Automation part 2

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 mymfc29C.xls, automation, variant and COlevariant class.

Index:

The MYMFC29C SDI Automation Component - EXE Example: with User Interface
MYMFC29C From Scratch
`CMymfc29CDoc` class
`CMainFrame` class
`CMymfc29CView` class
The Story

The MYMFC29C SDI Automation Component EXE Example: with User Interface

This Automation component example illustrates the use of a document component class in an **SDI application** in which a new process is started for each object. This component program demonstrates an indexed property plus a method that constructs a new COM object.
The first Automation component example you saw, MYMFC29A, didn't have a user interface. The global class factory constructed a `CBank` object that did the component's work. What if you want your EXE component to have a window? If you've bought into the MFC document-view architecture, you'll want the document, view, and frame, with all the benefits they provide.
Suppose you created a regular MFC application and then added a COM-creatable class such as `CBank`. How do you attach the `CBank` object to the document and view? From a `CBank` class member function, you could navigate through the application object and main frame to the current document or view, but you'd have a tough time in an MDI application if you encountered several component objects and several documents. There is a better way. You make the document class the **creatable class**, and you have the full support of AppWizard for this task. This is true for both MDI and SDI applications.
The MDI MSDN's Autoclik example demonstrates how COM triggers the construction of new document, view, and child frame objects each time an Automation client creates a new component object. Because the MYMFC29C example is an SDI program, Windows starts a new process each time the client creates an object. Immediately after the program starts, COM, with the help of the MFC application framework, constructs not only the Automation-aware document but also the view and the main frame window.
Now is a good time to experiment with the MYMFC29C application, which was first generated by AppWizard with the **Automation** option checked. It's a Windows-based alarm clock program designed to be manipulated from an Automation client such as Excel. MYMFC29C has the following **properties** and **methods**.

Method/property	Description
`Time`	`DATE` property that holds a COM `DATE` (`m_Time`)
`Figure`	Indexed `VARIANT` property for the four figures on the clock face (`m_strFigure[]`)
`RefreshWin`	Method that invalidates the view window and brings the main frame window to the top (`Refresh`)
`ShowWin`	Method that displays the application's main window (`ShowWin`)
`CreateAlarm`	Method that creates a `CAlarm` object and returns its `IDispatch` pointer (`CreateAlarm`)

Table 1.

Here are the steps for building and running an already completed MYMFC29C.

From Visual C++, open the workspace mymfc29C.dsw or any directory that you have designated for your project. Build the project to produce the **mymfc29C.exe** file in the project's Debug subdirectory.

Run the program once to register it. The program is designed to be executed either as a **standalone application** or as an **Automation component**. When you run it from Windows or from Visual C++, it updates the Registry automatically and displays the face of a clock with the characters `XII`, `III`, `VI`, and `IX` at the 12, 3, 6, and 9 o'clock positions. Exit the program.

Next, load the mymfc29C.xls Excel workbook file. The worksheet should look something like the one shown here.

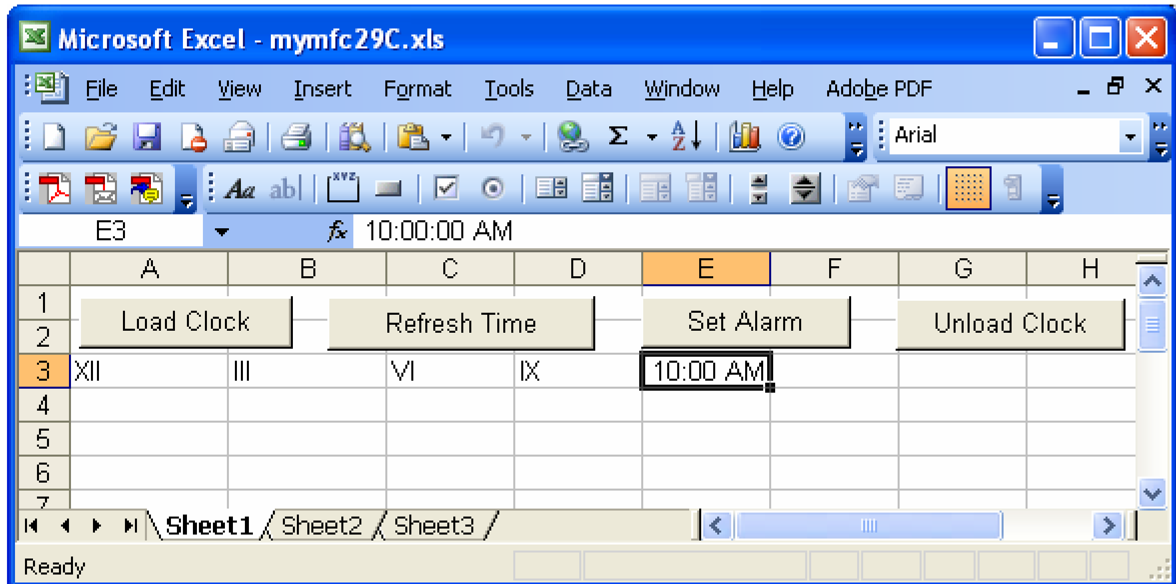

Figure 1: **mymfc29C.xls**, a client, to test MYMFC29C component.

Click the **Load Clock** button, and then double-click the **Set Alarm** button. There could be a long delay after you click the **Load Clock** button, depending on your system. The clock should appear as shown below, with the letter **A** indicating the alarm setting.

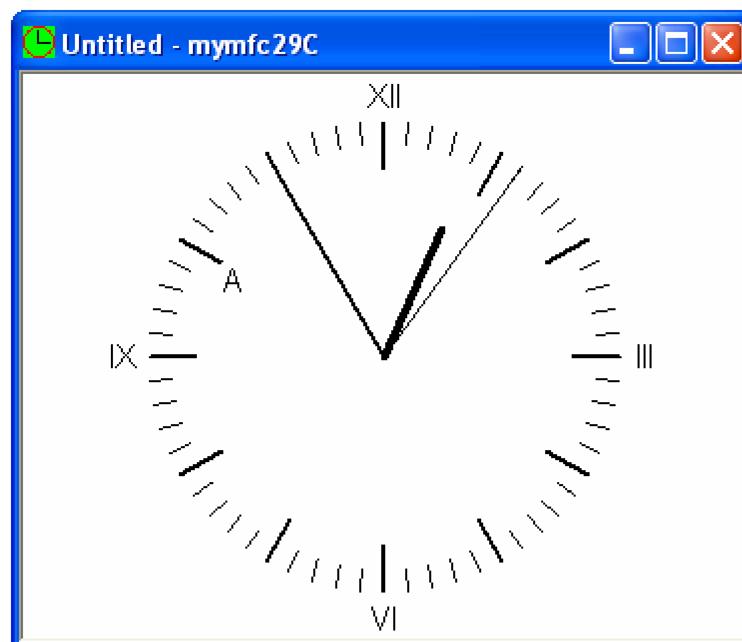

Figure 2: Alarm is set through a client.

If you've started the component program from the debugger, you can watch the Debug window to see when `InitInstance()` is called and when the document object is constructed.

MYMFC29C From Scratch

To experience a real programming, the following are steps to build MYMFC29C from scratch. Have a good practice! This is an SDI application with Automation support.

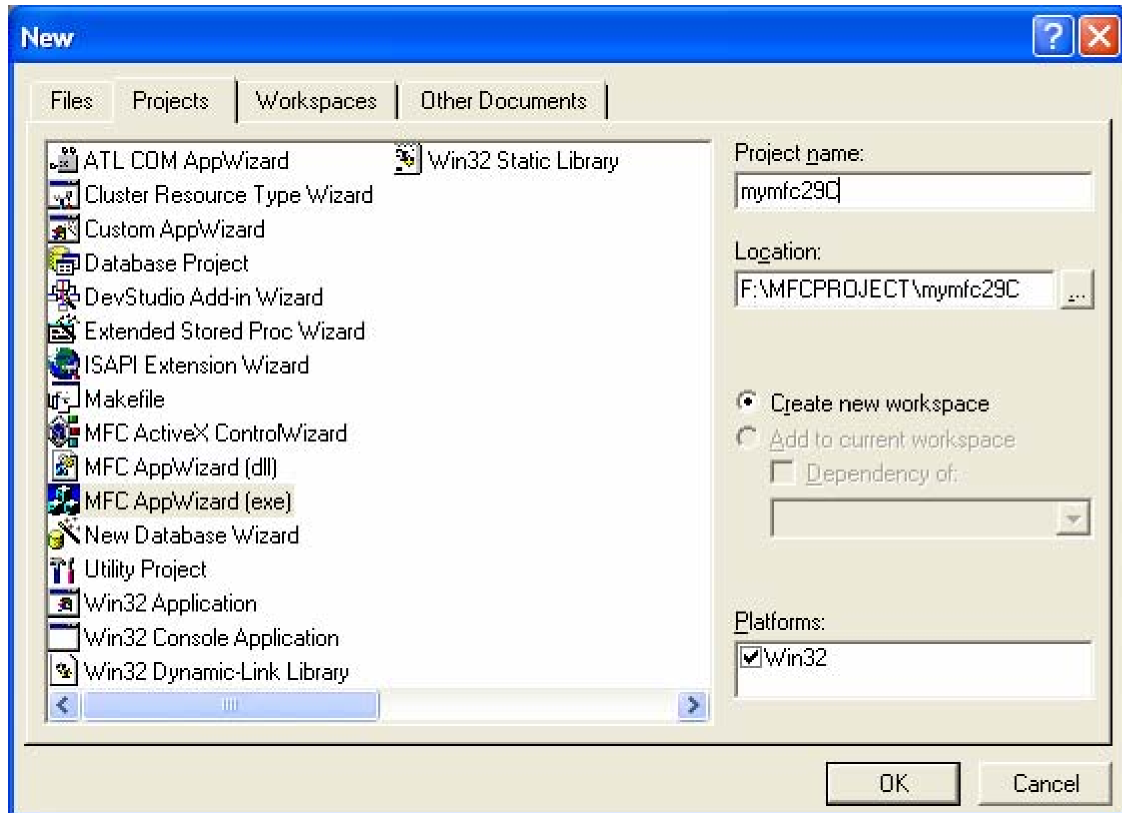

Figure 3: Visual C++ new project dialog.

Select the **Single Document**.

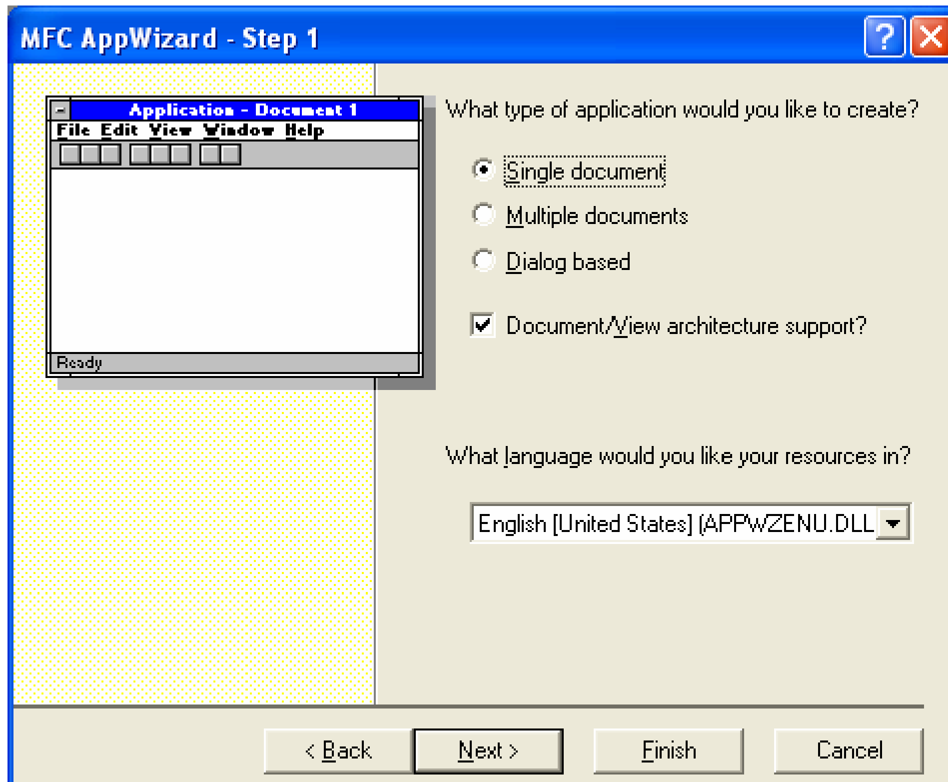

Figure 4: MYMFC29C – AppWizard step 1 of 6, SDI application.

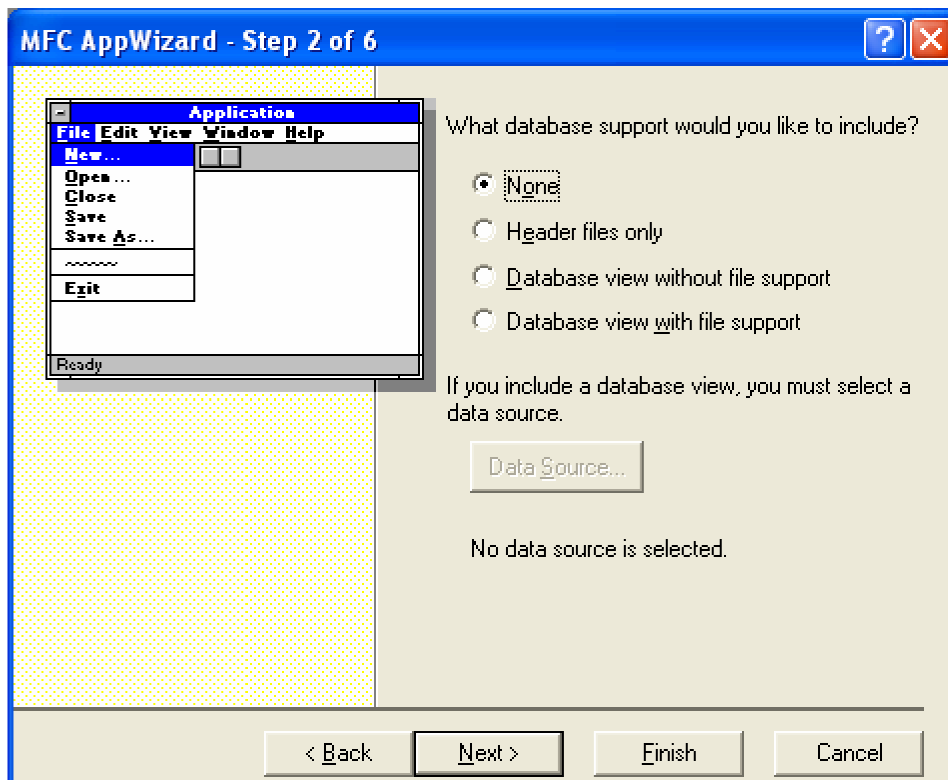

Figure 5: MYMFC29C – AppWizard step 2 of 6.

Don't forget to select **Automation** option.

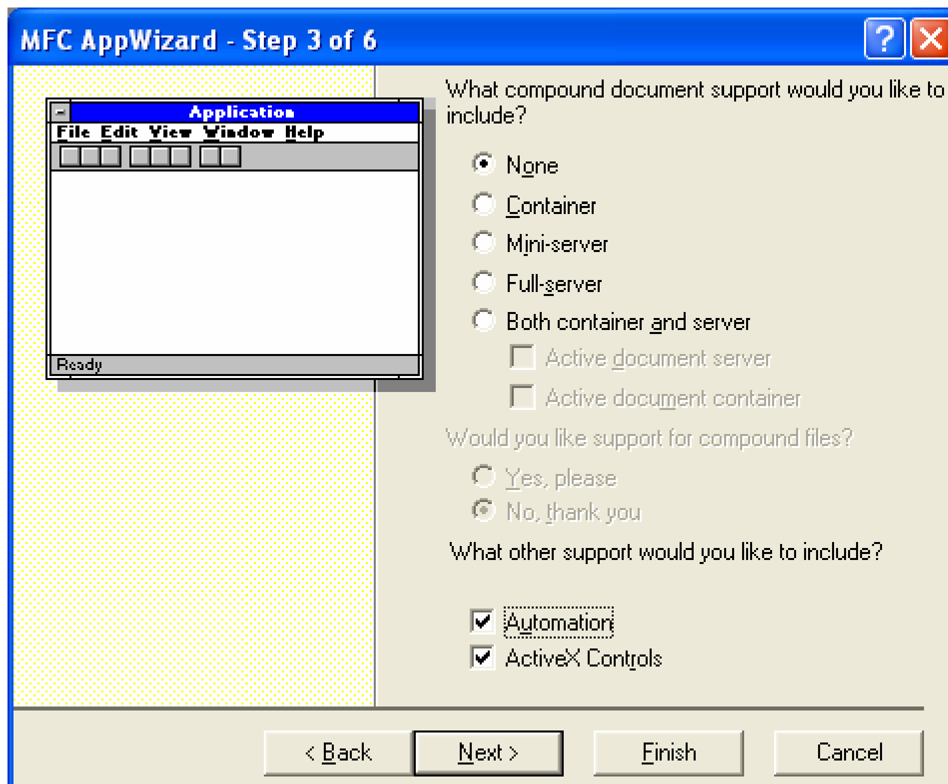

Figure 6: MYMFC29C – AppWizard step 3 of 6.

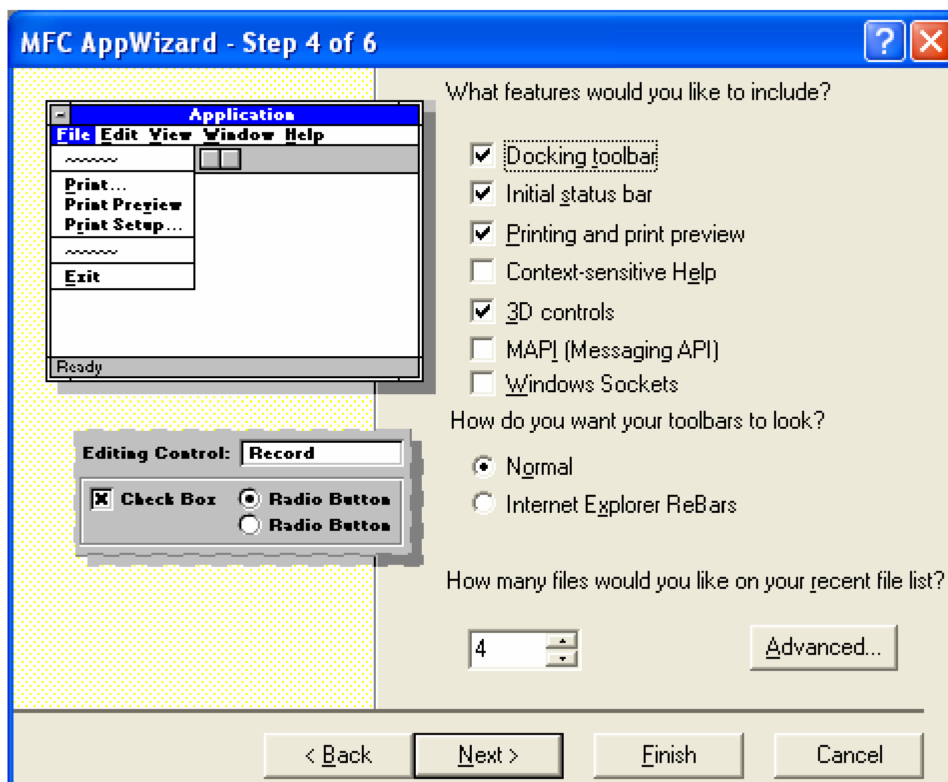

Figure 7: MYMFC29C – AppWizard step 4 of 6.

Click the **Advanced** button. Here, we just use the default settings for **Localized strings**. Take note that the **Doc type name** has the maximum six characters.

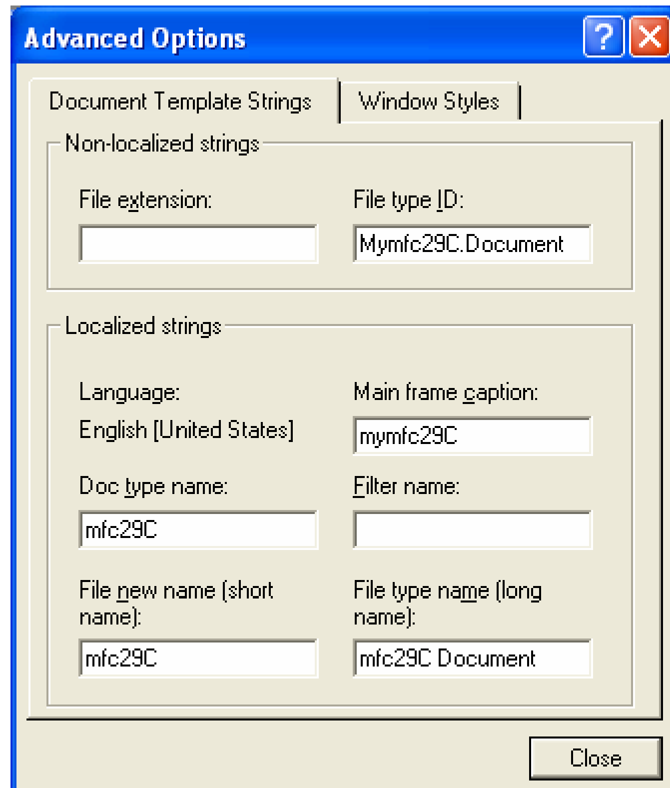

Figure 8: MYMFC29C – AppWizard default **Localized strings**.

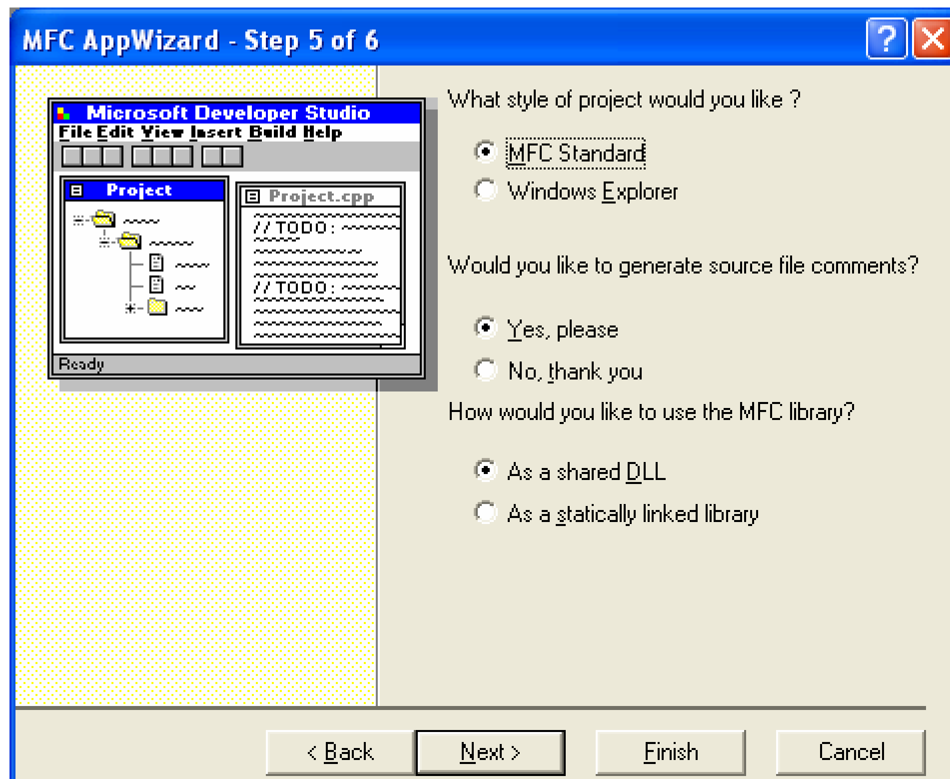

Figure 9: MYMFC29C – AppWizard step 5 of 6.

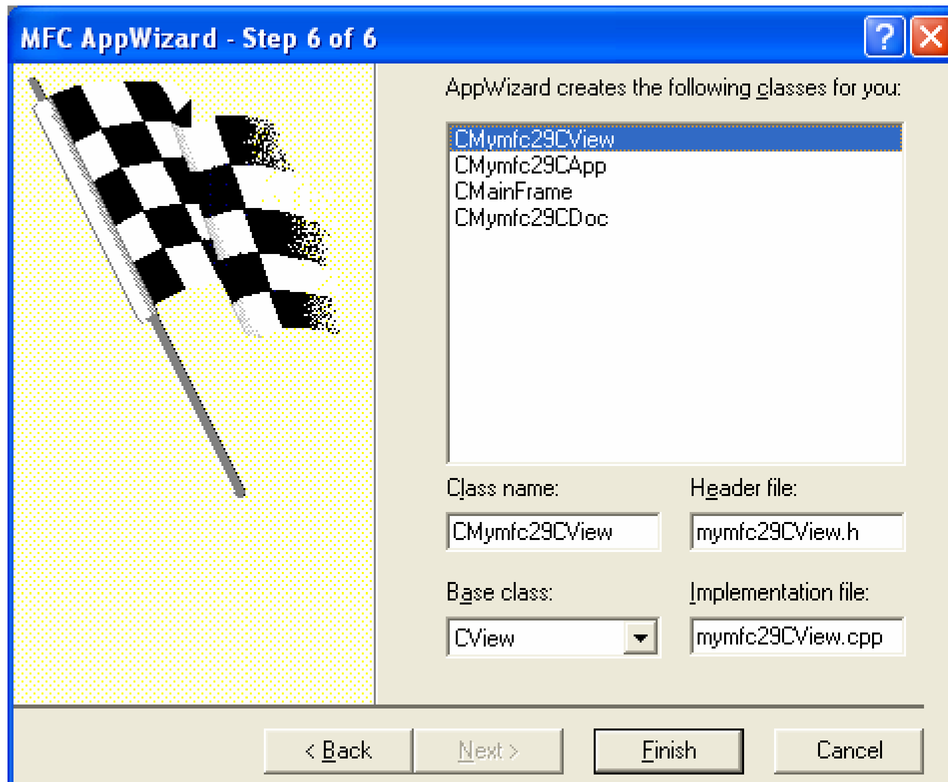

Figure 10: MYMFC29C – AppWizard step 6 of 6.

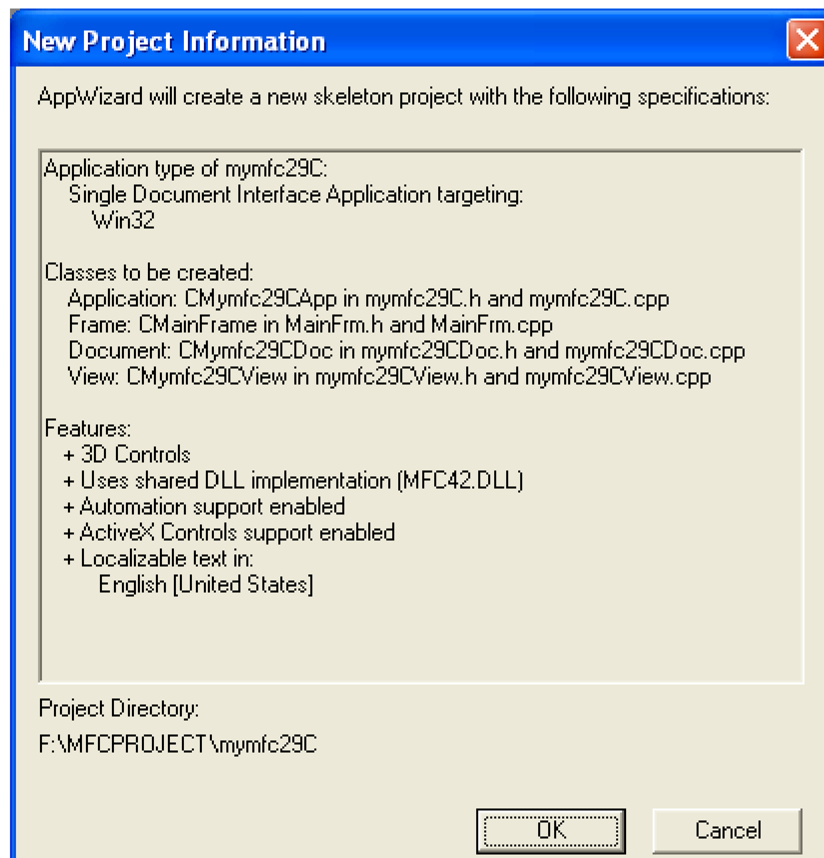

Figure 11: MYMFC29C project summary.

Let create our own icon for this project. Before that, disable the `IDR_MFC29CTYPE` icon by enclosing it in double quotes.

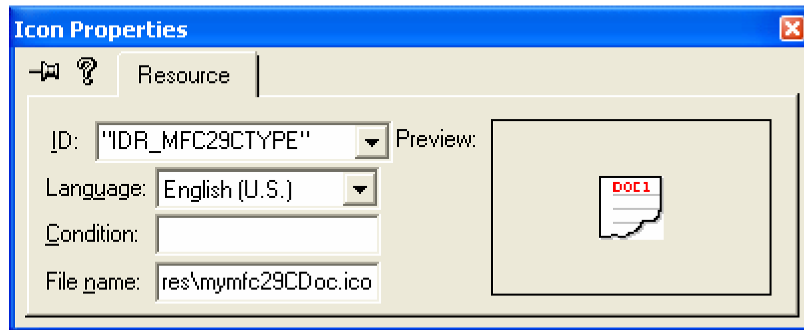

Figure 12: Disabling the default icon.

You can verify it through the **View Resource Symbols** menu.

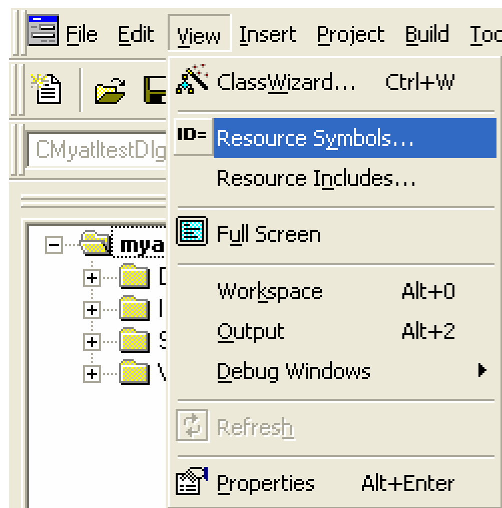

Figure 13: Viewing resource symbols.

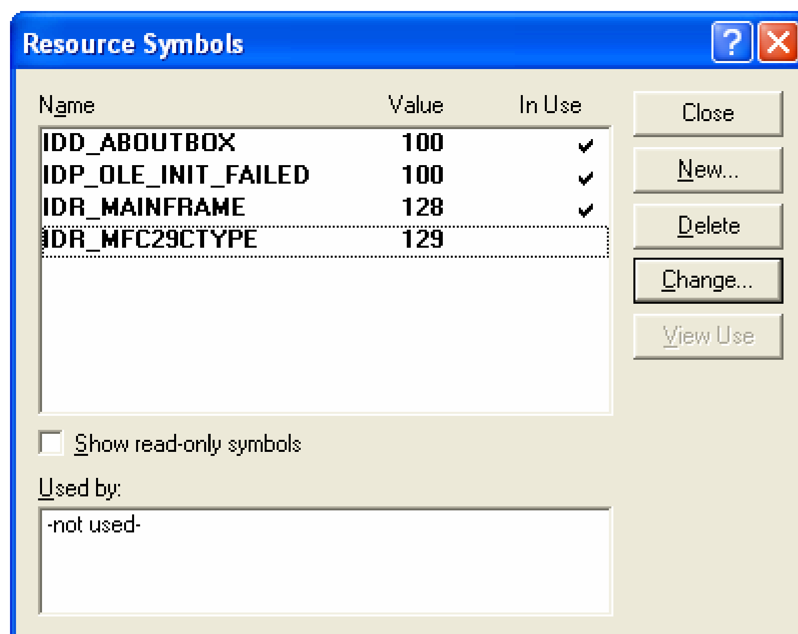

Figure 14: MYMFC29C resource symbols.

Modify the icon for both the **Standard** (32 x 32) and **Small** (16 x 16).

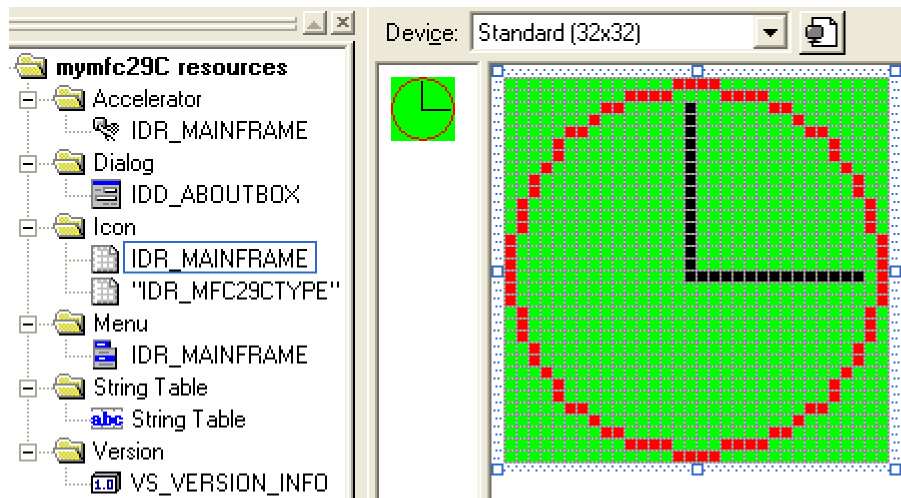

Figure 15: Standard icon.

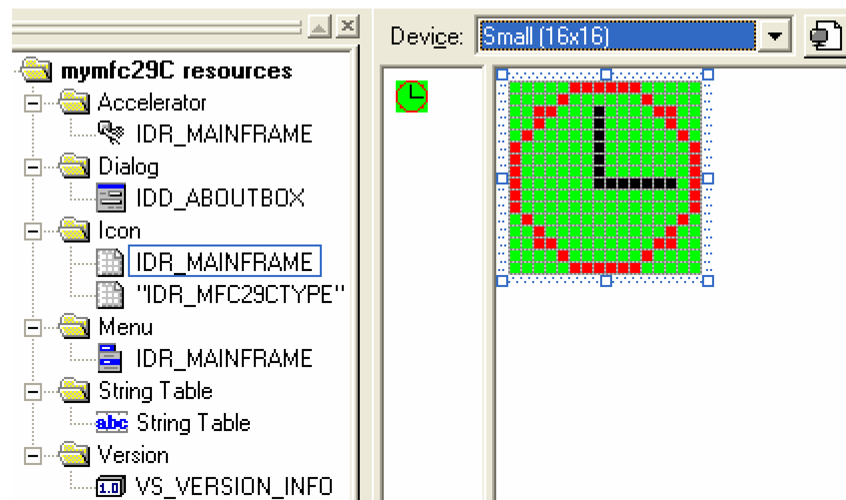

Figure 16: Small icon.

Create a new `CAlarm` class using ClassWizard.

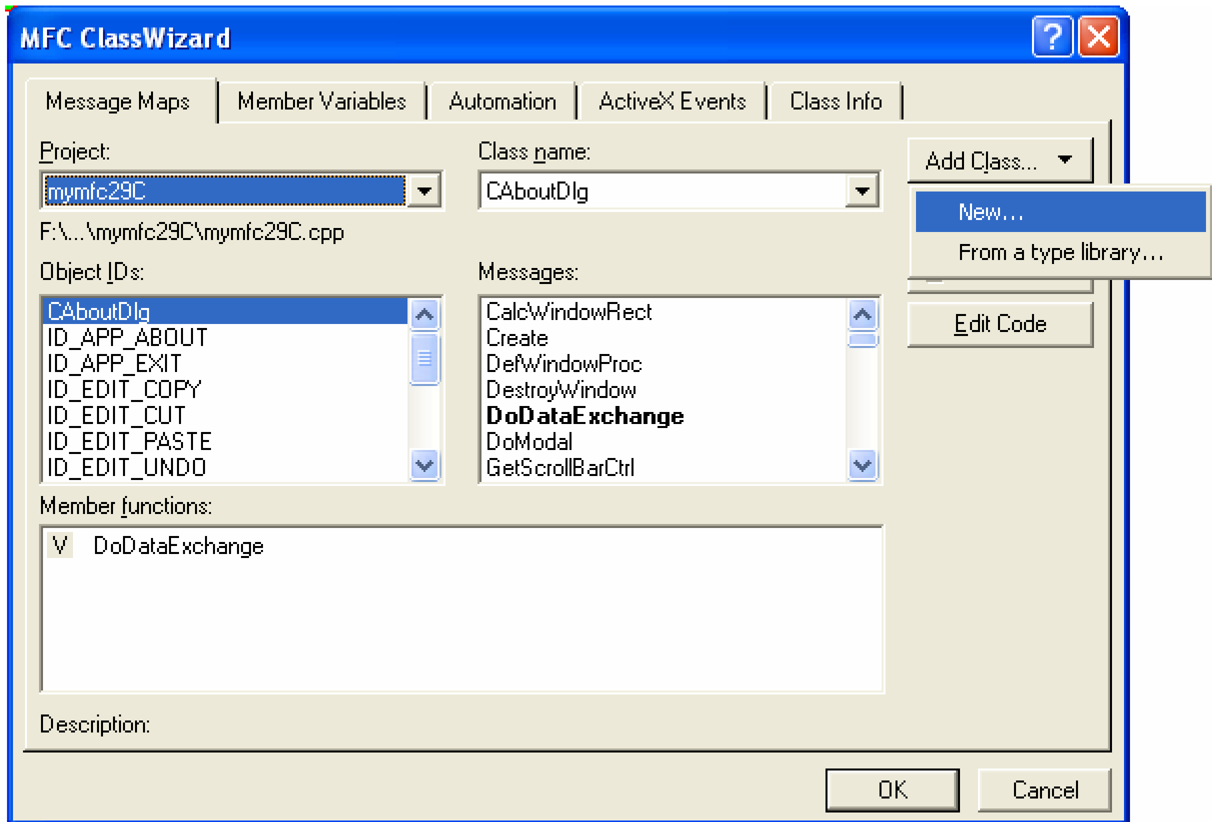

Figure 17: Creating and adding new class to a project using ClassWizard.

Don't forget to select the **Automation** support option.

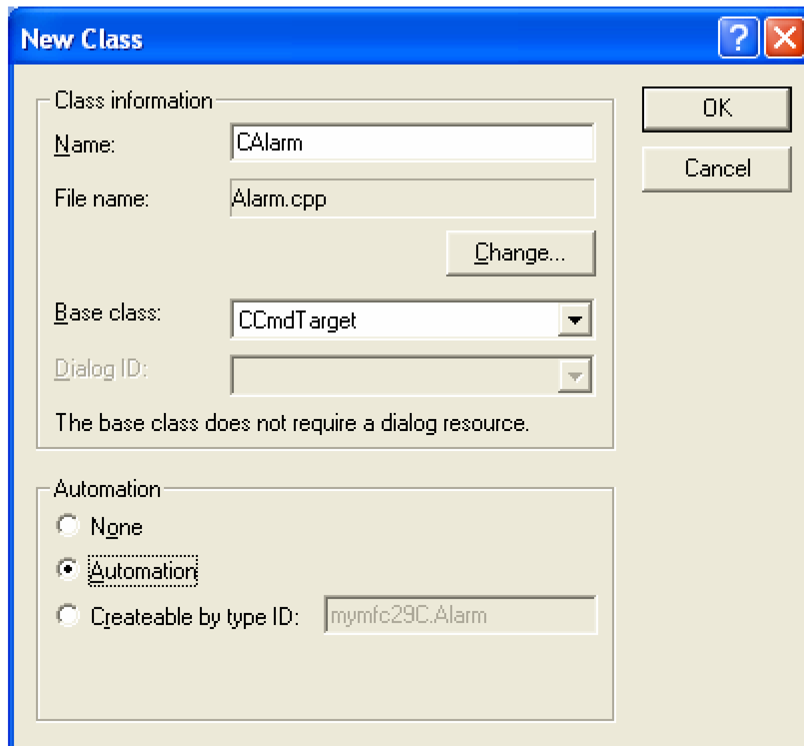

Figure 18: Entering `CAlarm` class information.

Let do a clean-up by rebuilding the ClassWizard database. Delete the **CLW** file under the project directory.

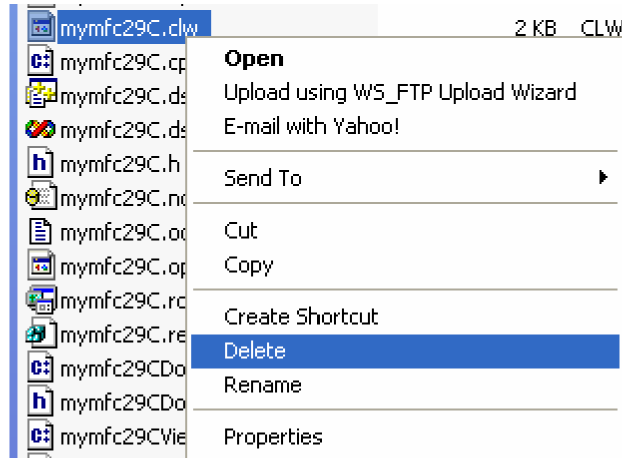

Figure 19: Deleting the ClassWizard database file.

Then launch ClassWizard, the following prompt will be displayed. Just click the **Yes** button for the following prompt.

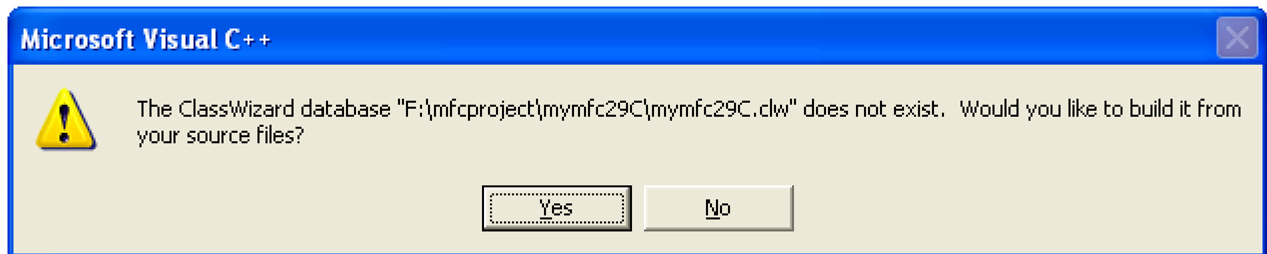

Figure 20: Building ClassWizard database prompt.

And click the **OK** button for the following dialog.

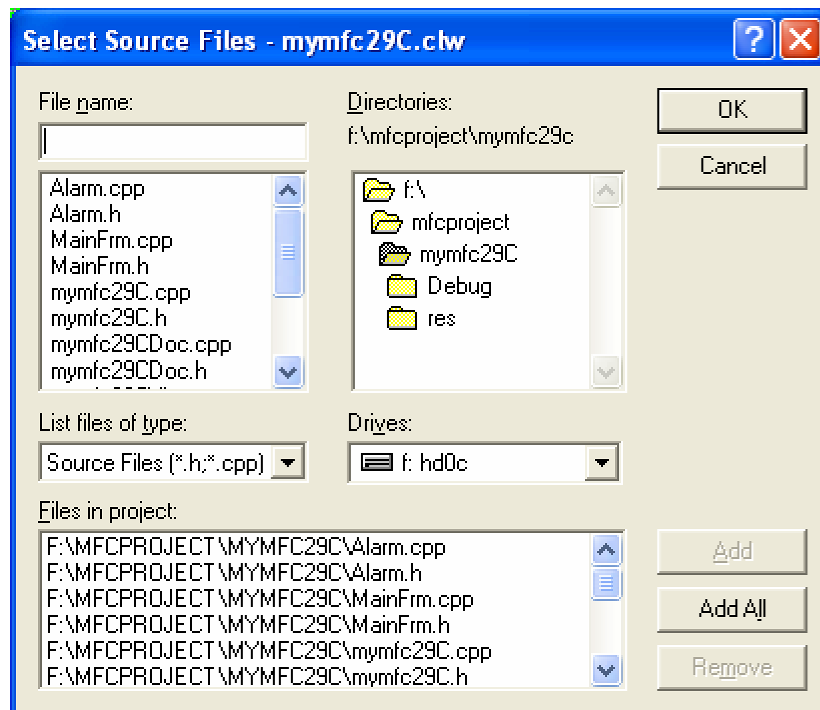

Figure 21: ClassWizard database building files inclusion dialog.

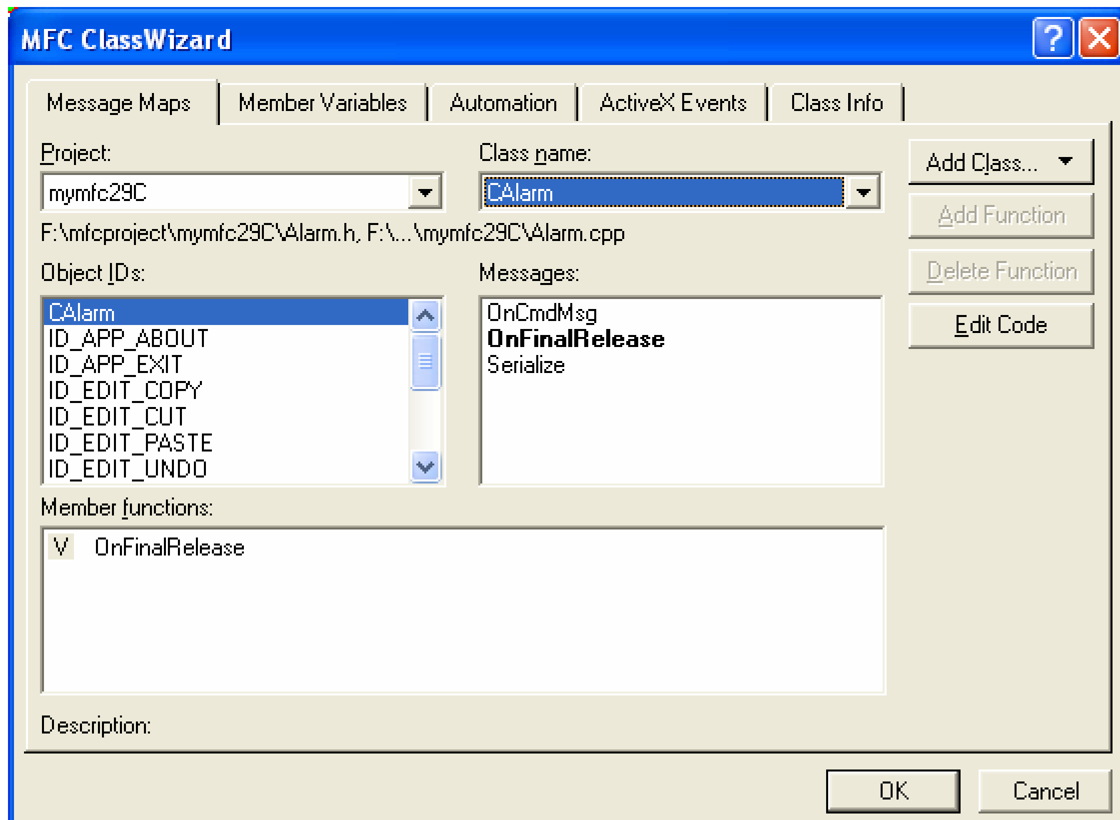

Figure 22: `CAlarm` class in ClassWizard.

Add property to `CAlarm` without notification function. Select the **Automation** tab of the ClassWizard. Then click the **Add Property** button.

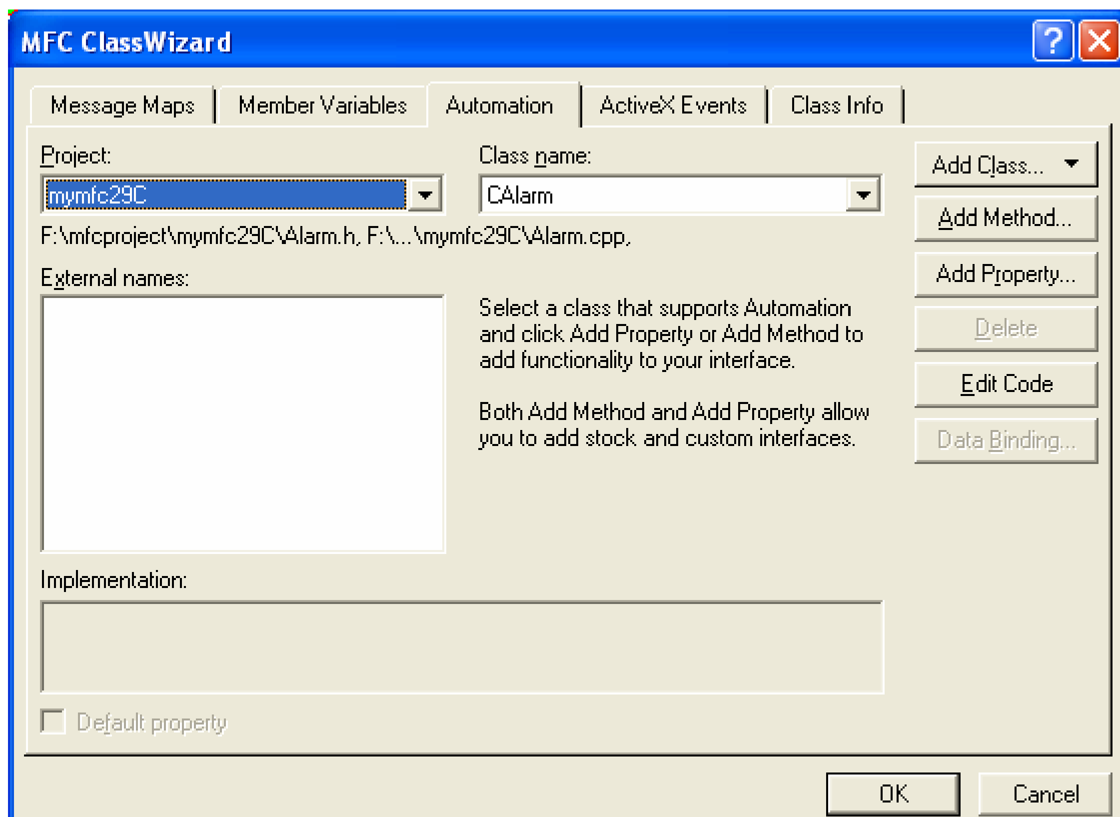

Figure 23: Adding property and method through **Automation** page of ClassWizard.

Add `Time` property, fill up as shown below.

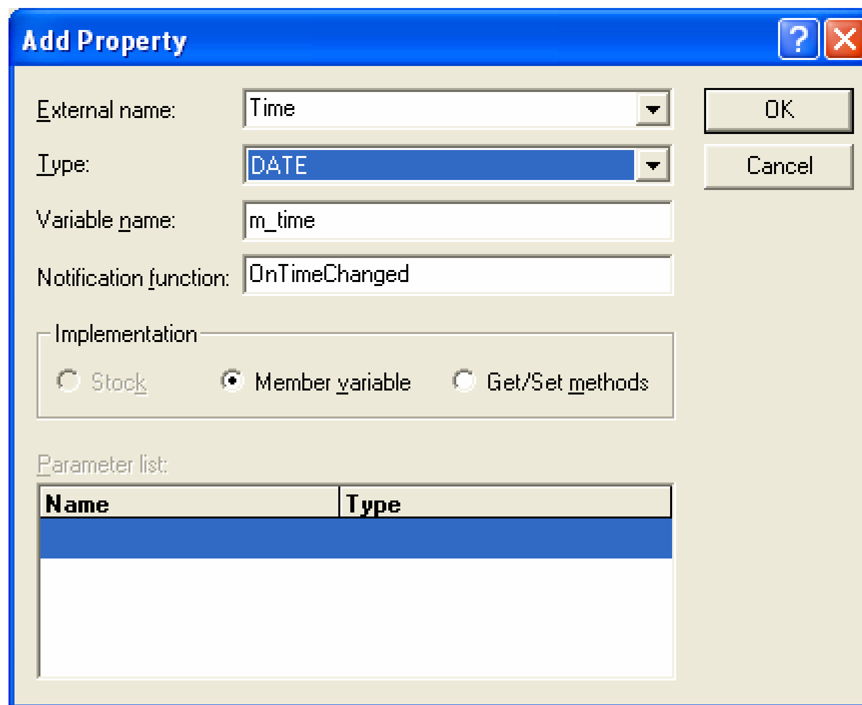

Figure 24: Adding `Time` property.

In **Alarm.h**, replace:

```
DECLARE_DYNCREATE(CAlarm)
```

with

```
DECLARE_DYNAMIC(CAlarm)
```

```
// CAlarm command target

class CAlarm : public CCmdTarget
{
    DECLARE_DYNAMIC(CAlarm)

    CAlarm();           // protected
```

Listing 1.

Modify the `protected` constructor to become `public` (or you can use `friend` keyword) and add the following parameter.

```
DATE time
```

```
class CAlarm : public CCmdTarget
{
    DECLARE_DYNAMIC(CAlarm)

public:
    CAlarm(DATE time);      // public constructor

// Attributes
public:
```

Listing 2.

In **Alarm.cpp** change:

```
IMPLEMENT_DYNCREATE(CAlarm, CCmdTarget)
```

to

```
IMPLEMENT_DYNAMIC(CAlarm, CCmdTarget)
```

```
// CAlarm

IMPLEMENT_DYNAMIC(CAlarm, CCmdTarget)
```

Listing 3.

Then, modify the `CAlarm` constructor as shown below.

```
CAlarm::CAlarm(DATE time)
{
        TRACE("Entering CAlarm ctor -- vaTime = %f\n", (double) time);
        m_time = time;
        EnableAutomation();
}
```

```
// CAlarm

IMPLEMENT_DYNAMIC(CAlarm, CCmdTarget)

CAlarm::CAlarm(DATE time)
{
    TRACE("Entering CAlarm ctor -- vaTime = %f\n", (double) time);
    m_time = time;

    EnableAutomation();
}
```

Listing 4.

Add the following **properties** and **methods** for the `CMymfc29CDoc` class through the **Automation** tab of the ClassWizard.

Name	Description
`Time`	`DATE` property that holds a COM `DATE` (`m_time`)
`Figure`	Indexed `VARIANT` property for the four figures on the clock face (`m_strFigure[]`)
`RefreshWin`	Method that invalidates the view window and brings the main frame window to the top (`Refresh`)
`ShowWin`	Method that displays the application's main window (`ShowWin`)
`CreateAlarm`	Method that creates a `CAlarm` object and returns its `IDispatch` pointer (`CreateAlarm`)

Table 2.

Launch ClassWizard and click the **Automation** tab, select `CMymfc29CDoc` in Class name combo box and click **Add Property** button and fill in the information as shown in the following Figures.

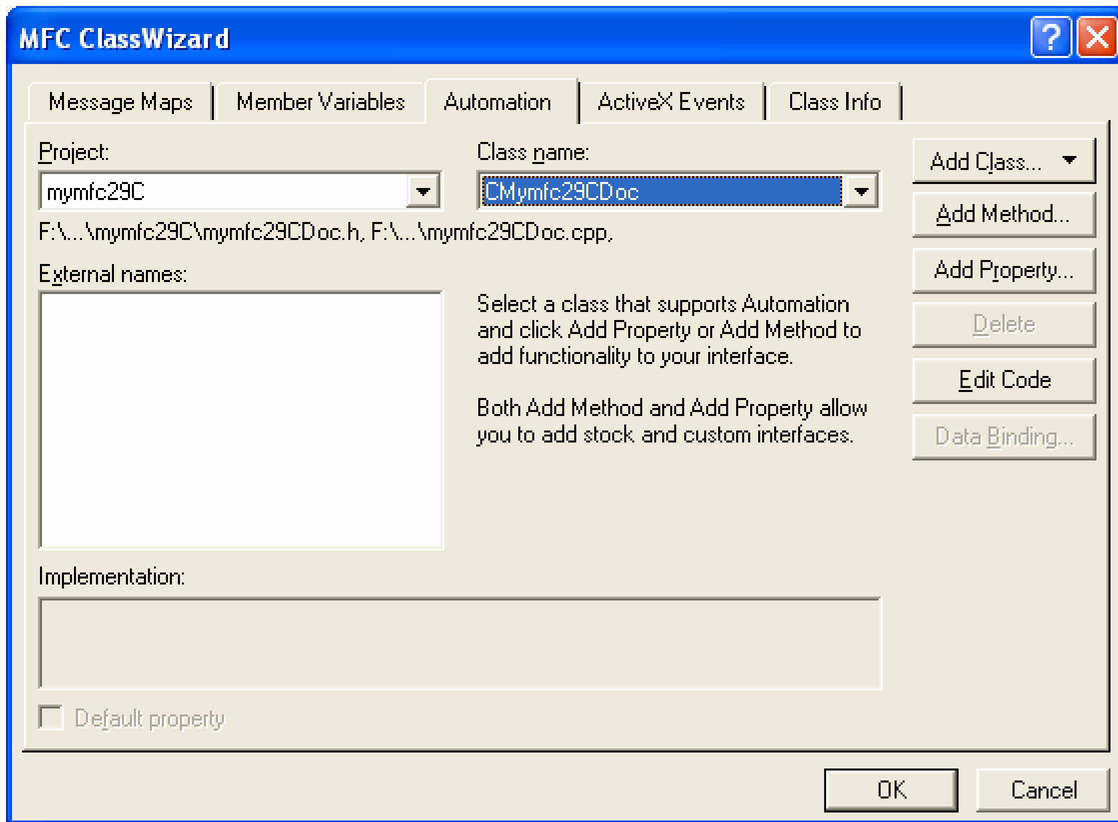

Figure 25: Adding property through **Automation** page of ClassWizard.

Fill up the property information as shown in the following Figures.

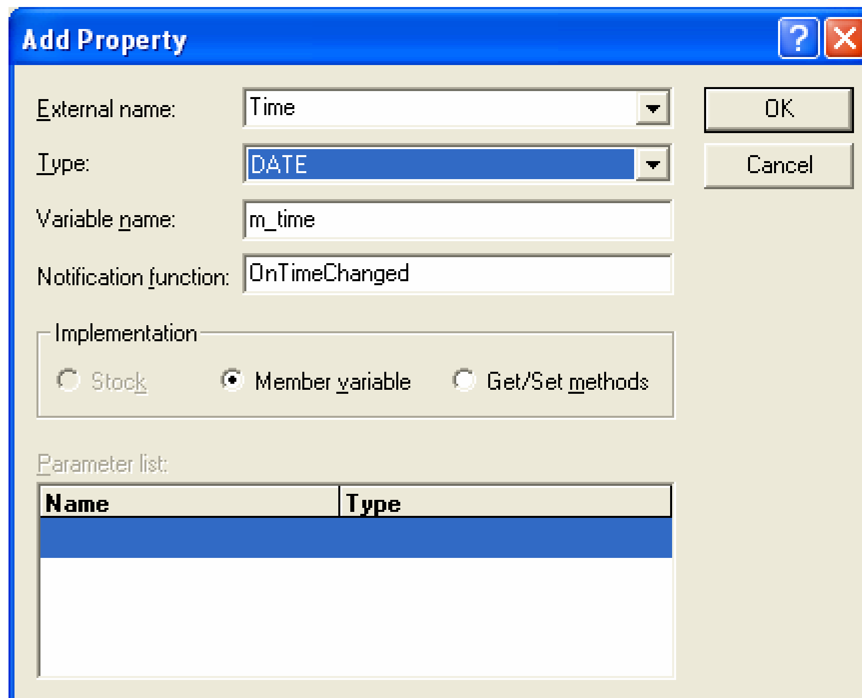

Figure 26: `Time` property.

Figure 27: `Figure` property.

Click the **Add Method** button and add methods as shown in the following Figures.

Figure 28: `RefreshWin` method.

Add Method

External name: ShowWin

Internal name: ShowWin

Return type: void

Implementation: Stock / Custom

Parameter list:

Name	Type

OK

Cancel

Figure 29: `ShowWin` method.

Add Method

External name: CreateAlarm

Internal name: CreateAlarm

Return type: LPDISPATCH

Implementation: Stock / Custom

Parameter list:

Name	Type
time	DATE

OK

Cancel

Figure 30: `CreateAlarm` method.

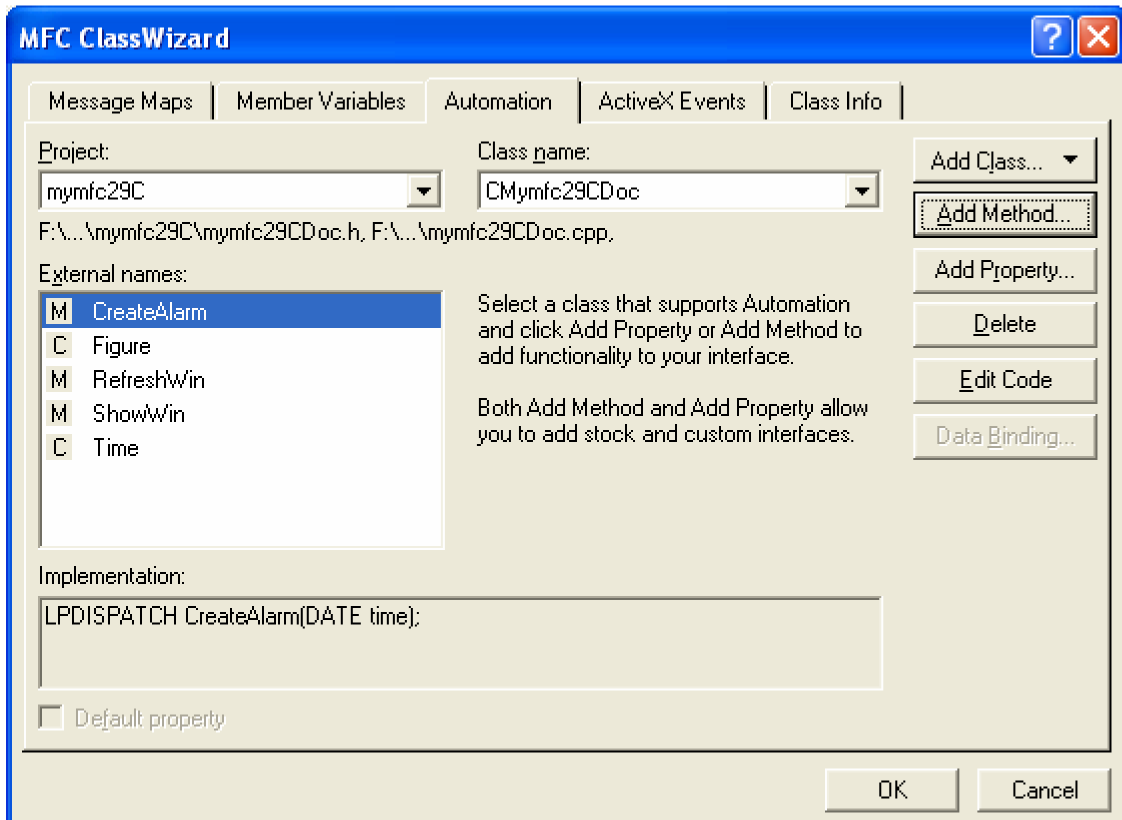

Figure 31: The added properties and methods to `CMymfc29Doc` class.

Add the `SaveModified()` message handler to `CMymfc29CDoc` class.

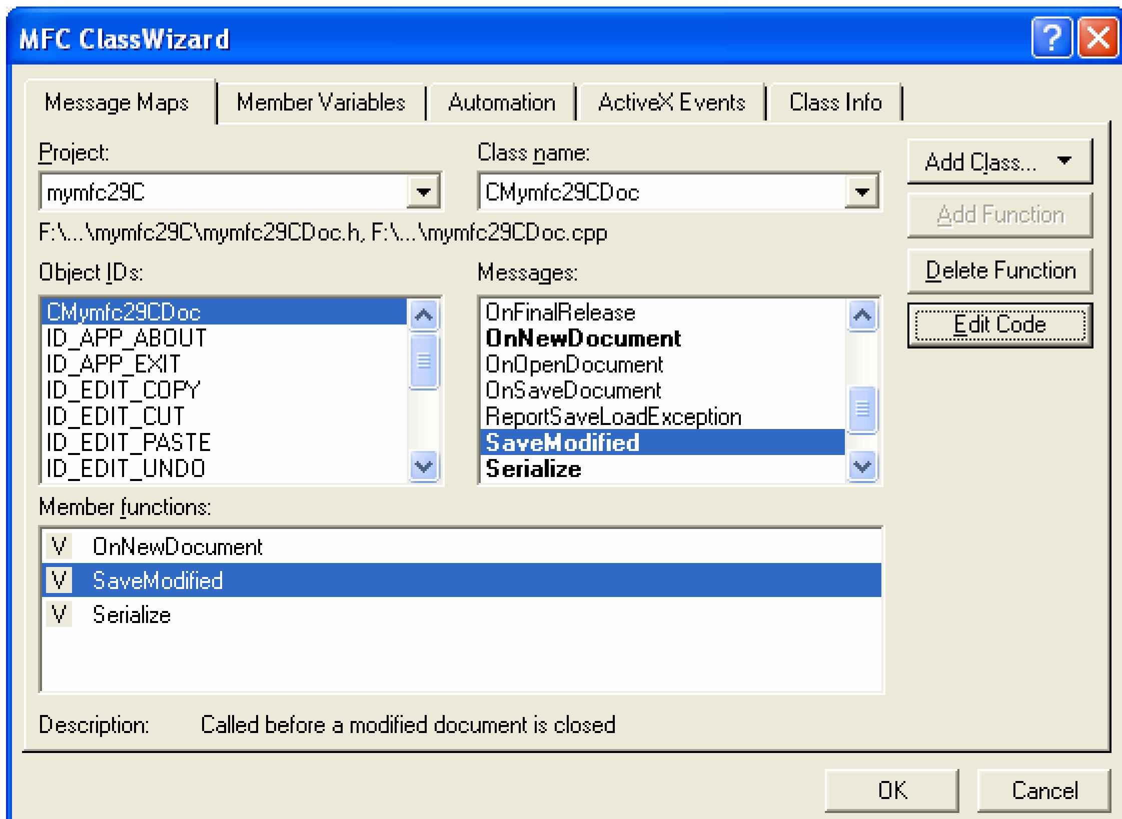

Figure 32: Adding `SaveModified()` message handler to `CMymfc29CDoc` class.

`CMymfc29CDoc` class

Add public member variables for the `mymfc29CDoc` class in the **mymfc29CDoc.h**.

```
public:
       CString m_strFigure[4];
       CAlarm * m_pAlarm;
```

```
class CMymfc29CDoc : public CDocument
{
public:
    CString m_strFigure[4];
    CAlarm * m_pAlarm;

protected: // create from serializatio
```

Listing 5.

Edit/add codes for methods, properties and message handlers of the `CMymfc29C1Doc` class as shown in the following.

```
CMymfc29CDoc::CMymfc29CDoc()
{
       // TODO: add one-time construction code here
       TRACE("Entering CMymfc29CDoc ctor\n");
       m_time = COleDateTime(0, 0, 0, 5, 10, 15);
       // bogus initial values
       m_strFigure[0] = "XII"; m_strFigure[1] = "III";
       m_strFigure[2] = "VI"; m_strFigure[3] = "IX";
       m_pAlarm = NULL;

       EnableAutomation();

       AfxOleLockApp();
}
```

```
// CMymfc29CDoc construction/destruction

CMymfc29CDoc::CMymfc29CDoc()
{
        // TODO: add one-time construction code here
    TRACE("Entering CMymfc29CDoc ctor\n");
    m_time = COleDateTime(0, 0, 0, 5, 10, 15);
    // bogus initial values
    m_strFigure[0] = "XII"; m_strFigure[1] = "III";
    m_strFigure[2] = "VI"; m_strFigure[3] = "IX";
    m_pAlarm = NULL;

    EnableAutomation();

    AfxOleLockApp();
}
```

Listing 6.

```
VARIANT CMymfc29CDoc::GetFigure(short n)
{
       TRACE("Entering CMymfc29CDoc::GetFigure -- n = %d m_strFigure[n] =
%s\n",
            n, m_strFigure[n]);
       return COleVariant(m_strFigure[n]).Detach();
```

```
}
```

```
VARIANT CMymfc29CDoc::GetFigure(short n)
{
    TRACE("Entering CMymfc29CDoc::GetFigure -- n = %d m_strFigure[n] = %s\n",
           n, m_strFigure[n]);
    return COleVariant(m_strFigure[n]).Detach();
}
```

Listing 7.

```
void CMymfc29CDoc::SetFigure(short n, const VARIANT FAR& newValue)
{
      // TODO: Add your property handler here
      TRACE("Entering CMymfc29CDoc::SetFigure -- n = %d, vt = %d\n", n,
              newValue.vt);
      COleVariant vaTemp;
      vaTemp.ChangeType(VT_BSTR, (COleVariant*) &newValue);
      m_strFigure[n] = vaTemp.bstrVal; // converts double-to-single
}
```

```
void CMymfc29CDoc::SetFigure(short n, const VARIANT FAR& newValue)
{
    // TODO: Add your property handler here
    TRACE("Entering CMymfc29CDoc::SetFigure -- n = %d, vt = %d\n", n,
           newValue.vt);
    COleVariant vaTemp;
    vaTemp.ChangeType(VT_BSTR, (COleVariant*) &newValue);
    m_strFigure[n] = vaTemp.bstrVal; // converts double-to-single
}
```

Listing 8.

```
void CMymfc29CDoc::ShowWin()
{
      // TODO: Add your dispatch handler code here
      TRACE("Entering CMymfc29CDoc::ShowWin\n");
      CRect rectWindow;
      CWnd* pFrm = AfxGetApp()->m_pMainWnd;
      pFrm->GetWindowRect(rectWindow);
      WINDOWPLACEMENT wndpl;
      wndpl.length = sizeof(WINDOWPLACEMENT);
      wndpl.showCmd = SW_SHOWNORMAL;
      wndpl.rcNormalPosition.left = rectWindow.left;
      wndpl.rcNormalPosition.top = rectWindow.top;
      wndpl.rcNormalPosition.right = rectWindow.left + 150;
      wndpl.rcNormalPosition.bottom = rectWindow.top + 150;
      pFrm->SetWindowPlacement(&wndpl);
      pFrm->ShowWindow(AfxGetApp()->m_nCmdShow);
      pFrm->UpdateWindow();
      pFrm->BringWindowToTop();
}
```

```
void CMymfc29CDoc::ShowWin()
{
    // TODO: Add your dispatch handler code here
    TRACE("Entering CMymfc29CDoc::ShowWin\n");
    CRect rectWindow;
    CWnd* pFrm = AfxGetApp()->m_pMainWnd;
    pFrm->GetWindowRect(rectWindow);
    WINDOWPLACEMENT wndpl;
    wndpl.length = sizeof(WINDOWPLACEMENT);
    wndpl.showCmd = SW_SHOWNORMAL;
    wndpl.rcNormalPosition.left = rectWindow.left;
    wndpl.rcNormalPosition.top = rectWindow.top;
    wndpl.rcNormalPosition.right = rectWindow.left + 150;
    wndpl.rcNormalPosition.bottom = rectWindow.top + 150;
    pFrm->SetWindowPlacement(&wndpl);
    pFrm->ShowWindow(AfxGetApp()->m_nCmdShow);
    pFrm->UpdateWindow();
    pFrm->BringWindowToTop();
}
```

Listing 9.

```
LPDISPATCH CMymfc29CDoc::CreateAlarm(DATE time)
{
      // TODO: Add your dispatch handler code here
      TRACE("Entering CMymfc29CDoc::CreateAlarm, time = %f\n", time);
      // OLE deletes any prior CAlarm object
      m_pAlarm = new CAlarm(time);
      return m_pAlarm->GetIDispatch(FALSE);   // no AddRef here
}

BOOL CMymfc29CDoc::SaveModified()
{
      // TODO: Add your specialized code here and/or call the base class
      return TRUE;
}
```

```
LPDISPATCH CMymfc29CDoc::CreateAlarm(DATE time)
{
    // TODO: Add your dispatch handler code here
    TRACE("Entering CMymfc29CDoc::CreateAlarm, time = %f\n", time);
    // OLE deletes any prior CAlarm object
    m_pAlarm = new CAlarm(time);
    return m_pAlarm->GetIDispatch(FALSE);   // no AddRef here
}

BOOL CMymfc29CDoc::SaveModified()
{
    // TODO: Add your specialized code here and/or call the base class
    return TRUE;
}
```

Listing 10.

CMainFrame class

Add the following statement in the CMainFrame::PreCreateWindow function to disable the menu.

```
      cs.hMenu = NULL;   // no menu
      return CFrameWnd::PreCreateWindow(cs);
```

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
     cs.hMenu = NULL;  // no menu
     return CFrameWnd::PreCreateWindow(cs);
}
```

Listing 11.

`CMymfc29CView` class

Using ClassView or manually add the following data members in **mymfc29CView.h**.

```
private:
   static const int nX[60];
   static const int nY[60];
   CFont m_font;
```

```
class CMymfc29CView : public CView
{
private:
    static const int nX[60];
    static const int nY[60];
    CFont m_font;
```

Listing 12.

Add the following in the **mymfc29CView.cpp**.

```
// computer-generated x-y coordinates for 60 points around a circle
const int CMymfc29CView::nX[] = {
   0, 105, 208, 309, 407, 500, 588, 669, 743, 809, 866, 914,
   951, 978, 995, 1000, 995, 978, 951, 914, 866, 809, 743, 669,
   588, 500, 407, 309, 208, 105, 0, -104, -207, -308, -406, -499,
   -587, -668, -742, -808, -865, -913, -950, -977, -994, -999, -994, -977,
   -950, -913, -865, -808, -742, -668, -587, -499, -406, -308, -207, -104 };
const int CMymfc29CView::nY[] = {
   999, 994, 978, 951, 913, 866, 809, 743, 669, 587, 499, 406,
   309, 207, 104, 0, -104, -207, -309, -406, -499, -587, -669, -743,
   -809, -866, -913, -951, -978, -994, -999, -994, -978, -951, -913, -866,
   -809, -743, -669, -587, -500, -406, -309, -207, -104, 0, 104, 207,
   309, 406, 499, 587, 669, 743, 809, 866, 913, 951, 978, 994 };
```

```
// CMymfc29CView

// computer-generated x-y coordinates for 60 points around a circle

const int CMymfc29CView::nX[] = {
    0, 105, 208, 309, 407, 500, 588, 669, 743, 809, 866, 914,
    951, 978, 995, 1000, 995, 978, 951, 914, 866, 809, 743, 669,
    588, 500, 407, 309, 208, 105, 0, -104, -207, -308, -406, -499,
    -587, -668, -742, -808, -865, -913, -950, -977, -994, -999, -994, -977,
    -950, -913, -865, -808, -742, -668, -587, -499, -406, -308, -207, -104 };
const int CMymfc29CView::nY[] = {
    999, 994, 978, 951, 913, 866, 809, 743, 669, 587, 499, 406,
    309, 207, 104, 0, -104, -207, -309, -406, -499, -587, -669, -743,
    -809, -866, -913, -951, -978, -994, -999, -994, -978, -951, -913, -866,
    -809, -743, -669, -587, -500, -406, -309, -207, -104, 0, 104, 207,
    309, 406, 499, 587, 669, 743, 809, 866, 913, 951, 978, 994 };

IMPLEMENT_DYNCREATE(CMymfc29CView, CView)
```

Listing 13.

Add the code for the `CMymfc29CView::OnDraw()` member function.

```
void CMymfc29CView::OnDraw(CDC* pDC)
{
   CMymfc29CDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   CRect rectClient;
   CPen pen0(PS_SOLID, 0, (COLORREF) 0);
   CPen pen1(PS_SOLID, 20, (COLORREF) 0);
   CPen pen2(PS_SOLID, 30, (COLORREF) 0);
   int i, j, n, m, nS, nM, nH;
   CSize sz;
   pDC->SetMapMode(MM_ISOTROPIC);
   GetClientRect(rectClient);
   pDC->SetWindowExt(2400, 2400);
   pDC->SetViewportExt(rectClient.right, -rectClient.bottom);
   pDC->SetViewportOrg(rectClient.right / 2, rectClient.bottom / 2);
   pDC->SelectObject(&m_font);
   pDC->SelectObject(&pen0);
   for(i = 0; i < 60; i++) { // second/minute ticks
      pDC->MoveTo(nX[i], nY[i]);
      pDC->LineTo(nX[i] * 9 / 10, nY[i] * 9 / 10);
   }
   pDC->SelectObject(&pen1);
   for(i = 0; i < 12; i++) { // hour ticks
      j = i * 5;
      pDC->MoveTo(nX[j], nY[j]);
      pDC->LineTo(nX[j] * 8 / 10, nY[j] * 8 / 10);
   }
   for(i = 0; i < 4; i++) { // figures
      j = i * 15;
      sz = pDC->GetTextExtent(pDoc->m_strFigure[i], pDoc-
>m_strFigure[i].GetLength());
      n  = nX[j] * 10 / 9 - sz.cx / 2;
      m  = nY[j] * 10 / 9 + sz.cy / 2;
      pDC->TextOut(n, m, pDoc->m_strFigure[i]);
   }

   // draw the clock hands
   COleDateTime dt = pDoc->m_time;
   nH = dt.GetHour();
   nM = dt.GetMinute();
   nS = dt.GetSecond();
   pDC->SelectObject(&pen0);
   i = nS;
   pDC->MoveTo(0, 0);
   pDC->LineTo(nX[i], nY[i]); // seconds

   pDC->SelectObject(&pen1);
   i = nM;
   pDC->MoveTo(0, 0);
   pDC->LineTo(nX[i] * 8 / 10, nY[i] * 8 / 10); // minutes

   pDC->SelectObject(&pen2);
   i = nH * 5;
   i = (i % 60) + nM / 12;
   pDC->MoveTo(0, 0);
   pDC->LineTo(nX[i] * 6 / 10, nY[i] * 6 / 10); // hours

   pDC->SelectStockObject(BLACK_PEN);
   CAlarm* pAlarm;
   if((pAlarm = pDoc->m_pAlarm) != NULL) {
```

```
        COleDateTime dt = pAlarm->m_time;
        nH = dt.GetHour();
        nM = dt.GetMinute();
        nS = dt.GetSecond();
          TRACE("dt = %f, nH = %d, nM = %d, nS = %d\n", dt.m_dt, nH, nM, nS);
        i = nH * 5;
        i = (i % 60) + nM / 12;
        pDC->TextOut(nX[i] * 8 / 10, nY[i] * 8 / 10, "A"); // alarm
    }
}
```

Add the WM_CREATE Windows message handler.

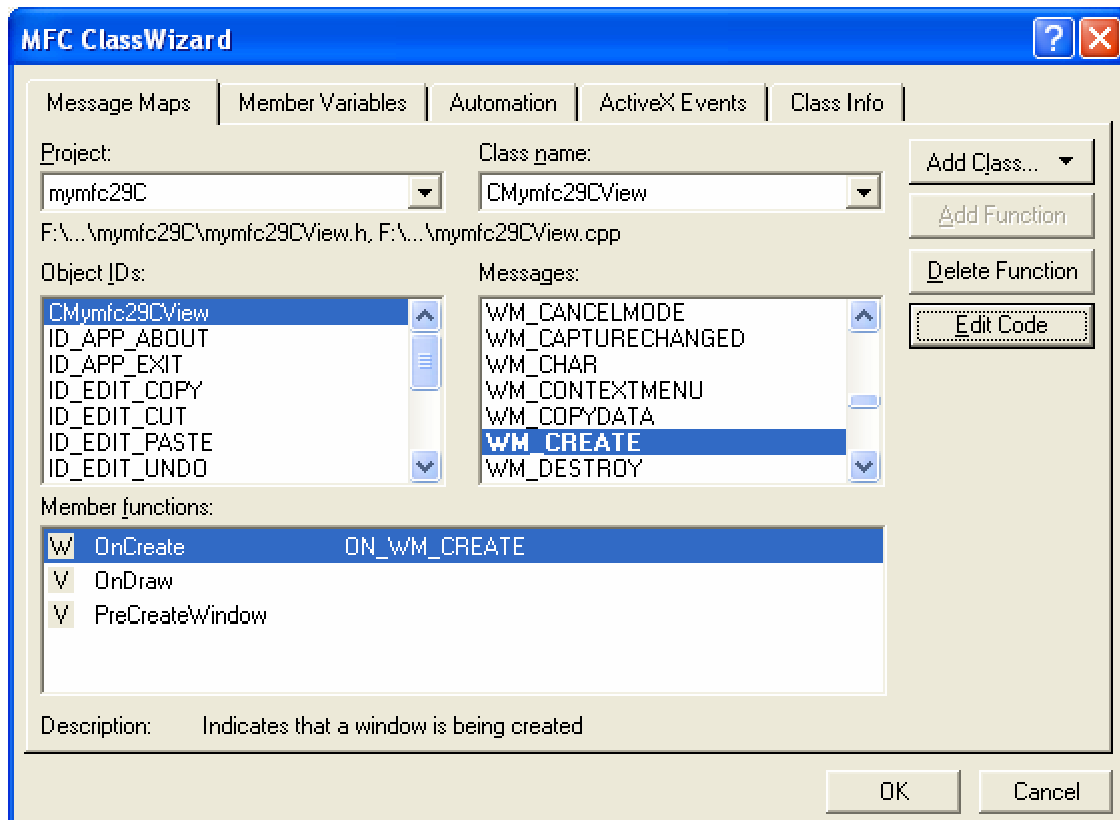

Figure 33: Adding WM_CREATE Windows message handler to CMymfc29CView class.

Then, add/edit the OnCreate() code.

```
int CMymfc29CView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CView::OnCreate(lpCreateStruct) == -1)
   return -1;
   m_font.CreateFont(150, 0, 0, 0, 400, FALSE, FALSE,
                0, ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                DEFAULT_PITCH | FF_SWISS,
                "Arial");
   return 0;
}
```

```
// CMymfc29CView message handlers

int CMymfc29CView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
    return -1;
    m_font.CreateFont(150, 0, 0, 0, 400, FALSE, FALSE,
                    0, ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                    CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                    DEFAULT_PITCH | FF_SWISS,
                    "Arial");
    return 0;
}
```

Listing 14.

Add the #include statement for the **mymfc29CView.cpp**.

```
#include "Alarm.h"
```

```
#include "stdafx.h"
#include "mymfc29C.h"

#include "Alarm.h"

#include "mymfc29CDoc.h"
#include "mymfc29CView.h"
```

Listing 15.

Change the protected OLE dispatch map functions in **mymfc29CDoc.h** to public (or you can use friend keyword).

```
public:
    // Generated OLE dispatch map functions
    //{{AFX_DISPATCH(CMymfc29CDoc)
    DATE m_time;
    afx_msg void OnTimeChanged();
    afx_msg void Refresh();
    afx_msg void ShowWin();
    afx_msg LPDISPATCH CreateAlarm(DATE time);
    afx_msg VARIANT GetFigure(short n);
    afx_msg void SetFigure(short n, const VARIANT FAR& newValue);
    //}}AFX_DISPATCH
    DECLARE_DISPATCH_MAP()
    DECLARE_INTERFACE_MAP()
};
```

Listing 16.

Also change the protected OLE dispatch map functions in **Alarm.h** to public (or you can use friend keyword).

```
public:
    //{{AFX_DISPATCH(CAlarm)
    DATE m_time;
    afx_msg void OnTimeChanged();
    //}}AFX_DISPATCH
    DECLARE_DISPATCH_MAP()
    DECLARE_INTERFACE_MAP()
};
```

Listing 17.

Add the #include statement to the **mymfc29C.cpp** and **mymfc29CDoc.cpp** file.

```
#include "Alarm.h"
```

```
#include "stdafx.h"
#include "mymfc29C.h"

#include "Alarm.h"

#include "MainFrm.h"
#include "mymfc29CDoc.h"
#include "mymfc29CView.h"
```

Listing 18.

```
#include "stdafx.h"
#include "mymfc29C.h"

#include "Alarm.h"

#include "mymfc29CDoc.h"
```

Listing 19.

Build the program and make sure there is no error. Next, we need a client to test our component.

For the Excel, launch Excel, save the file as mymfc29C.xls and invoke the Visual Basic editor. Double-click the **sheet1** on the top-left window. Copy and paste the following macros.

```
Dim Clock As Object
Dim Alarm As Object

Sub LoadClock()
    Set Clock = CreateObject("Mymfc29C.Document")
    Range("A3").Select
    n = 0
    Do Until n < 4
        Clock.figure(n) = Selection.Value
        Selection.Offset(0, 1).Range("A1").Select
        n = n + 1
    Loop
    RefreshClock
    Clock.ShowWin
End Sub

Sub RefreshClock()
    Clock.Time = Now()
    Clock.RefreshWin
End Sub

Sub CreateAlarm()
    Range("E3").Select
    Set Alarm = Clock.CreateAlarm(Selection.Value)
    RefreshClock
End Sub

Sub UnloadClock()
    Set Clock = Nothing
End Sub
```

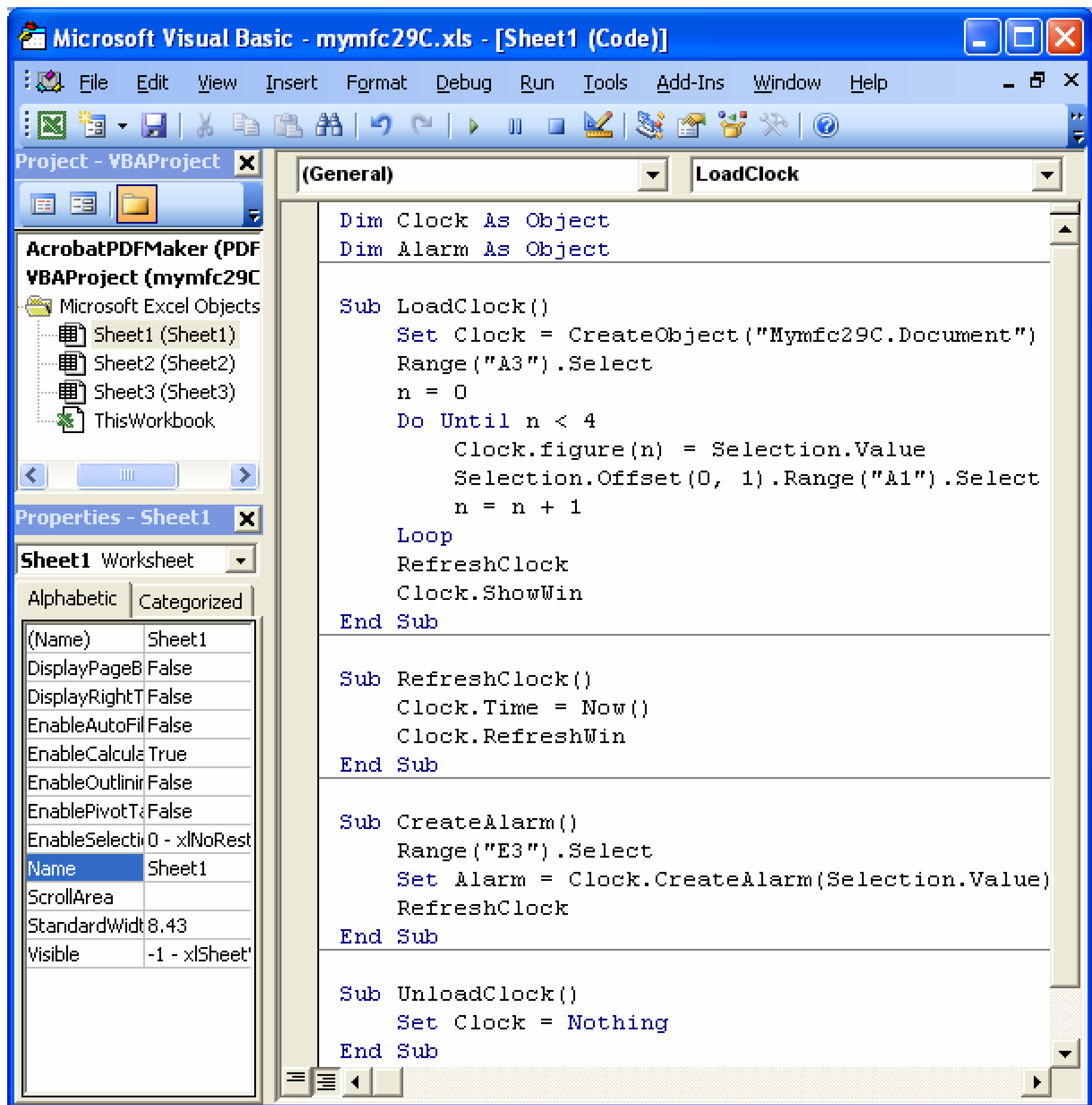

Figure 34: Creating macros with Visual Basic editor.

Arrange the buttons as shown below and assign the previously created macros to the buttons appropriately. Be careful, the cells in macro code should match with the Excel cell.

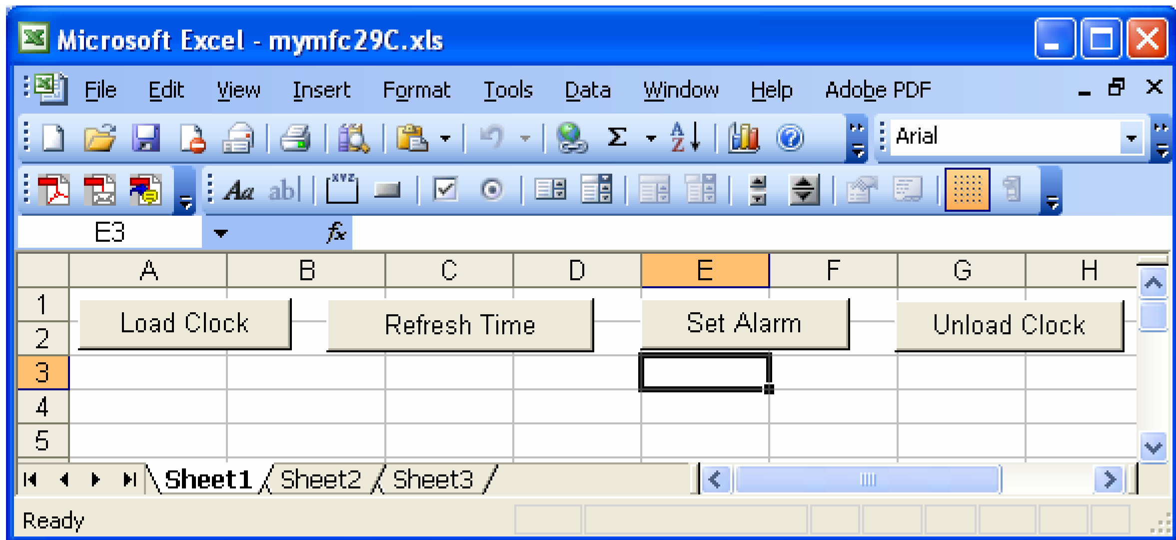

Figure 35: **mymfc29C.xls** button layout.

Next, we are going to test the MYMFC29C component. Click the **Load Clock** button.

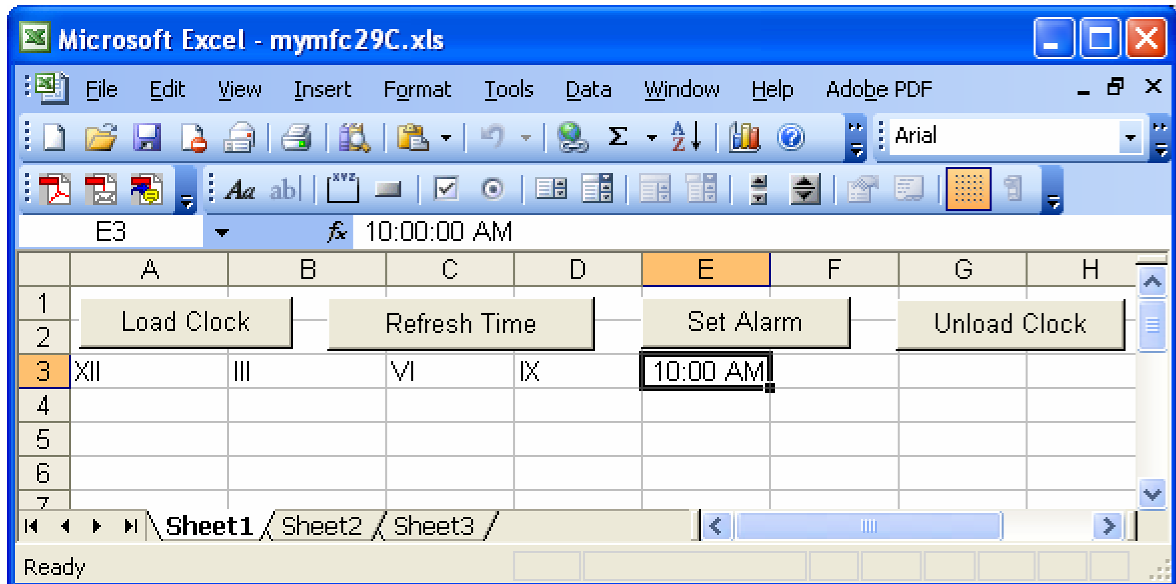

Figure 36: **mymfc29C.xls**, a client program in action.

The following clock is launched. At the same time `XII`, `III`, `VI` and `IX` value will be displayed in the appropriate Excel cells. Type the alarm in **`E3`** cell and click the **Set Alarm** button. The alarm will be set in the clock as shown in Figure 38.

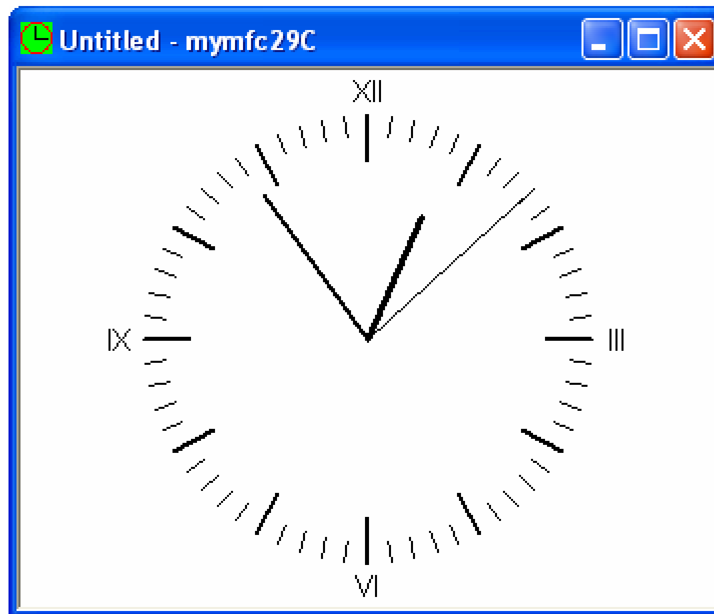

Figure 37: Clock (component) triggered through client program (Excel).

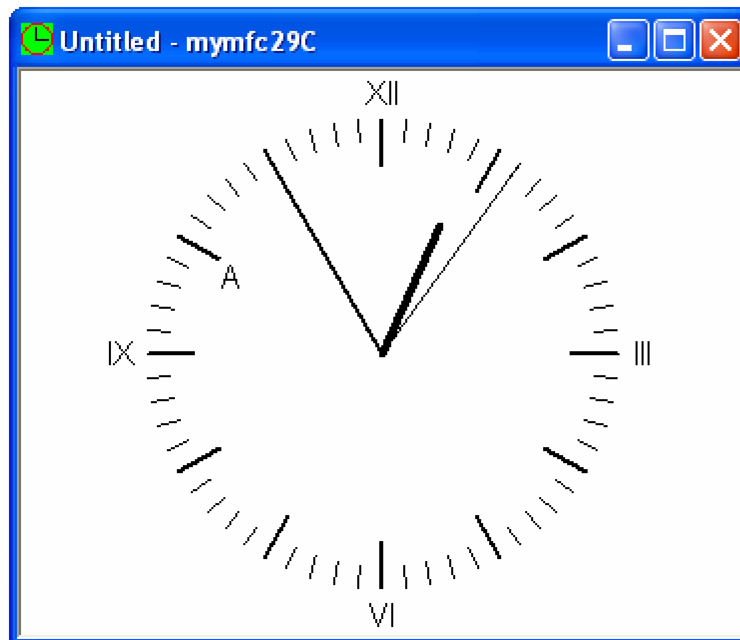

Figure 38: Alarm was set through a client.

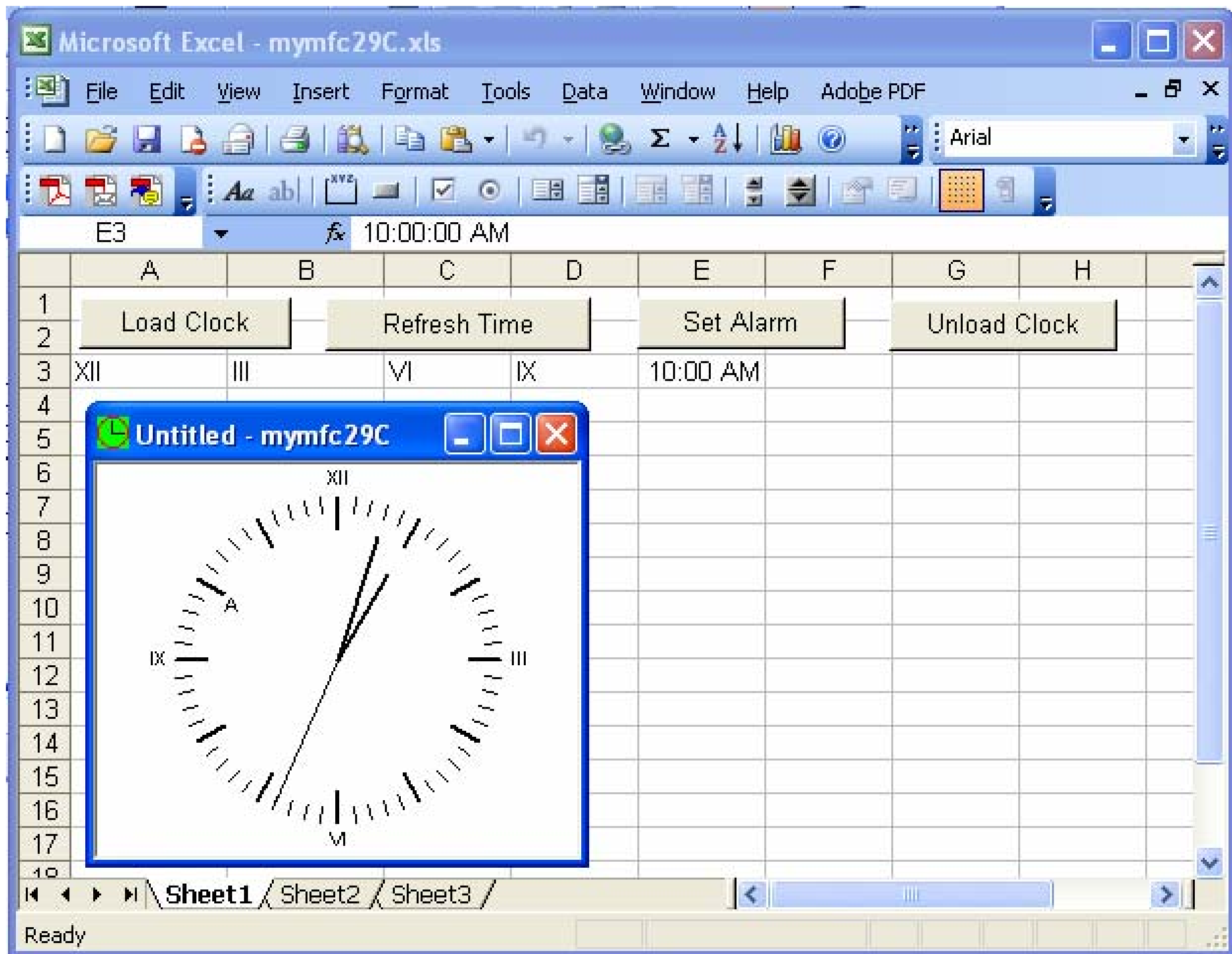

Figure 39: MYMFC29C in action, component and client side-by-side.

The Story

If you're wondering why there's no menu, it's because of the following statement in the `CMainFrame::PreCreateWindow` function:

```
cs.hMenu = NULL;   // no menu
return CFrameWnd::PreCreateWindow(cs);
```

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.hMenu = NULL;  // no menu
    return CFrameWnd::PreCreateWindow(cs);
}
```

Listing 20.

Close the **Clock** program and then click the **Unload Clock** button. If you've started the component program from the debugger, you can watch the Debug window for a message box that indicates that the `ExitInstance()` function is called.

AppWizard did most of the work of setting up the document as an Automation component. In the derived application class `CMymfc29CApp`, it generated a data member for the component, as shown here:

```
public:
    COleTemplateServer m_server;
```

The MFC COleTemplateServer class is derived from COleObjectFactory. It is designed to create a COM document object when a client calls IClassFactory::CreateInstance. The class ID comes from the **global clsid** variable defined in **mymfc29C.cpp**. The human-readable program ID (**Mymfc29C.Document**) comes from the IDR_MAINFRAME string resource.

In the InitInstance() function (in **mymfc29C.cpp**), AppWizard generated the code below, which connects the component object (the document) to the application's document template.

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CMymfc29CDoc),
    RUNTIME_CLASS(CMainFrame),    // main SDI frame window
    RUNTIME_CLASS(CMymfc29CView));
AddDocTemplate(pDocTemplate);
.
.
.
m_server.ConnectTemplate(clsid, pDocTemplate, TRUE);
```

Now all the plumbing is in place for COM and the framework to construct the document, together with the view and frame. When the objects are constructed, however, the main window is not made visible. That's your job. You must write a method that shows the window.

The following UpdateRegistry() call from the InitInstance() function updates the Windows Registry with the contents of the project's IDR_MAINFRAME string resource:

```
m_server.UpdateRegistry(OAT_DISPATCH_OBJECT);
```

The following dispatch map in the **mymfc29CDoc.cpp** file shows the properties and methods for the CMymfc29CDoc class. Note that the Figure property is an indexed property that ClassWizard can generate if you specify a parameter. Later you'll see the code you have to write for the GetFigure() and SetFigure() functions.

```
...
BEGIN_DISPATCH_MAP(CMymfc29CDoc, CDocument)
    //{{AFX_DISPATCH_MAP(CMymfc29CDoc)
    DISP_PROPERTY_NOTIFY(CMymfc29CDoc, "Time", m_time, OnTimeChanged,
        VT_DATE)
    DISP_FUNCTION(CMymfc29CDoc, "ShowWin", ShowWin, VT_EMPTY, VTS_NONE)
    DISP_FUNCTION(CMymfc29CDoc, "CreateAlarm", CreateAlarm,
        VT_DISPATCH, VTS_DATE)
    DISP_FUNCTION(CMymfc29CDoc, "RefreshWin", Refresh, VT_EMPTY,
        VTS_NONE)
    DISP_PROPERTY_PARAM(CMymfc29CDoc, "Figure", GetFigure, SetFigure,
        VT_VARIANT, VTS_I2)
    //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
...
```

The ShowWin() and RefreshWin() member functions aren't very interesting, but the CreateAlarm() method is worth a close look. Here's the corresponding CreateAlarm() member function:

```
LPDISPATCH CMymfc29CDoc::CreateAlarm(DATE time)
{
    TRACE("Entering CMymfc29CDoc::CreateAlarm, time = %f\n", time);
    // OLE deletes any prior CAlarm object
    m_pAlarm = new CAlarm(time);
    return m_pAlarm->GetIDispatch(FALSE);    // no AddRef() here
```

```
}
```

We've chosen to have the component create an alarm object when a controller calls `CreateAlarm()`. `CAlarm` is an **Automation component** class that we've generated with ClassWizard. It is not COM-creatable, which means there's no `IMPLEMENT_OLECREATE` macro and no class factory. The `CreateAlarm()` function constructs a `CAlarm` object and returns an `IDispatch` pointer. The `FALSE` parameter for `CCmdTarget::GetIDispatch` means that the reference count is not incremented; the `CAlarm` object already has a reference count of 1 when it is constructed.

The `CAlarm` class is declared in **Alarm.h** as follows:

```
class CAlarm : public CCmdTarget
{
    DECLARE_DYNAMIC(CAlarm)
public:
    CAlarm(DATE time);

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAlarm)
    public:
    virtual void OnFinalRelease();
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CAlarm();

    // Generated message map functions
    //{{AFX_MSG(CAlarm)
        // NOTE - the ClassWizard will add and remove member
        //  functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
    // Generated OLE dispatch map functions
public:
    //{{AFX_DISPATCH(CAlarm)
    DATE m_time;
    //}}AFX_DISPATCH
    DECLARE_DISPATCH_MAP()
    DECLARE_INTERFACE_MAP()
};
```

Notice the absence of the `DECLARE_DYNCREATE` macro. **Alarm.cpp** contains a dispatch map, as follows:

```
BEGIN_DISPATCH_MAP(CAlarm, CCmdTarget)
    //{{AFX_DISPATCH_MAP(CAlarm)
    DISP_PROPERTY(CAlarm, "Time", m_time, VT_DATE)
    //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

Why do we have a `CAlarm` class? We could have added an `AlarmTime` property in the `CMymfc29CDoc` class instead, but then we would have needed another property or method to turn the alarm on and off. By using the `CAlarm` class, what we're really doing is setting ourselves up to support multiple alarms, a collection of alarms.

To implement an **Automation collection**, we would write another class, CAlarms, which would contain the methods Add, Remove, and Item. Add and Remove are self-explanatory; Item returns an IDispatch pointer for a collection element identified by an index, numeric, or some other key. We would also implement a read-only Count property that returned the number of elements. The document class (which owns the collection) would have an Alarms method with an optional VARIANT parameter. If the parameter were omitted, the method would return the IDispatch pointer for the collection. If the parameter specified an index, the method would return an IDispatch pointer for the selected alarm.

If we wanted our collection to support the VBA "For Each" syntax, we'd have some more work to do. We'd add an IEnum VARIANT interface to the CAlarms class to enumerate the collection of variants and implement the Next() member function of this interface to step through the collection. Then we'd add a CAlarms method named _NewEnum that returned an IEnumVARIANT interface pointer. If we wanted the collection to be general, we'd allow separate enumerator objects (with an IEnumVARIANT interface) and we'd implement the other IEnumVARIANT functions, Skip(), Reset(), and Clone().

The Figure property is an indexed property, which makes it interesting. The Figure property represents the four figures on the clock face, XII, III, VI, and IX. It's a CString array, so we can use Roman numerals. Here's the declaration in **mymfc29CDoc.h**:

```
public:
    CString m_strFigure[4];
```

And here are the GetFigure() and SetFigure() functions in **mymfc29CDoc.cpp**:

```
VARIANT CMymfc29CDoc::GetFigure(short n)
{
    TRACE("Entering CMymfc29CDoc::GetFigure -- n = %d m_strFigure[n] =
%s\n", n, m_strFigure[n]);
    return COleVariant(m_strFigure[n]).Detach();
}

void CMymfc29CDoc::SetFigure(short n, const VARIANT FAR& newValue)
{
    TRACE("Entering CMymfc29CDoc::SetFigure -- n = %d, vt = %d\n", n,
          newValue.vt);
    COleVariant vaTemp;
    vaTemp.ChangeType(VT_BSTR, (COleVariant*) &newValue);
    m_strFigure[n] = vaTemp.bstrVal; // converts double-to-single
}
```

These functions tie back to the DISP_PROPERTY_PARAM macro in the CMymfc29CDoc dispatch map. The first parameter is the index number, specified as a short integer by the last macro parameter. Property indexes don't have to be integers, and the index can have several components (row and column numbers, for example). The ChangeType() call in SetFigure() is necessary because the controller might otherwise pass numbers instead of strings. You've just seen collection properties and indexed properties. What's the difference? A controller can't add or delete elements of an indexed property, but it can add elements to a collection and it can delete elements from a collection. What draws the clock face? As you might expect, it's the OnDraw() member function of the view class. This function uses GetDocument() to get a pointer to the document object, and then it accesses the document's property data members and method member functions.

The Excel macro code appears below.

```
Dim Clock As Object
Dim Alarm As Object

Sub LoadClock()
    Set Clock = CreateObject("Mymfc29C.Document")
    Range("A3").Select
    n = 0
    Do Until n < 4
        Clock.figure(n) = Selection.Value
        Selection.Offset(0, 1).Range("A1").Select
```

```
            n = n + 1
        Loop
        RefreshClock
        Clock.ShowWin
    End Sub

    Sub RefreshClock()
        Clock.Time = Now()
        Clock.RefreshWin
    End Sub

    Sub CreateAlarm()
        Range("E3").Select
        Set Alarm = Clock.CreateAlarm(Selection.Value)
        RefreshClock
    End Sub

    Sub UnloadClock()
        Set Clock = Nothing
    End Sub
```

Notice the `Set Alarm` statement in the `CreateAlarm` macro. It calls the `CreateAlarm` method to return an `IDispatch` pointer, which is stored in an object variable. If the macro is run a second time, a new alarm is created, but the original one is destroyed because its reference count goes to 0.

You've seen a modal dialog in a DLL (MYMFC29B) and you've seen a main frame window in an EXE (MYMFC29C). Be careful with modal dialogs in EXEs. It's fine to have an About dialog that is invoked directly by the component program, but it isn't a good idea to invoke a modal dialog in an out-of-process component method function. The problem is that once the modal dialog is on the screen, the user can switch back to the client program. MFC clients handle this situation with a special "Server Busy" message box, which appears right away. Excel does something similar, but it waits 30 seconds, and this could confuse the user.

Continue on next module. part 3...

----------------End Automation part 2------------

Further reading and digging:

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