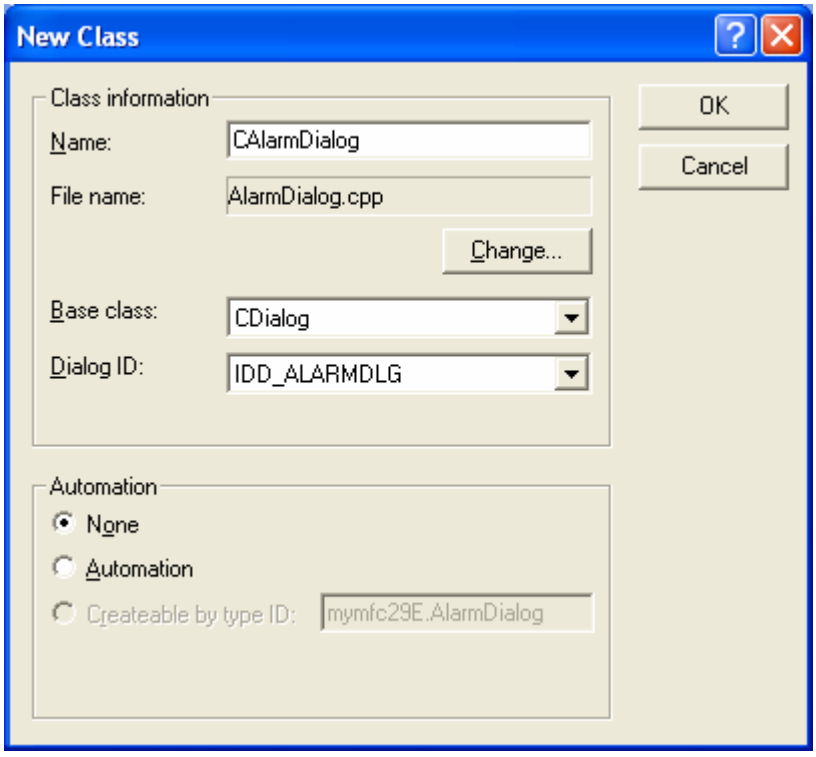

Figure 11: Entering the `CAlarmDialog` class information.

Add member variables and initialize them. Click the **Member Variables** tab and use the following information for the variables.

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 2.

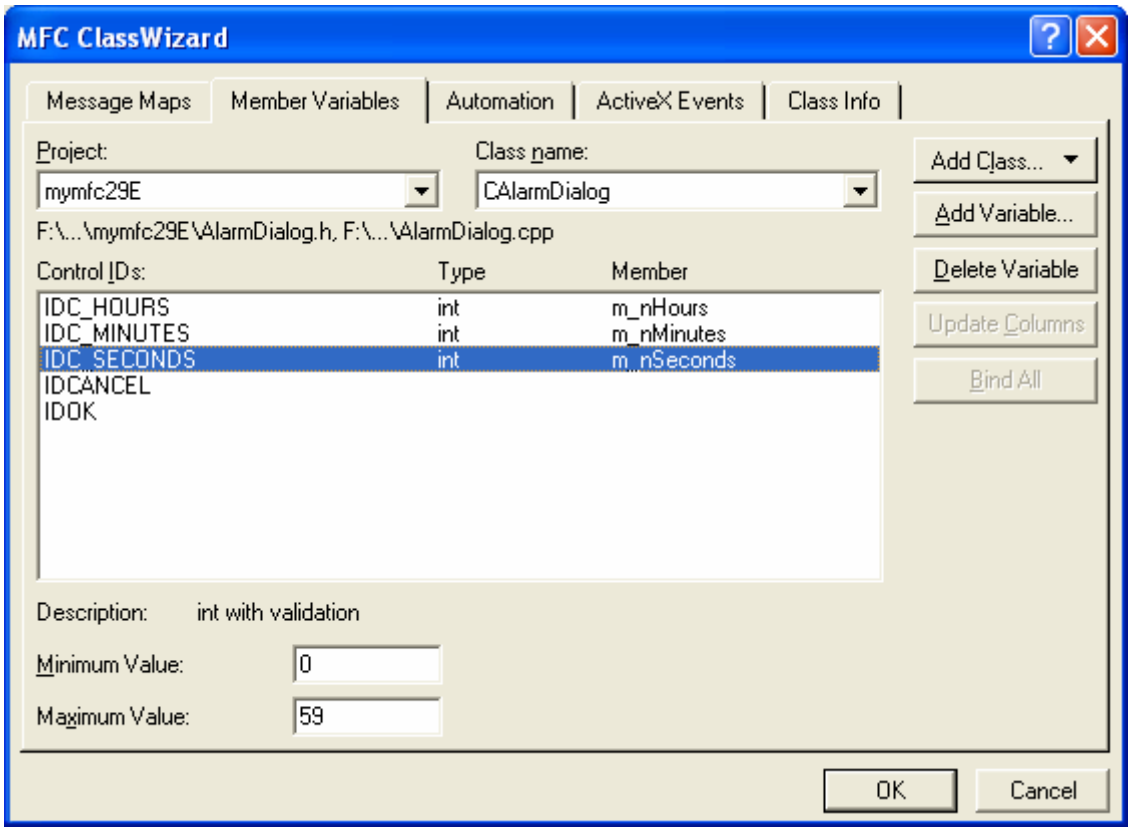

Figure 12: Adding member variables to controls in dialog.

Add menu items for **Bank Comp**.

ID	Menu	Prompt
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 3.

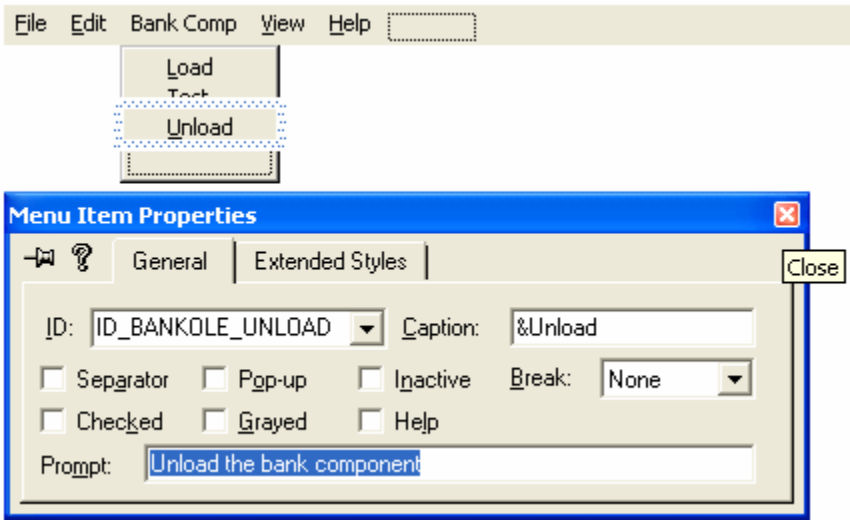

Figure 13: Adding **Bank Comp** menu and its' items.

Add menu items for **DLL Comp**.

ID	Menu	Prompt
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 4.

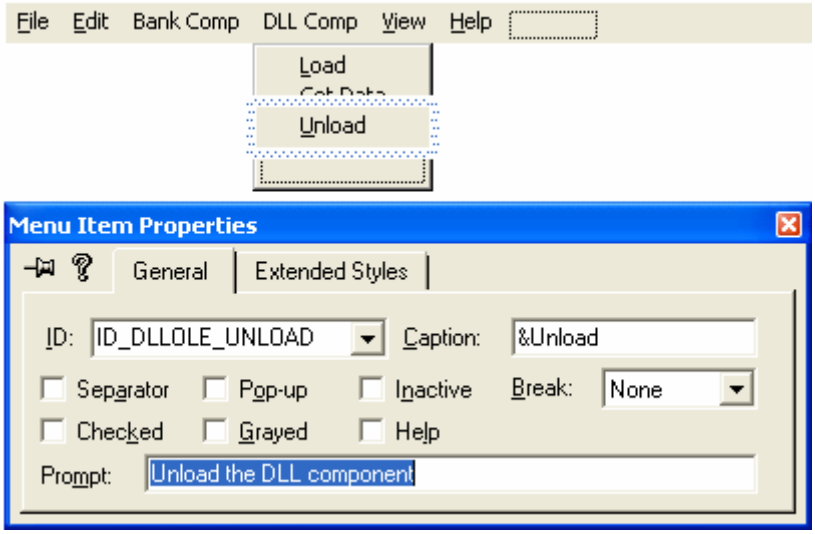

Figure 14: Adding **DLL Comp** menu and its' items.

Add menu items for **Clock Comp**.

ID	Menu	Prompt
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 5.

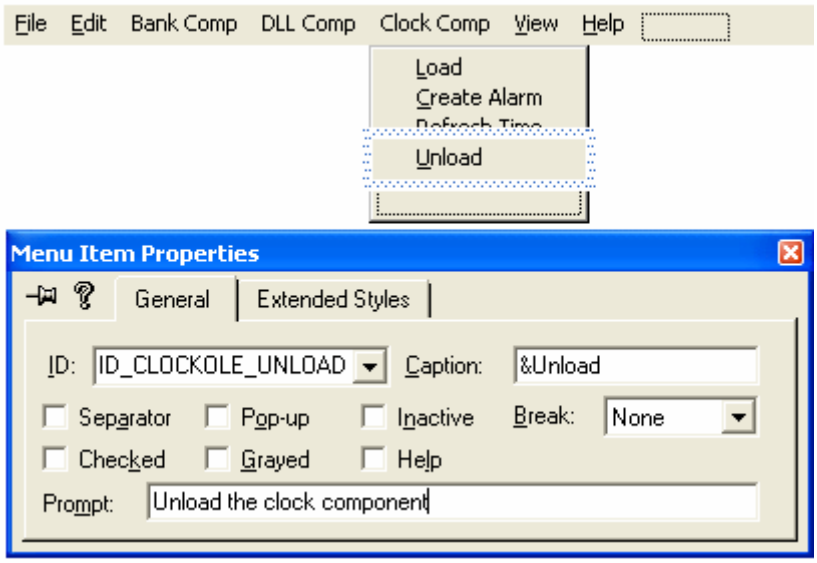

Figure 15: Adding **Clock Comp** menu and its' items.

Next, Using ClassWizard, add the menu **commands** and **update command UI** events in the `CMymfc29EView` 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 6.

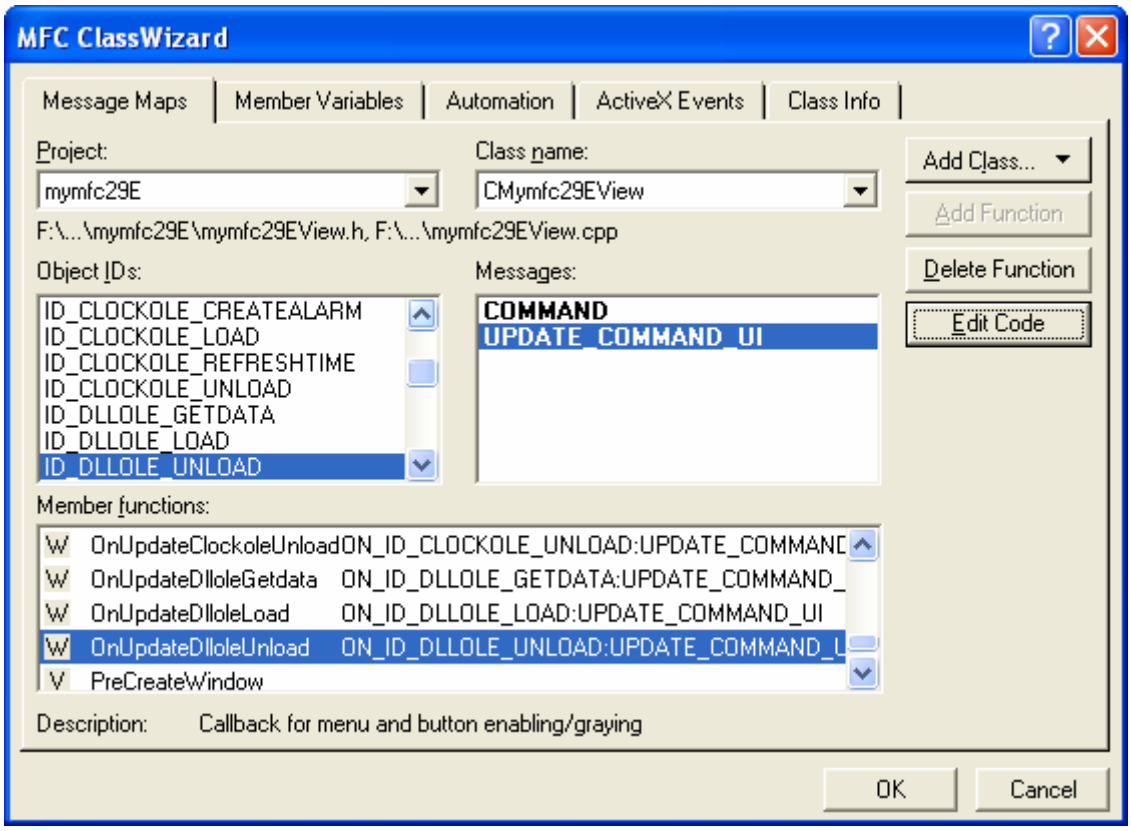

Figure 16: Adding menu Commands and Update Commands to `CMymfc29EView` class.

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();
```

```
BOOL CMymfc29EApp::InitInstance()
{

    AfxOleInit();

    AfxEnableControlContainer();

    // Standard initialization
```

Listing 2.

Back to **StdAfx.h** and add the following `#import` directives. you should change the path accordingly to suit your project directory. For Tenouk, all Visual C++ project is under **F:\mfcproject** directory.

```
#import "..\mymfc29A\debug\mymfc29A.tlb" rename_namespace("BankDriv")
using namespace BankDriv;

#import "..\mymfc29B\debug\mymfc29B.tlb" rename_namespace("Mymfc29BDriv")
using namespace Mymfc29BDriv;

#import "..\mymfc29C\debug\mymfc29C.tlb" rename_namespace("ClockDriv")
using namespace ClockDriv;
```

```
#include <afxdisp.h>        // Automation support

#import "..\mymfc29A\debug\mymfc29A.tlb" rename_namespace("BankDriv")
using namespace BankDriv;

#import "..\mymfc29B\debug\mymfc29B.tlb" rename_namespace("Mymfc29BDriv")
using namespace Mymfc29BDriv;

#import "..\mymfc29C\debug\mymfc29C.tlb" rename_namespace("ClockDriv")
using namespace ClockDriv;

//{{AFX_INSERT_LOCATION}}
```

Listing 3.

Add the following embedded smart pointers in the `CMymfc29EView` class header, **mymfc29EView.h**.

```
public:
      IMymfc29BAutoPtr m_auto;
      IBankPtr         m_bank;
      IMymfc29CPtr     m_clock;
      IAlarmPtr        m_alarm;
```

```
class CMymfc29EView : public CView
{
public:
    IMymfc29BAutoPtr m_auto;
    IBankPtr         m_bank;
    IMymfc29CPtr     m_clock;
    IAlarmPtr        m_alarm;

protected: // create from serializa
    CMymfc29EView();
    DECLARE_DYNCREATE(CMymfc29EView

// Attributes
```

Listing 4.

Add the `#include` directive for **AlarmDialog.h** in the `CMymfc29EView` class implementation, **mymfc29EView.cpp**. If you rebuild your ClassWizard database, this step is unnecessary.

```
#include "AlarmDialog.h"
```

```
#include "stdafx.h"
#include "mymfc29E.h"

#include "mymfc29EDoc.h"
#include "mymfc29EView.h"
#include "AlarmDialog.h"

#ifdef _DEBUG
```

Listing 5.

Add code for `CMymfc29DView::OnDraw()` member function in **mymfc29EView.cpp**.

```
void CMymfc29EView::OnDraw(CDC* pDC)
{
    CMymfc29EDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(100, 0, "--COleDispatchDriver usage, mymfc29E--");
    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...");
}
```

```
// CMymfc29EView drawing

void CMymfc29EView::OnDraw(CDC* pDC)
{
    CMymfc29EDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(100, 0, "--COleDispatchDriver usage, mymfc29E--");
    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 6.

Finally, add codes for the **commands** and **update command UI** events of the `CMymfc29EView` class.

```
void CMymfc29EView::OnBankoleLoad()
{
    if(m_bank.CreateInstance(__uuidof(Bank)) != S_OK)
    {
        AfxMessageBox("Bank component not found");
        return;
    }
      else
      { AfxMessageBox("OK, Bank component found lor!"); }
}

void CMymfc29EView::OnUpdateBankoleLoad(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_bank.GetInterfacePtr() == NULL); }

void CMymfc29EView::OnBankoleTest()
{
    try {
        m_bank->Deposit(20.0);
        m_bank->Withdrawal(15.0);
        TRACE("new balance = %f\n", m_bank->GetBalance());
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}
```

```
// CMymfc29EView message handlers

void CMymfc29EView::OnBankoleLoad()
{
    if(m_bank.CreateInstance(__uuidof(Bank)) != S_OK)
    {
        AfxMessageBox("Bank component not found");
        return;
    }
    else
    {
        AfxMessageBox("OK, Bank component found lor!");
    }
}

void CMymfc29EView::OnUpdateBankoleLoad(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bank.GetInterfacePtr() == NULL);
}

void CMymfc29EView::OnBankoleTest()
{
    try {
        m_bank->Deposit(20.0);
        m_bank->Withdrawal(15.0);
        TRACE("new balance = %f\n", m_bank->GetBalance());
    } catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());
    }
}
```

Listing 7.

```
void CMymfc29EView::OnUpdateBankoleTest(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_bank.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnBankoleUnload()
{ m_bank.Release(); }

void CMymfc29EView::OnUpdateBankoleUnload(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_bank.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnClockoleCreatealarm()
{
    CAlarmDialog dlg;
    try
    {
        if (dlg.DoModal() == IDOK)
        {
            COleDateTime dt(2004, 12, 23, dlg.m_nHours, dlg.m_nMinutes,
                dlg.m_nSeconds);
            LPDISPATCH pAlarm = m_clock->CreateAlarm(dt);
            m_alarm.Attach((IAlarm*) pAlarm);  // releases prior object!
            m_clock->RefreshWin();
        }
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}
```

```
void CMymfc29EView::OnUpdateBankoleTest(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bank.GetInterfacePtr() != NULL);
}

void CMymfc29EView::OnBankoleUnload()
{
    m_bank.Release();
}

void CMymfc29EView::OnUpdateBankoleUnload(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bank.GetInterfacePtr() != NULL);
}

void CMymfc29EView::OnClockoleCreatealarm()
{
    CAlarmDialog dlg;
    try
    {
        if (dlg.DoModal() == IDOK)
        {
            COleDateTime dt(2004, 12, 23, dlg.m_nHours, dlg.m_nMinutes,
                dlg.m_nSeconds);
            LPDISPATCH pAlarm = m_clock->CreateAlarm(dt);
            m_alarm.Attach((IAlarm*) pAlarm);  // releases prior object!
            m_clock->RefreshWin();
        }
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());
    }
}
```

Listing 8.

```
void CMymfc29EView::OnUpdateClockoleCreatealarm(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnClockoleLoad()
{
    if(m_clock.CreateInstance(__uuidof(Document)) != S_OK)
    {
        AfxMessageBox("Clock component not found");
        return;
    }
    try
    {
        m_clock->PutFigure(0, COleVariant("XII"));
        m_clock->PutFigure(1, COleVariant("III"));
        m_clock->PutFigure(2, COleVariant("VI"));
        m_clock->PutFigure(3, COleVariant("IX"));
        OnClockoleRefreshtime();
        m_clock->ShowWin();
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateClockoleLoad(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() == NULL); }
```

```
void CMymfc29EView::OnUpdateClockoleCreatealarm(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL);
}

void CMymfc29EView::OnClockoleLoad()
{
    if(m_clock.CreateInstance(__uuidof(Document)) != S_OK)
    {
        AfxMessageBox("Clock component not found");
        return;
    }
    try
    {
        m_clock->PutFigure(0, COleVariant("XII"));
        m_clock->PutFigure(1, COleVariant("III"));
        m_clock->PutFigure(2, COleVariant("VI"));
        m_clock->PutFigure(3, COleVariant("IX"));
        OnClockoleRefreshtime();
        m_clock->ShowWin();
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());
    }
}

void CMymfc29EView::OnUpdateClockoleLoad(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.GetInterfacePtr() == NULL);
}
```

Listing 9.

```
void CMymfc29EView::OnClockoleRefreshtime()
{
    COleDateTime now = COleDateTime::GetCurrentTime();
    try
    {
        m_clock->PutTime(now);
        m_clock->RefreshWin();
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateClockoleRefreshtime(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnClockoleUnload()
{ m_clock.Release(); }

void CMymfc29EView::OnUpdateClockoleUnload(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL); }
```

```
void CMymfc29EView::OnClockoleRefreshtime()
{
    COleDateTime now = COleDateTime::GetCurrentTime();
    try
    {
        m_clock->PutTime(now);
        m_clock->RefreshWin();
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());
    }
}

void CMymfc29EView::OnUpdateClockoleRefreshtime(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL);
}

void CMymfc29EView::OnClockoleUnload()
{
    m_clock.Release();
}

void CMymfc29EView::OnUpdateClockoleUnload(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL);
}
```

Listing 10.

```
void CMymfc29EView::OnDlloleGetdata()
{
    try {
        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);
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateDlloleGetdata(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_auto.GetInterfacePtr() != NULL); }
```

```
void CMymfc29EView::OnDlloleGetdata()
{
    try {
        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);
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());
    }
}

void CMymfc29EView::OnUpdateDlloleGetdata(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_auto.GetInterfacePtr() != NULL);
}
```

Listing 11.

```
void CMymfc29EView::OnDlloleLoad()
{
    if(m_auto.CreateInstance(__uuidof(Auto)) != S_OK)
    {
        AfxMessageBox("Mymfc29BAuto component not found");
        return;
    }
      else
      { AfxMessageBox("Mymfc29BAuto component found lol!"); }

}

void CMymfc29EView::OnUpdateDlloleLoad(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_auto.GetInterfacePtr() == NULL); }

void CMymfc29EView::OnDlloleUnload()
{ m_auto.Release(); }

void CMymfc29EView::OnUpdateDlloleUnload(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_auto.GetInterfacePtr() != NULL); }
```

```
void CMymfc29EView::OnDlloleLoad()
{
    if(m_auto.CreateInstance(__uuidof(Auto)) != S_OK)
    {
        AfxMessageBox("Mymfc29BAuto component not found");
        return;
    }
    else
    {
        AfxMessageBox("Mymfc29BAuto component found lol!");
    }

}

void CMymfc29EView::OnUpdateDlloleLoad(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_auto.GetInterfacePtr() == NULL);
}

void CMymfc29EView::OnDlloleUnload()
{
    m_auto.Release();
}

void CMymfc29EView::OnUpdateDlloleUnload(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_auto.GetInterfacePtr() != NULL);
}
```

Listing 12.

Check your ClassView for the added classes and the member functions.

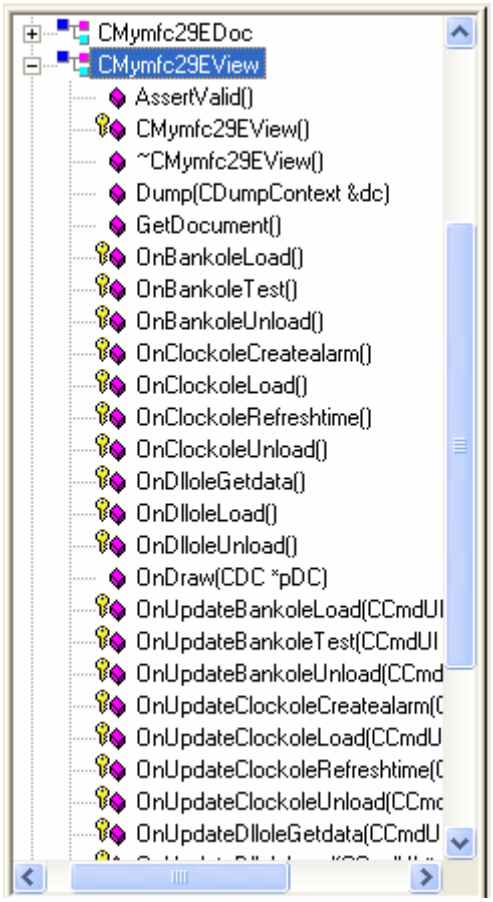

Figure 17: The added classes seen through ClassView.

Build and run MYMFC29E. Test all the components through the menu items. The following is the outputs for some of the menu items.

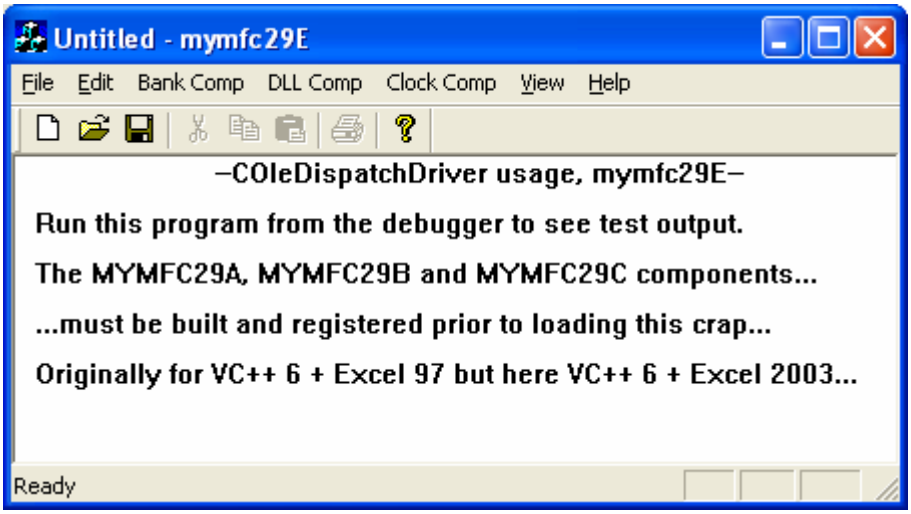

Figure 18: MYMFC29E in action.

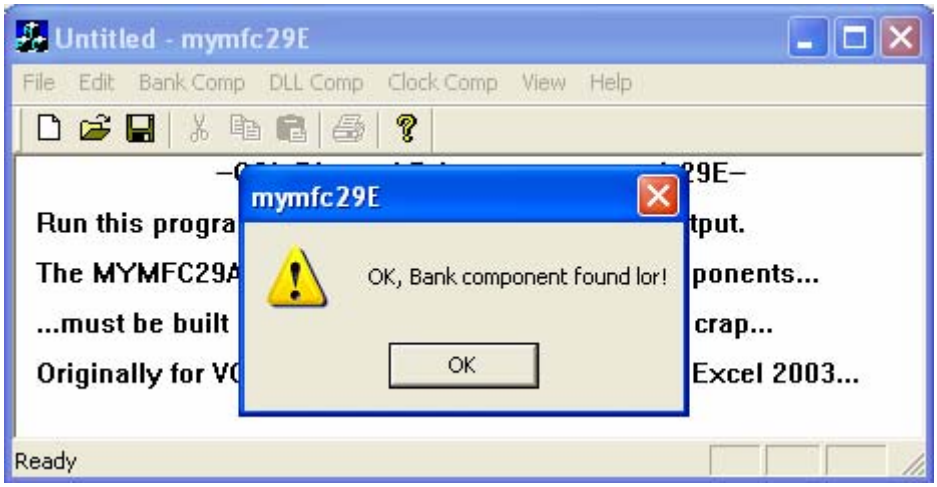

Figure 19: MYMFC29E output for the **Bank Comp Load** menu.

Figure 20: MYMFC29E output for the **DLL Comp Load** menu.

For the **DLL Comp Get Data**, you may encounter the Debug Assertion failed dialog cause by the ASSERT in the try-catch block. For this reason, just click the **Ignore** button. This should be tested through the Excel program (mymfc29B.xls).

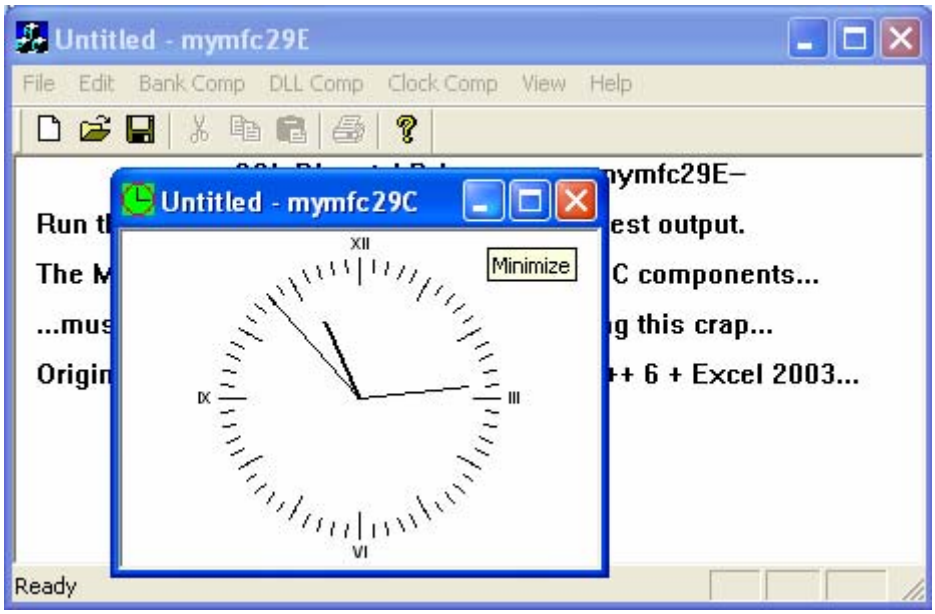

Figure 21: MYMFC29E output for the **Clock Comp Load** menu.

The story

The following are the added codes. As said before, you have to change the directory path accordingly to suit your program.

```
#include <afxdisp.h>      // Automation support

// You should change the path accordingly to suit yours...
// Tenouk's project directory is
// F:\mfcproject\all_tenouk_mfc_project_directories_are_here
#import "..\mymfc29A\debug\mymfc29A.tlb" rename_namespace("BankDriv")
using namespace BankDriv;

#import "..\mymfc29B\debug\mymfc29B.tlb" rename_namespace("Mymfc29BDriv")
using namespace Mymfc29BDriv;

#import "..\mymfc29C\debug\mymfc29C.tlb" rename_namespace("ClockDriv")
using namespace ClockDriv;
```

And of course you'll need to call `AfxOleInit()` in your application class `InitInstance()` member function. The view class header contains embedded smart pointers as shown:

```
	IMymfc29BAutoPtr m_auto;
	IBankPtr         m_bank;
	IMymfc29CPtr     m_clock;
	IAlarmPtr        m_alarm;
```

Here is the code for the view class menu command handlers:

```
////////////////////////////////////////////////////////////////////////////
// CMymfc29EView message handlers

void CMymfc29EView::OnBankoleLoad()
{
```

```
    if(m_bank.CreateInstance(__uuidof(Bank)) != S_OK)
    {
        AfxMessageBox("Bank component not found");
        return;
    }
      else
      { AfxMessageBox("OK, Bank component found lor!"); }
}

void CMymfc29EView::OnUpdateBankoleLoad(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_bank.GetInterfacePtr() == NULL); }

void CMymfc29EView::OnBankoleTest()
{
    try {
        m_bank->Deposit(20.0);
        m_bank->Withdrawal(15.0);
        TRACE("new balance = %f\n", m_bank->GetBalance());
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateBankoleTest(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_bank.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnBankoleUnload()
{ m_bank.Release(); }

void CMymfc29EView::OnUpdateBankoleUnload(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_bank.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnClockoleCreatealarm()
{
    CAlarmDialog dlg;
    try
    {
        if (dlg.DoModal() == IDOK)
        {
            COleDateTime dt(2004, 12, 23, dlg.m_nHours, dlg.m_nMinutes,
                dlg.m_nSeconds);
            LPDISPATCH pAlarm = m_clock->CreateAlarm(dt);
            m_alarm.Attach((IAlarm*) pAlarm);  // releases prior object!
            m_clock->RefreshWin();
        }
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateClockoleCreatealarm(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnClockoleLoad()
{
    if(m_clock.CreateInstance(__uuidof(Document)) != S_OK)
    {
        AfxMessageBox("Clock component not found");
        return;
    }
    try
```

```
    {
        m_clock->PutFigure(0, COleVariant("XII"));
        m_clock->PutFigure(1, COleVariant("III"));
        m_clock->PutFigure(2, COleVariant("VI"));
        m_clock->PutFigure(3, COleVariant("IX"));
        OnClockoleRefreshtime();
        m_clock->ShowWin();
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateClockoleLoad(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() == NULL); }

void CMymfc29EView::OnClockoleRefreshtime()
{
    COleDateTime now = COleDateTime::GetCurrentTime();
    try
    {
        m_clock->PutTime(now);
        m_clock->RefreshWin();
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateClockoleRefreshtime(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnClockoleUnload()
{ m_clock.Release(); }

void CMymfc29EView::OnUpdateClockoleUnload(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_clock.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnDlloleGetdata()
{
    try {
        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);
    }
    catch(_com_error& e)
    { AfxMessageBox(e.ErrorMessage()); }
}

void CMymfc29EView::OnUpdateDlloleGetdata(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_auto.GetInterfacePtr() != NULL); }

void CMymfc29EView::OnDlloleLoad()
{
    if(m_auto.CreateInstance(__uuidof(Auto)) != S_OK)
    {
        AfxMessageBox("Mymfc29BAuto component not found");
        return;
    }
```

```
    else
     { AfxMessageBox("Mymfc29BAuto component found lol!"); }
}

void CMymfc29EView::OnUpdateDlloleLoad(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_auto.GetInterfacePtr() == NULL); }

void CMymfc29EView::OnDlloleUnload()
{ m_auto.Release(); }

void CMymfc29EView::OnUpdateDlloleUnload(CCmdUI* pCmdUI)
{ pCmdUI->Enable(m_auto.GetInterfacePtr() != NULL); }
```

Note the use of the try/catch blocks (exception handling) in the functions that manipulate the components. These are particularly necessary for processing errors that occur when a component program stops running. In the previous example, MYMFC29D, the MFC `COleDispatchDriver` class took care of this detail.

VBA Early Binding

When you ran the MYMFC29A, MYMFC29B, and MYMFC29C components from Excel VBA, you used something called **late binding**. Normally, each time VBA accesses a property or a method, it calls `IDispatch::GetIDsOfNames` to look up the dispatch ID from the symbolic name. Not only is this inefficient, VBA can't do type-checking until it actually accesses a property or a method. Suppose, for example, that a VBA program tried to get a property value that it assumed was a number, but the component provided a string instead. VBA would give you a runtime error when it executed the **Property Get** statement.
With early binding, VBA can preprocess the Visual Basic code, converting property and method symbols to `DISPID`s before it runs the component program. In so doing, it can check property types, method return types, and method parameters, giving you compile-time error messages. Where can VBA get the advance information it needs? From the component's type library, of course. It can use that same type library to allow the VBA programmer to browse the component's properties and methods. VBA reads the type library before it even loads the component program.

Registering a Type Library

You've already seen that Visual C++ generates a TLB file (type library file) for each component. For VBA to locate that type library, its location must be specified in the Windows Registry. The simplest way of doing this is to write a text **REG** file that the Windows **Regedit**/**Regedit32** program can import. Here's the **mymfc29B.reg** file example for the MYMFC29B DLL component:

```
REGEDIT4

[HKEY_CLASSES_ROOT\TypeLib\{A9515ACA-5B85-11D0-848F-00400526305B}]

[HKEY_CLASSES_ROOT\TypeLib\{A9515ACA-5B85-11D0-848F-00400526305B}\1.0]
@="Mymfc29B"

[HKEY_CLASSES_ROOT\TypeLib\{A9515ACA-5B85-11D0-848F-00400526305B}\1.0\0]

[HKEY_CLASSES_ROOT\TypeLib\{A9515ACA-5B85-11D0-848F-00400526305B}\1.0\0\win32]
@="F:\\mfcproject\\mymfc29B\\Debug\\mymfc29B.tlb"

[HKEY_CLASSES_ROOT\TypeLib\{A9515ACA-5B85-11D0-848F-00400526305B}\1.0\FLAGS]
@="0"

[HKEY_CLASSES_ROOT\TypeLib\{A9515ACA-5B85-11D0-848F-00400526305B}\1.0\HELPDIR]
@="F:\\mfcproject\\mymfc29B\\Debug"

[HKEY_CLASSES_ROOT\Interface\{A9515AD7-5B85-11D0-848F-00400526305B}]
@="IMymfc29BAuto"

[HKEY_CLASSES_ROOT\Interface\{A9515AD7-5B85-11D0-848F-00400526305B}\ProxyStubClsid]
@="{00020420-0000-0000-C000-000000000046}"
```

```
[HKEY_CLASSES_ROOT\Interface\{A9515AD7-5B85-11D0-848F-00400526305B}\ProxyStubClsid32]
@="{00020420-0000-0000-C000-000000000046}"

[HKEY_CLASSES_ROOT\Interface\{A9515AD7-5B85-11D0-848F-00400526305B}\TypeLib]
@="{A9515ACA-5B85-11D0-848F-00400526305B}"
"Version"="1.0"
```

Notice that this file generates subtrees under the Registry's **TypeLib** and **Interface** keys. The third entry specifies the path for the version 1.0 TLB file. The format is shown below.

```
[Entry #1 – TypeLib]
[Entry #2 – Interface]
[Entry #3 – version]
```

The `0` subkey stands for "neutral language." If you had a multilingual application, you would have separate entries for English, French, and so forth. Browsers use the **TypeLib** entries, and the **Interface** entries are used for runtime type-checking and, for an EXE component, marshaling the dispinterface.

How a Component Can Register Its Own Type Library

When an EXE component is run stand-alone, it can call the MFC `AfxRegisterTypeLib()` function to make the necessary Registry entries, as shown here:

```
VERIFY(AfxOleRegisterTypeLib(AfxGetInstanceHandle(), theTypeLibGUID,
"mymfc29B.tlb"));
```

Shown here is `theTypeLibGUID`, a static variable of type GUID:

```
// {A9515ACA-5B85-11D0-848F-00400526305B}
static const GUID theTypeLibGUID =
{ 0xa9515aca, 0x5b85, 0x11d0, { 0x84, 0x8f, 0x00, 0x40, 0x05, 0x26, 0x30, 0x5b
} };
```

The `AfxRegisterTypeLib()` function is declared in the **afxwin.h** header, which requires `_AFXDLL` to be defined. That means you can't use it in a regular DLL unless you copy the code from the MFC source files.

The ODL File

Now is a good time to look at the ODL file for the same project.

```
// mymfc29B.odl : type library source for mymfc29B.dll

// This file will be processed by the MIDL compiler to produce the
// type library (mymfc29B.tlb).

[ uuid(4C646167-C06E-43AF-927B-E5B1BA2BAC58), version(1.0) ]
library Mymfc29B
{
	importlib("stdole32.tlb");
	importlib("stdole2.tlb");

	// Primary dispatch interface for CMymfc29BAuto

	[ uuid(7A97BA38-BF4A-4586-93C6-72B5EE7E0DC2) ]
	dispinterface IMymfc29BAuto
	{
		properties:
			// NOTE - ClassWizard will maintain property information here.
			//    Use extreme caution when editing this section.
			//{{AFX_ODL_PROP(CMymfc29BAuto)
			[id(1)] long LongData;
			[id(2)] VARIANT TextData;
			//}}AFX_ODL_PROP
```

```
        methods:
                // NOTE - ClassWizard will maintain method information here.
                //    Use extreme caution when editing this section.
                //{{AFX_ODL_METHOD(CMymfc29BAuto)
                [id(3)] boolean DisplayDialog();
                //}}AFX_ODL_METHOD

        };

        // Class information for CMymfc29BAuto

        [ uuid(39D9E31F-25CB-4511-B5A8-5406E29BA565) ]
        coclass Auto
        {
                [default] dispinterface IMymfc29BAuto;
        };

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

As you can see, numerous connections exist among the Registry, the type library, the component, and the VBA client.
A useful Visual C++ utility, **OLEVIEW**, lets you examine registered components and their type libraries.

How Excel Uses a Type Library

Let's examine the sequence of steps Excel uses to utilize your type library:

1. When Excel starts up, it reads the **TypeLib** section of the Registry to compile a list of all type libraries. It loads the type libraries for **VBA** and for the **Excel object** library.
2. After starting Excel, loading a workbook, and switching to the Visual Basic Editor, the user (or workbook author) chooses **References** from the **Tools** menu and checks the MYMFC29B LIB line (If your library is not listed, you can use the browse button to include those library). When the workbook is saved, this reference information is saved with it.

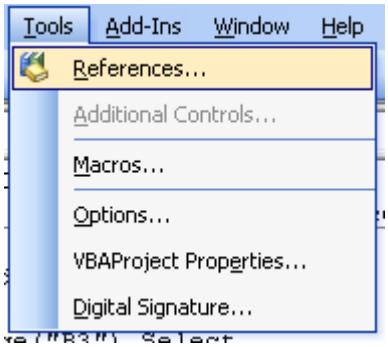

Figure 22: Invoking the **References** of the Excel type library in Visual Basic editor.

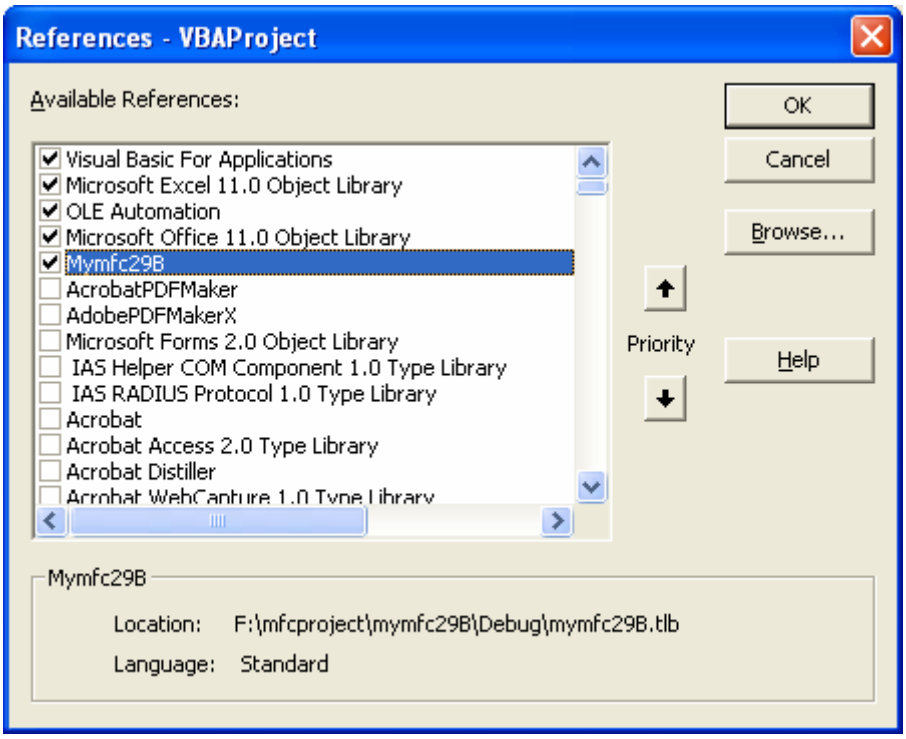

Figure 23: Adding/removing the Excel type library.

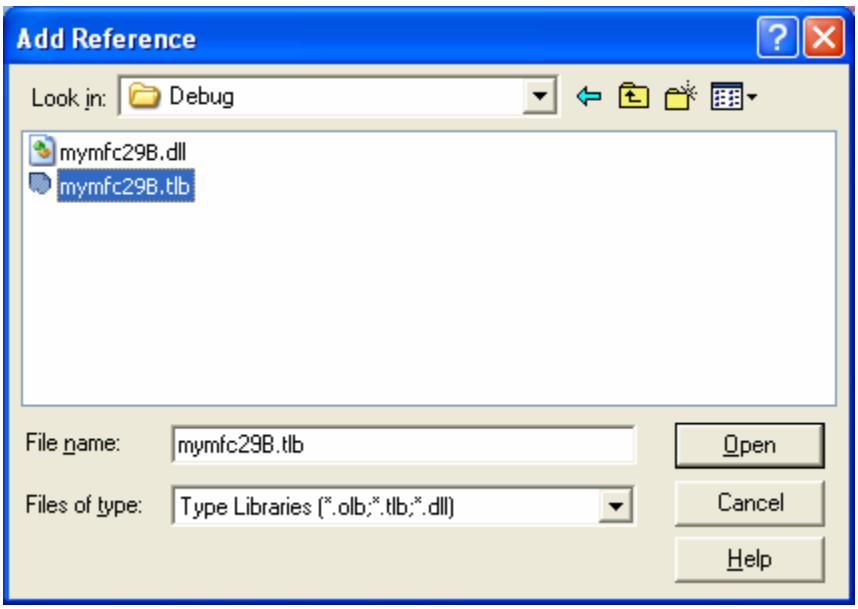

Figure 24: Adding new reference of the type library by clicking the **Browse** button in previous Figure.

3. Now the Excel user will be able to browse through the MYMFC29B properties and methods by choosing **Object Browser** from the Visual Basic Editor's **View** menu to view the Object Browser dialog.
4. To make use of the type library in your VBA program, you simply replace the line:

```
Dim DllComp as Object
```

with

```
Dim DllComp as IMymfc29BAuto
```

The VBA program will exit immediately if it can't find `IMymfc29BAuto` in its list of references.

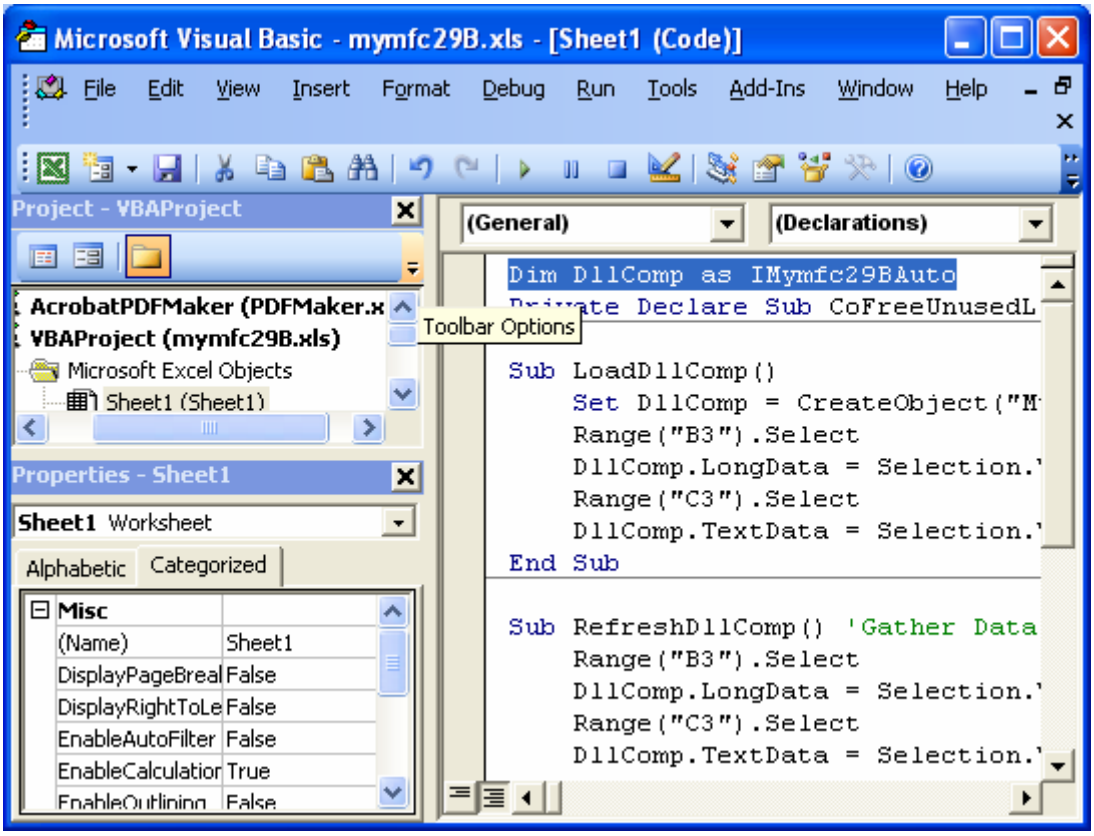

Figure 1: Visual Basic editor, changing some codes.

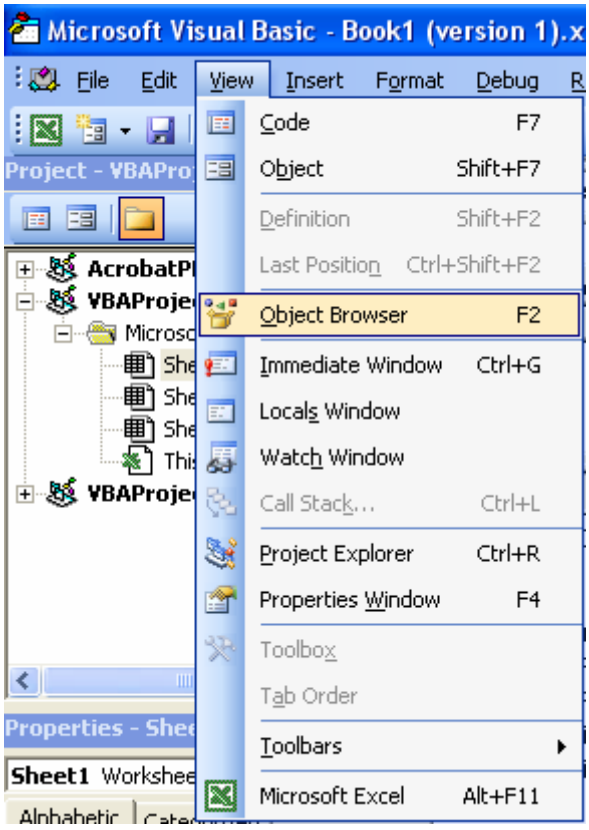

Figure 1: Invoking an Object Browser.

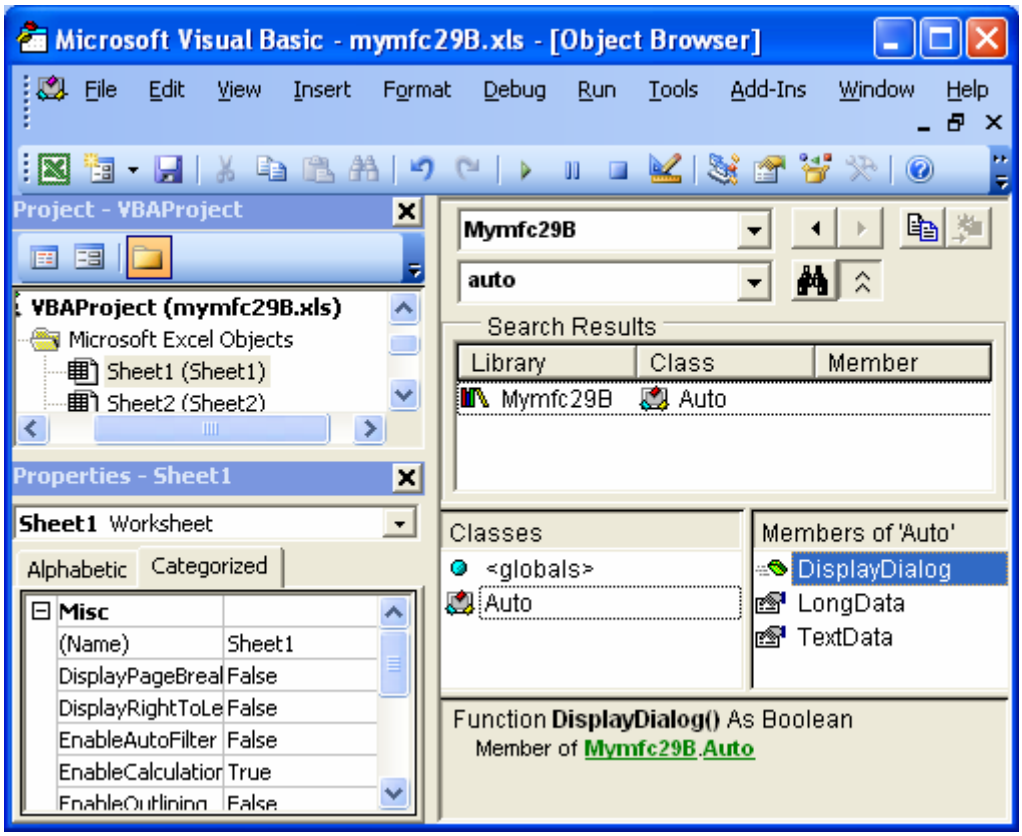

Figure 1: Object Browser window, browsing objects.

5. Immediately after VBA executes the `CreateObject` statement and loads the component program, it calls `QueryInterface()` for `IID_IMymfc29BAuto`, which is defined in the Registry, the type library, and the component class's interface map. `IMymfc29BAuto` is really an `IDispatch` interface. This is a sort of security check. If the component can't deliver this interface, the VBA program exits. Theoretically, Excel could use the `CLSID` in the type library to load the component program, but it uses the `CLSID` from the Registry instead, just as it did in late binding mode.

Why Use Early Binding?

You might think that early binding would make your Automation component run faster. You probably won't notice any speed increase, though, because the `IDispatch::Invoke` calls are the limiting factor. A typical MFC Invoke call from a compiled C++ client to a compiled C++ component requires about 0.5 millisecond, which is pretty gross.
The browse capability that the type library provides is probably more valuable than the compiled linkage. If you are writing a C++ controller, for example, you can load the type library through various COM functions, including `LoadTypeLib()`, and then you can access it through the `ITypeLib` and `ITypeInfo` interfaces. Plan to spend some time on that project, however, because the type library interfaces are tricky.

Faster Client-Component Connections

Microsoft has recognized the limitations of the `IDispatch` interface. It's naturally slow because all data must be funneled through `VARIANT`s and possibly converted on both ends. There's a new variation called a **dual interface**. (A discussion of dual interfaces is beyond the scope of this book. See Kraig Brockschmidt's Inside OLE, 2d ed. [Microsoft Press, 1995], for more information.) In a dual interface, you define your own custom interface, derived from `IDispatch`. The `Invoke()` and `GetIDsOfNames()` functions are included, but so are other functions. If the client is smart enough, it can bypass the inefficient `Invoke()` calls and use the specialized functions instead. Dual interfaces can support only standard Automation types, or they can support arbitrary types.
There is no direct MFC support for dual interfaces in Visual C++ 6.0, but the MSDN's ACDUAL Visual C++ sample should get you started.

----------End Automation part 4----------

Further reading and digging:

1. MSDN MFC 6.0 class library online documentation - used throughout this Tutorial.
2. MSDN MFC 7.0 class library online documentation - used in .Net framework and also backward compatible with 6.0 class library
3. DCOM at MSDN.
4. COM+ at MSDN.
5. COM at MSDN.
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.