Module 14: Splitter Windows and Multiple Views

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

Splitter Windows and Multiple Views

Except for the MYMFC18 example, each program you've seen so far in this book has had only one view attached to a document. If you've used a Microsoft Windows-based word processor, you know that it's convenient to have two windows open simultaneously on various parts of a document. Both windows might contain normal views, or one window might contain a page layout view and another might contain an outline view.
With the application framework, you can use a **splitter window** or **multiple MDI child windows** to display multiple views. You'll learn about both presentation options here, and you'll see that it's easy to make multiple view objects of the same view class (the normal view) in both cases. It's slightly more difficult, however, to use two or more view classes in the same application (say, the outline view and the page layout view).
This module emphasizes the selection and presentation of multiple views. The examples depend on a document with data initialized in the `OnNewDocument()` function. Look back now to Module 10 for a review of document-view communication.

The Splitter Window

A splitter window appears as a special type of frame window that holds several views in panes. The application can split the window on creation, or the user can split the window by choosing a menu command or by dragging a splitter box on the window's scroll bar. After the window has been split, the user can move the splitter bars with the mouse to adjust the relative sizes of the panes. Splitter windows can be used in both SDI and MDI applications. You can see examples of splitter windows in this module.
An object of class `CSplitterWnd` represents the splitter window. As far as Windows is concerned, a `CSplitterWnd` object is an actual window that fully occupies the frame window (`CFrameWnd` or `CMDIChildWnd`) client area. The view windows occupy the splitter window pane areas. The splitter window does not take part in the command dispatch mechanism. The active view window (in a splitter pane) is connected directly to its frame window.

View Options

When you combine multiview presentation methods with application models, you get a number of permutations. Here are some of them:

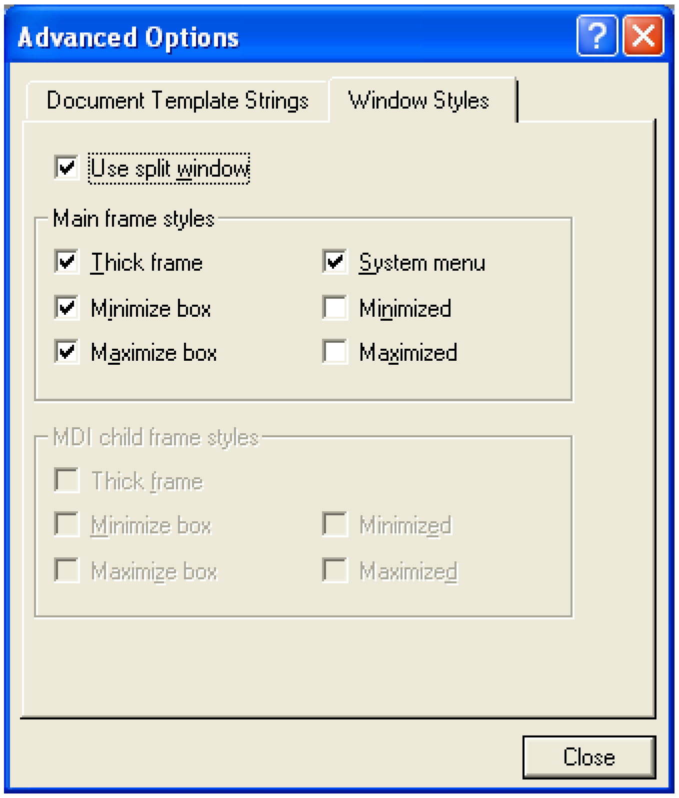
- SDI application with splitter window, single view class. This module's first example, MYMFC21, covers this case. Each splitter window pane can be scrolled to a different part of the document. The programmer determines the maximum number of horizontal and vertical panes; the user makes the split at runtime.
- SDI application with splitter window, multiple view classes. The MYMFC21B example illustrates this case. The programmer determines the number of panes and the sequence of views; the user can change the pane size at runtime.
- SDI application with no splitter windows, multiple view classes. The MYMFC21C example illustrates this case. The user switches view classes by making a selection from a menu.
- MDI application with no splitter windows, single view class. This is the standard MDI application you've seen already in Module 12. The **New Window** menu item lets the user open a new child window for a document that's open already.
- MDI application with no splitter windows, multiple view classes. A small change to the standard MDI application allows the use of multiple views. As example MYMFC21D shows, all that's necessary is to add a menu item and a handler function for each additional view class available.
- MDI application with splitter child windows. This case is covered thoroughly in the online documentation. The **SCRIBBLE** example illustrates the splitting of an MDI child window.

Dynamic and Static Splitter Windows

A **dynamic splitter window** allows the user to split the window at any time by choosing a menu item or by dragging a splitter box located on the scroll bar. The panes in a dynamic splitter window generally use the same view class. The top left pane is initialized to a particular view when the splitter window is created. In a dynamic splitter window, scroll bars are shared among the views. In a window with a single horizontal split, for example, the bottom scroll bar controls both views. A dynamic splitter application starts with a single view object. When the user splits the frame, other view objects are constructed. When the user un-splits the frame, view objects are destroyed.
The panes of a **static splitter window** are defined when the window is first created and they cannot be changed. The user can move the bars but cannot un-split or re-split the window. Static splitter windows can accommodate multiple view classes, with the configuration set at creation time. In a static splitter window, each pane has separate scroll bars. In a static splitter window application, all view objects are constructed when the frame is constructed and they are all destroyed when the frame is destroyed.

The MYMFC21 Example: A Single View Class SDI Dynamic Splitter

In this example, the user can dynamically split the view into **four panes** with **four separate view** objects, all managed by a single view class. We'll use the document and the view code from MYMFC19. AppWizard lets you add a dynamic splitter window to a new application. Create an SDI project. Click the **Advanced** button in the AppWizard Step 4 dialog. Click on the **Window Styles** tab, and select **Use Split Window** as shown here.



[✓] Use split window

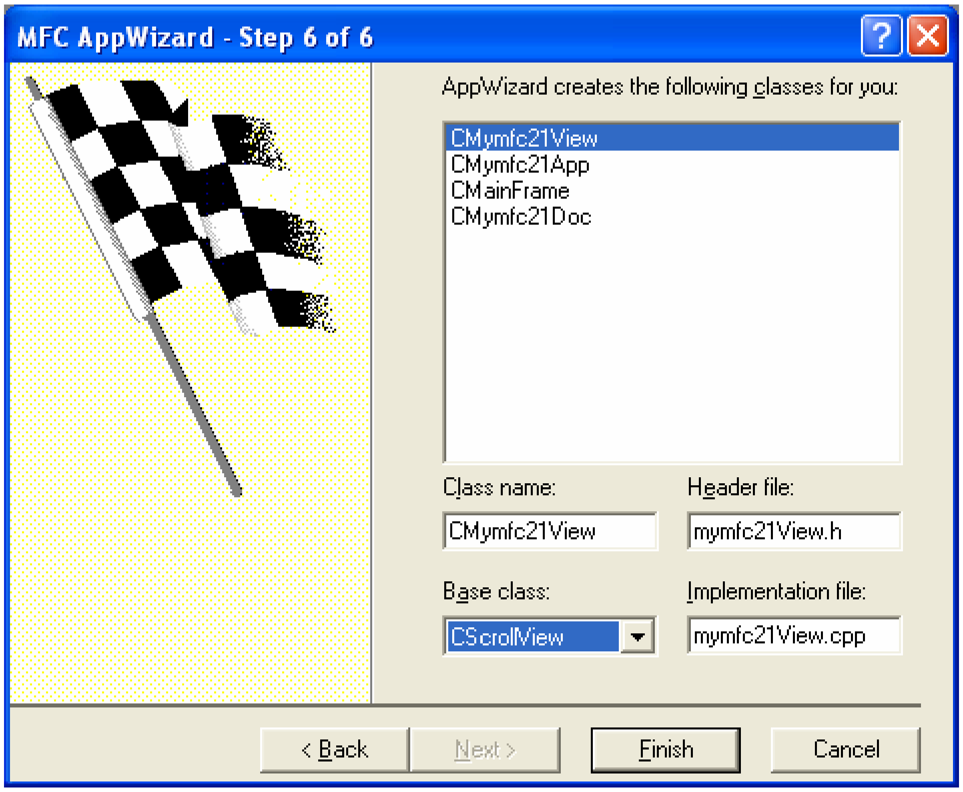
Main frame styles

[✓] Thick frame
[✓] System menu
[✓] Minimize box
[ ] Minimized
[✓] Maximize box
[ ] Maximized

MDI child frame styles

[ ] Thick frame
[ ] Minimize box
[ ] Minimized
[ ] Maximize box
[ ] Maximized

Figure 1: MYMFC21 step 4 of 6 AppWizard, the **Advanced** options settings.

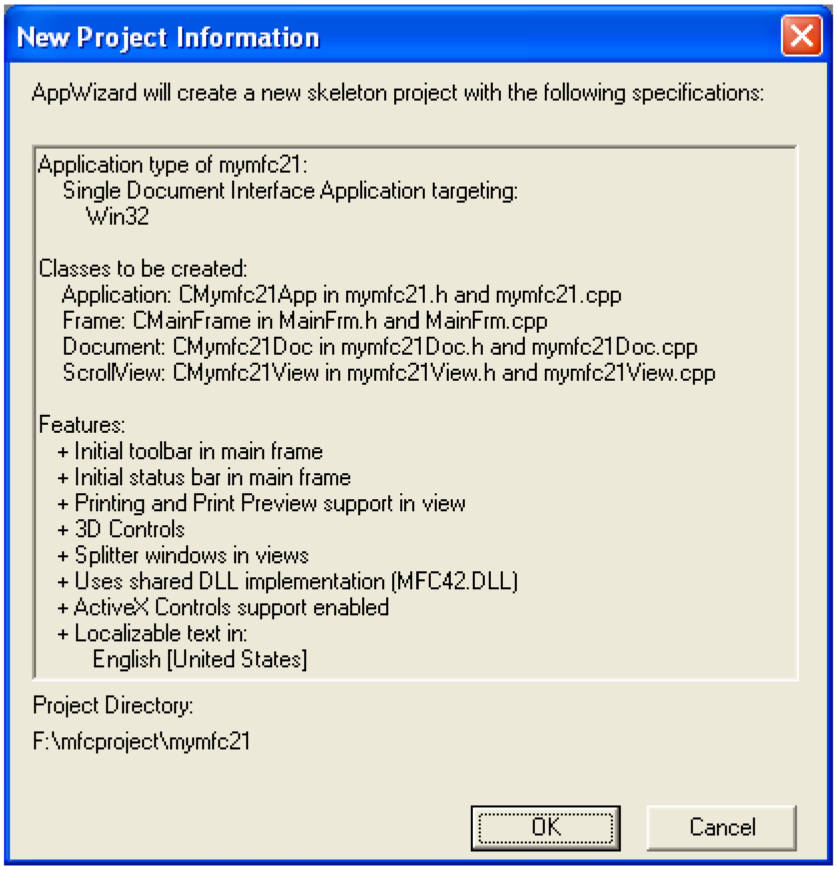


AppWizard creates the following classes for you:

CMymfc21View
CMymfc21App
CMainFrame
CMymfc21Doc

Class name: CMymfc21View
Header file: mymfc21View.h
Base class: CScrollView
Implementation file: mymfc21View.cpp

Figure 2: MYMFC21 step 6 of 6 AppWizard, using `CScrollView` for the view base class.



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

```
Application type of mymfc21:
    Single Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CMymfc21App in mymfc21.h and mymfc21.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    Document: CMymfc21Doc in mymfc21Doc.h and mymfc21Doc.cpp
    ScrollView: CMymfc21View in mymfc21View.h and mymfc21View.cpp

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

Project Directory:

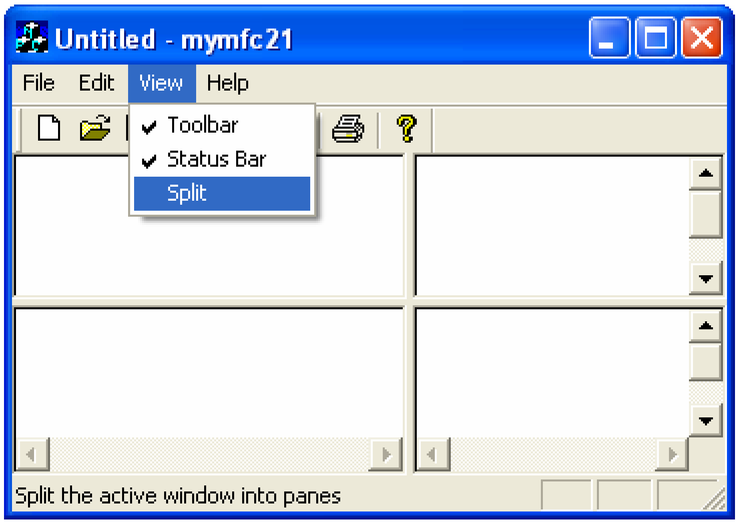
F:\mfcproject\mymfc21

Figure 3: MYMFC21 project summary.

When you check the **Use Split Window** check box, AppWizard adds code to your `CMainFrame` class. Of course, you could add the same code to the `CMainFrame` class of an existing application to add splitter capability.

Resources for Splitting

Build and run MYMFC21 program. When AppWizard generates an application with a splitter frame, it includes a **Split** option in the project's **View** menu when you build and run the program. The `ID_WINDOW_SPLIT` command ID is mapped in the `CView` class within the MFC library.



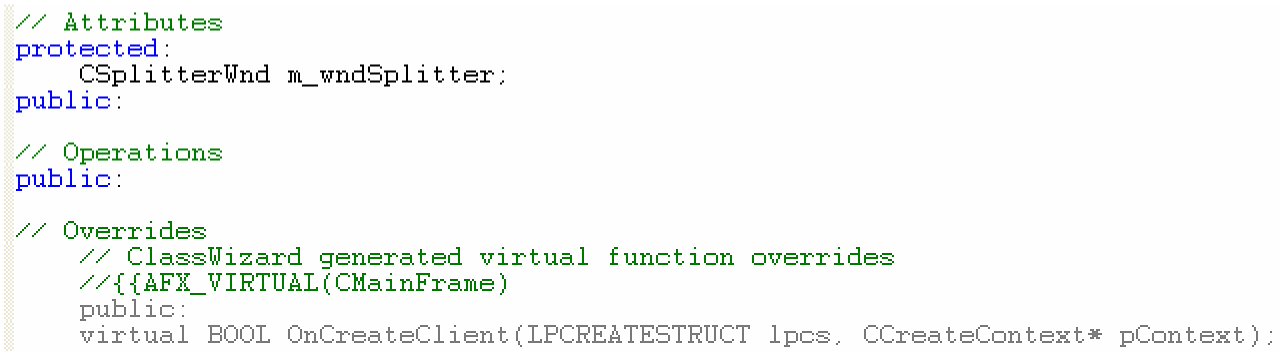
Split the active window into panes

Figure 4: MYMFC21 program output with split windows without any coding.

CMainFrame Class

The application's main frame window class needs a splitter window data member and a prototype for an overridden `OnCreateClient()` function. Here are the additions that AppWizard makes to the **MainFrm.**h file:

```
protected:
    CSplitterWnd m_wndSplitter;

public:
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
```

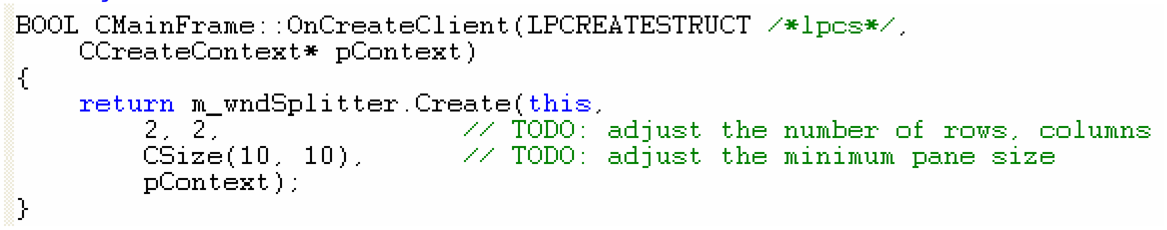


```
// Attributes
protected:
    CSplitterWnd m_wndSplitter;
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    public:
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
```

Listing 1.

The application framework calls the `CFrameWnd::OnCreateClient` virtual member function when the frame object is created. The base class version creates a single view window as specified by the document template. The AppWizard-generated `OnCreateClient()` override shown here (in **MainFrm.cpp**) creates a splitter window instead, and the splitter window creates the first view:

```
BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT /*lpcs*/,
    CCreateContext* pContext)
{
    return m_wndSplitter.Create( this,
        2, 2,           // TODO: adjust the number of rows, columns
        CSize(10, 10),  // TODO: adjust the minimum pane size
        pContext);
}
```



```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT /*lpcs*/,
    CCreateContext* pContext)
{
    return m_wndSplitter.Create(this,
        2, 2,               // TODO: adjust the number of rows, columns
        CSize(10, 10),      // TODO: adjust the minimum pane size
        pContext);
}
```

Listing 2.

The `CSplitterWnd Create()` member function creates a dynamic splitter window, and the `CSplitterWnd` object knows the view class because its name is embedded in the `CCreateContext` structure that's passed as a parameter to `Create()`. The second and third `Create()` parameters (2, 2) specify that the window can be split into a maximum of two rows and two columns. If you changed the parameters to (2, 1), you would allow only a single horizontal split. The parameters (1, 2) allow only a single vertical split. The `CSize` parameter specifies the minimum pane size.

The MYMFC20 steps

Before we see the action, let put in the MYMFC20 steps in this example. Add a `CStringArray` data member to the `CMymfc21Doc` class. Edit the **Mymfc21Doc.h** header file or use ClassView.

```
public:
    CStringArray m_stringArray;
```

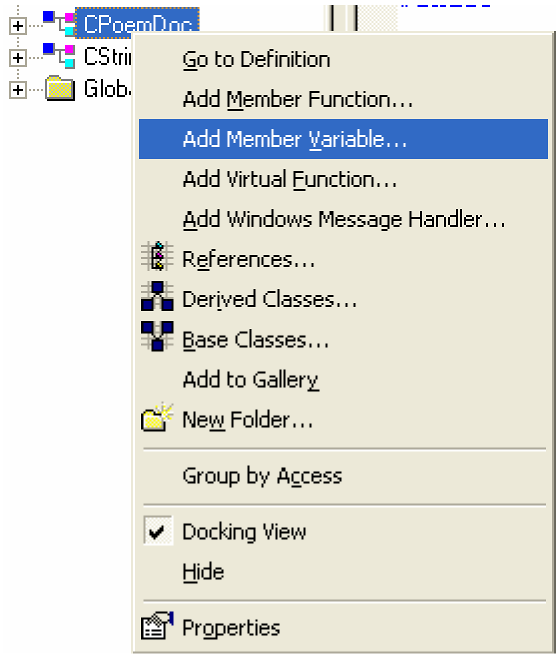
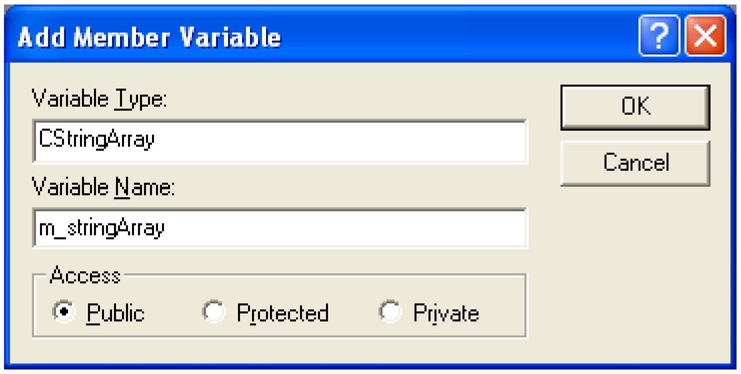


Figure 5: Adding member variable to `CMymfc21Doc` class using ClassView.



Variable Type: CStringArray
Variable Name: m_stringArray
Access: [✓ Public] Protected Private

Figure 6: Entering the member variable's type and name.

```
// Implementation
public:
    CStringArray m_stringArray;
```

Listing 3.

The document data is stored in a string array. The MFC library `CStringArray` class holds an array of `CString` objects, accessible by a zero-based subscript. You need not set a maximum dimension in the declaration because the array is dynamic.
Add a `CRect` data member to the `CStringView` class. **Edit** the **StringView.h** header file or use `ClassView`:

```
private:
    CRect m_rectPrint;
```

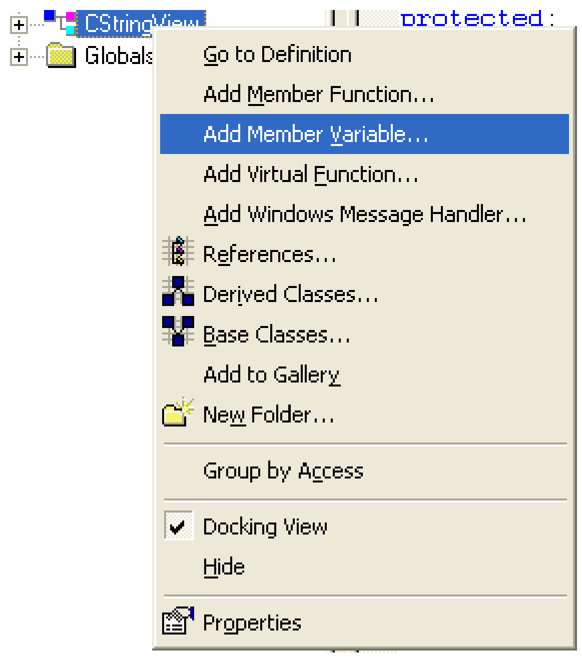
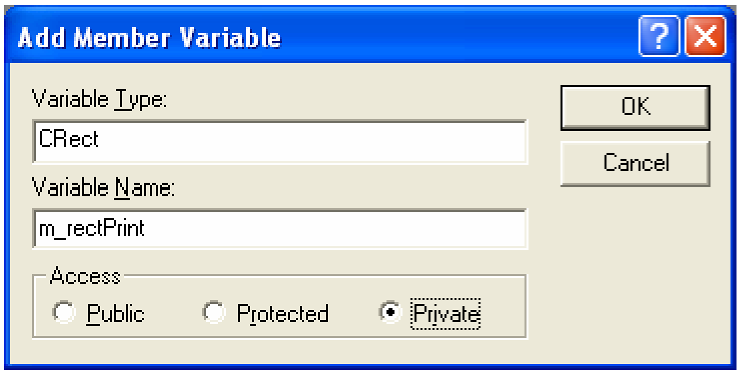


Figure 7: Using ClassView to add a `CRect` data member to the `CStringView` class.



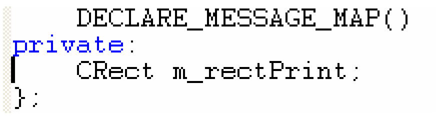
Add Member Variable

Variable Type: CRect

Variable Name: m_rectPrint

Access: ( ) Public ( ) Protected [✓ Private]

Figure 8: Entering the member variable type and name.



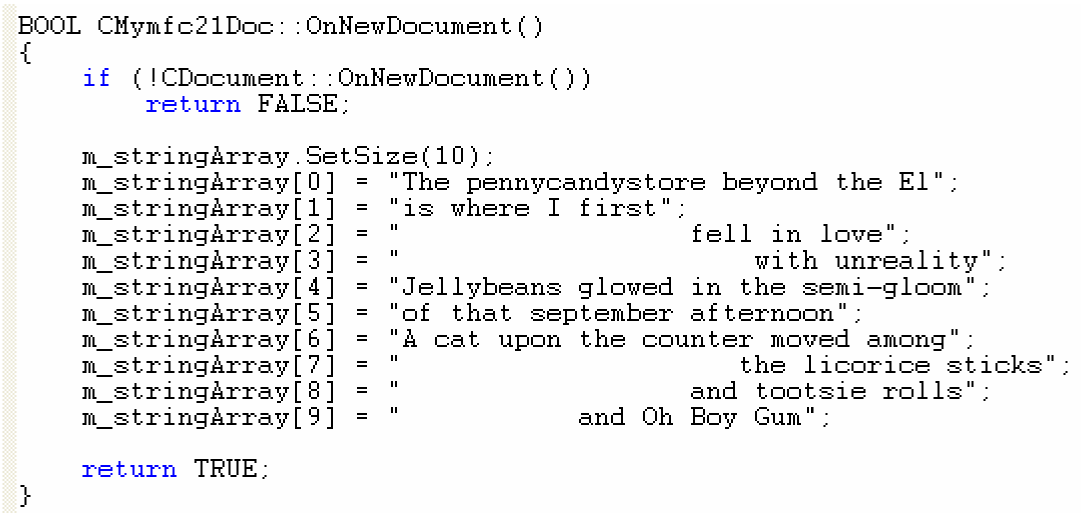
```
    DECLARE_MESSAGE_MAP()
private:
    CRect m_rectPrint;
};
```

Listing 4.

Edit three `CMymfc21Doc` member functions in the file **Mymfc21Doc.cpp**. AppWizard generated skeleton `OnNewDocument()` and `Serialize()` functions, but we'll have to use ClassWizard to override the `DeleteContents()` function. We'll initialize the poem document in the overridden `OnNewDocument()` function. `DeleteContents()` is called in `CDocument::OnNewDocument`, so by calling the base class function first we're sure the poem won't be deleted. The text, by the way, is an excerpt from the twentieth poem in Lawrence Ferlinghetti's book A Coney Island of the Mind. Type 10 lines of your choice. You can substitute another poem or maybe your favorite Win32 function description. Add the following code:

```
BOOL CMymfc21Doc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    m_stringArray.SetSize(10);
```

```
        m_stringArray[0] = "The pennycandystore beyond the El";
        m_stringArray[1] = "is where I first";
        m_stringArray[2] = "                  fell in love";
        m_stringArray[3] = "                      with unreality";
        m_stringArray[4] = "Jellybeans glowed in the semi-gloom";
        m_stringArray[5] = "of that september afternoon";
        m_stringArray[6] = "A cat upon the counter moved among";
        m_stringArray[7] = "                     the licorice sticks";
        m_stringArray[8] = "                  and tootsie rolls";
        m_stringArray[9] = "           and Oh Boy Gum";

        return TRUE;
    }
```

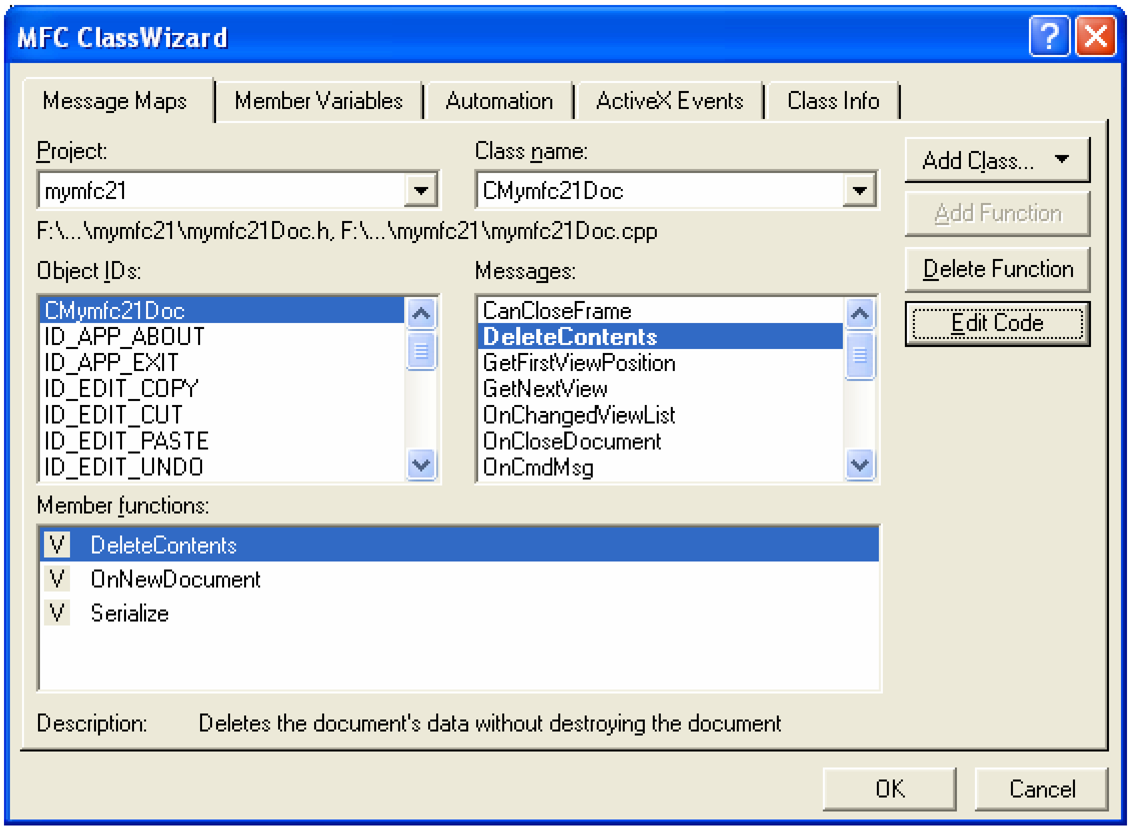


```
BOOL CMymfc21Doc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    m_stringArray.SetSize(10);
    m_stringArray[0] = "The pennycandystore beyond the El";
    m_stringArray[1] = "is where I first";
    m_stringArray[2] = "                  fell in love";
    m_stringArray[3] = "                      with unreality";
    m_stringArray[4] = "Jellybeans glowed in the semi-gloom";
    m_stringArray[5] = "of that september afternoon";
    m_stringArray[6] = "A cat upon the counter moved among";
    m_stringArray[7] = "                     the licorice sticks";
    m_stringArray[8] = "                  and tootsie rolls";
    m_stringArray[9] = "           and Oh Boy Gum";

    return TRUE;
}
```

Listing 5.

The `CStringArray` class supports dynamic arrays, but here we're using the `m_stringArray` object as though it were a static array of 10 elements. The application framework calls the document's virtual `DeleteContents()` function when it closes the document; this action deletes the strings in the array. A `CStringArray` contains actual objects, and a `CObArray` contains pointers to objects. This distinction is important when it's time to delete the array elements. Here the `RemoveAll()` function actually deletes the string objects:

```
    void CMymfc21Doc::DeleteContents()
    {
        // called before OnNewDocument() and when document is closed
        m_stringArray.RemoveAll();
    }
```



Project: mymfc21

Class name: CMymfc21Doc

F:\...\mymfc21\mymfc21Doc.h, F:\...\mymfc21\mymfc21Doc.cpp

Object IDs:

```
CMymfc21Doc
ID_APP_ABOUT
ID_APP_EXIT
ID_EDIT_COPY
ID_EDIT_CUT
ID_EDIT_PASTE
ID_EDIT_UNDO
```

Messages:

```
CanCloseFrame
DeleteContents
GetFirstViewPosition
GetNextView
OnChangedViewList
OnCloseDocument
OnCmdMsg
```

Member functions:

```
V   DeleteContents
V   OnNewDocument
V   Serialize
```

Description: Deletes the document's data without destroying the document

Figure 10: Adding the `RemoveAll()` function to the `CMymfc21Doc` class.

```
// CMymfc21Doc commands

void CMymfc21Doc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
// called before OnNewDocument() and when document is closed
    m_stringArray.RemoveAll();
}
```

Listing 6.

Serialization isn't important in this example, but the following function illustrates how easy it is to serialize strings. The application framework calls the `DeleteContents()` function before loading from the archive, so you don't have to worry about emptying the array. Add the following code:

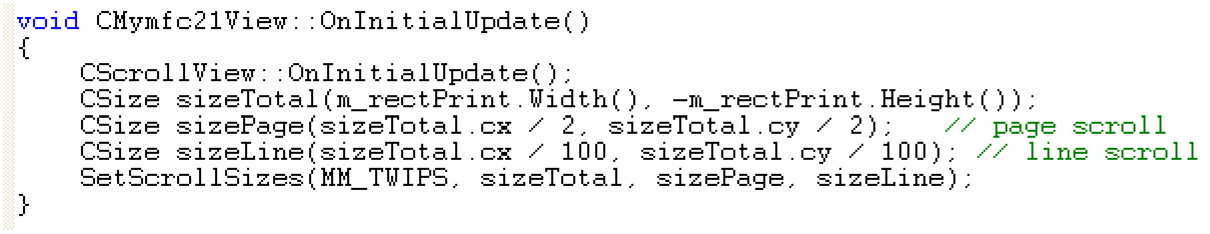
```
void CMymfc21Doc::Serialize(CArchive& ar)
{
    m_stringArray.Serialize(ar);
}
```

```
// CMymfc21Doc serialization

void CMymfc21Doc::Serialize(CArchive& ar)
{
    m_stringArray.Serialize(ar);
}
```

Listing 7.

Edit the `OnInitialUpdate()` function in **mymfc21View.cpp**. You must override the function for all classes derived from `CScrollView`. This function's job is to set the logical window size and the mapping mode. Add the following code:

```
void CMymfc21View::OnInitialUpdate()
{
       CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);
}
```



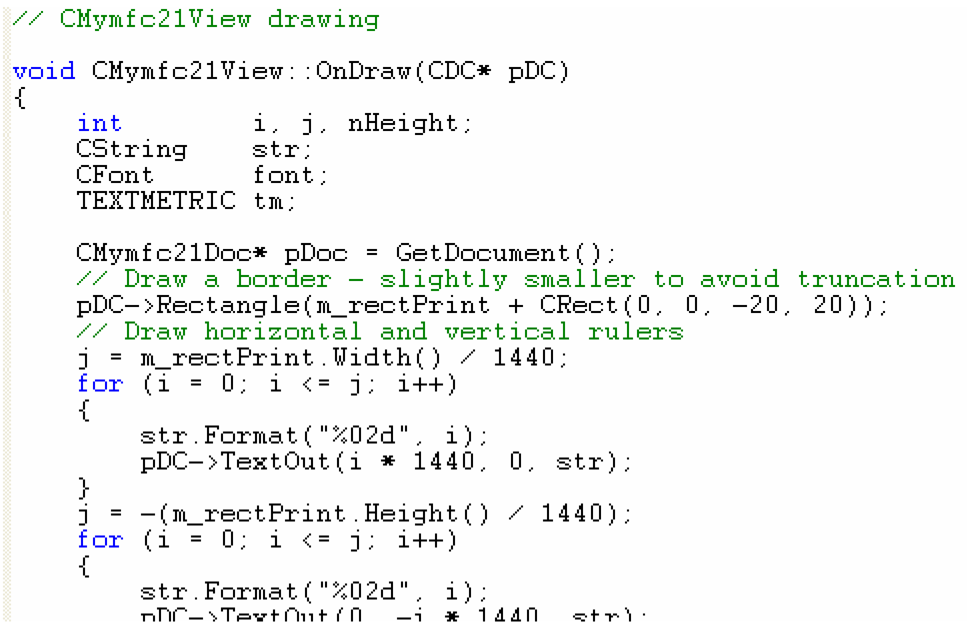
```
void CMymfc21View::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);
}
```

Listing 8.

Edit the `OnDraw()` function in **mymfc21View.cpp**. The `OnDraw()` function of class `CMymfc21View` draws on both the display and the printer. In addition to displaying the poem text lines in 10-point roman font, it draws a border around the printable area and a crude ruler along the top and left margins. `OnDraw()` assumes the `MM_TWIPS` mapping mode, in which 1 inch = 1440 units. Add the code shown below.

```
void CMymfc21View::OnDraw(CDC* pDC)
{
    int        i, j, nHeight;
    CString    str;
    CFont      font;
    TEXTMETRIC tm;

    CMymfc21Doc* pDoc = GetDocument();
    // Draw a border - slightly smaller to avoid truncation
    pDC->Rectangle(m_rectPrint + CRect(0, 0, -20, 20));
    // Draw horizontal and vertical rulers
    j = m_rectPrint.Width() / 1440;
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(i * 1440, 0, str);
    }
    j = -(m_rectPrint.Height() / 1440);
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(0, -i * 1440, str);
    }
    // Print the poem 0.5 inch down and over;
    // use 10-point roman font
    font.CreateFont(-200, 0, 0, 0, 400, FALSE, FALSE, 0, ANSI_CHARSET,
                    OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
                    DEFAULT_QUALITY, DEFAULT_PITCH | FF_ROMAN, "Times New Roman");
    CFont* pOldFont = (CFont*) pDC->SelectObject(&font);
    pDC->GetTextMetrics(&tm);
    nHeight = tm.tmHeight + tm.tmExternalLeading;
    TRACE("font height = %d, internal leading = %d\n", nHeight,
tm.tmInternalLeading);
    j = pDoc->m_stringArray.GetSize();
    for (i = 0; i < j; i++)
```

```
        {
            pDC->TextOut(720, -i * nHeight - 720, pDoc->m_stringArray[i]);
        }
        pDC->SelectObject(pOldFont);
        TRACE("LOGPIXELSX = %d, LOGPIXELSY = %d\n", pDC->GetDeviceCaps(LOGPIXELSX),
              pDC->GetDeviceCaps(LOGPIXELSY));
        TRACE("HORZSIZE = %d, VERTSIZE = %d\n", pDC->GetDeviceCaps(HORZSIZE),
              pDC->GetDeviceCaps(VERTSIZE));

    }
```

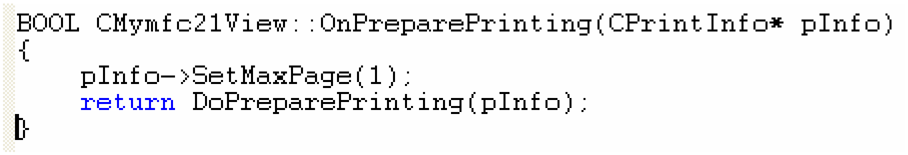


```
// CMymfc21View drawing

void CMymfc21View::OnDraw(CDC* pDC)
{
    int         i, j, nHeight;
    CString     str;
    CFont       font;
    TEXTMETRIC  tm;

    CMymfc21Doc* pDoc = GetDocument();
    // Draw a border - slightly smaller to avoid truncation
    pDC->Rectangle(m_rectPrint + CRect(0, 0, -20, 20));
    // Draw horizontal and vertical rulers
    j = m_rectPrint.Width() / 1440;
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(i * 1440, 0, str);
    }
    j = -(m_rectPrint.Height() / 1440);
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(0, -i * 1440, str);
```

Listing 9.

Edit the `OnPreparePrinting()` function in **mymfc21View.cpp**. This function sets the maximum number of pages in the print job. This example has only one page. It's absolutely necessary to call the base class `DoPreparePrinting()` function in your overridden `OnPreparePrinting()` function. Add the following code:

```
BOOL CMymfc21View::OnPreparePrinting(CPrintInfo* pInfo)
{
        pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```



```
BOOL CMymfc21View::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```

Listing 10.

Edit the constructor in **mymfc21View.cpp**. The initial value of the print rectangle should be 8-by-15 inches, expressed in twips (1 inch = 1440 twips). Add the following code:

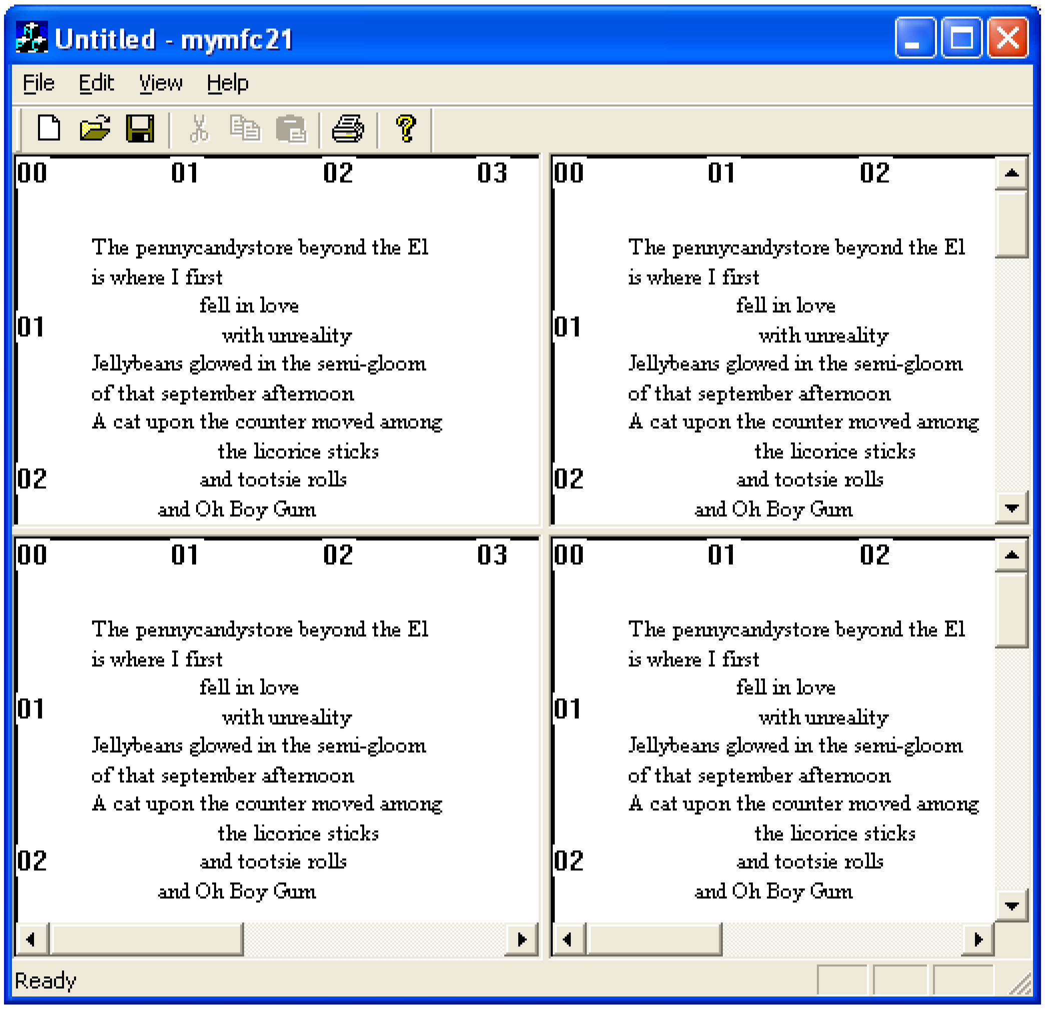
```
CMymfc21View::CMymfc21View() : m_rectPrint(0, 0, 11520, -15120)
{
        // TODO: add construction code here
}
```

```
CMymfc21View::CMymfc21View() : m_rectPrint(0, 0, 11520, -15120)
{
    // TODO: add construction code here
}
```

Listing 11.

Testing the MYMFC21 Application

When the application starts, you can split the window by choosing **Split** from the **View** menu or by dragging the splitter boxes at the left and top of the scroll bars. Figure 11 shows a typical single view window with a four-way split. Multiple views share the scroll bars.



```
00          01          02          03
      The pennycandystore beyond the El
      is where I first
                fell in love
                  with unreality
01    Jellybeans glowed in the semi-gloom
      of that september afternoon
      A cat upon the counter moved among
                  the licorice sticks
02              and tootsie rolls
             and Oh Boy Gum
```

```
00          01          02
      The pennycandystore beyond the El
      is where I first
                fell in love
                  with unreality
01    Jellybeans glowed in the semi-gloom
      of that september afternoon
      A cat upon the counter moved among
                  the licorice sticks
02              and tootsie rolls
             and Oh Boy Gum
```

```
00          01          02          03
      The pennycandystore beyond the El
      is where I first
                fell in love
                  with unreality
01    Jellybeans glowed in the semi-gloom
      of that september afternoon
      A cat upon the counter moved among
                  the licorice sticks
02              and tootsie rolls
             and Oh Boy Gum
```

```
00          01          02
      The pennycandystore beyond the El
      is where I first
                fell in love
                  with unreality
01    Jellybeans glowed in the semi-gloom
      of that september afternoon
      A cat upon the counter moved among
                  the licorice sticks
02              and tootsie rolls
             and Oh Boy Gum
```

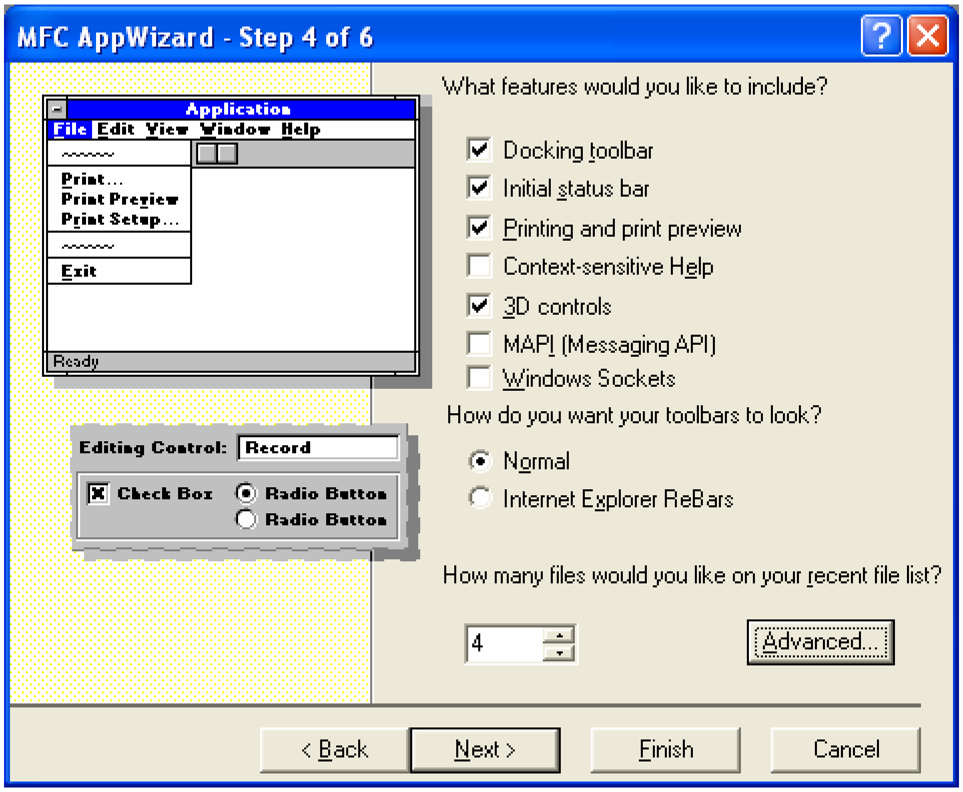
Figure 11: MYMFC21 output, a single view window with a four-way split.

The MYMFC21B Example: A Double View Class SDI Static Splitter

In MYMFC21B, we'll extend MYMFC21 by defining a second view class and allowing a static splitter window to show the two views. (The **HexView.h** and **HexView.cpp** files are cloned from the original view class.) This time the splitter window works a little differently. Instead of starting off as a single pane, the splitter is initialized with two panes. The user can move the bar between the panes by dragging it with the mouse or by choosing the Window **Split** menu item. The easiest way to generate a static splitter application is to let AppWizard generate a dynamic splitter application and then edit the generated `CMainFrame::OnCreateClient` function.

The steps

As usual, begin with the Visual C++ AppWizard, creating an SDI application.


MFC AppWizard - Step 4 of 6

What features would you like to include?

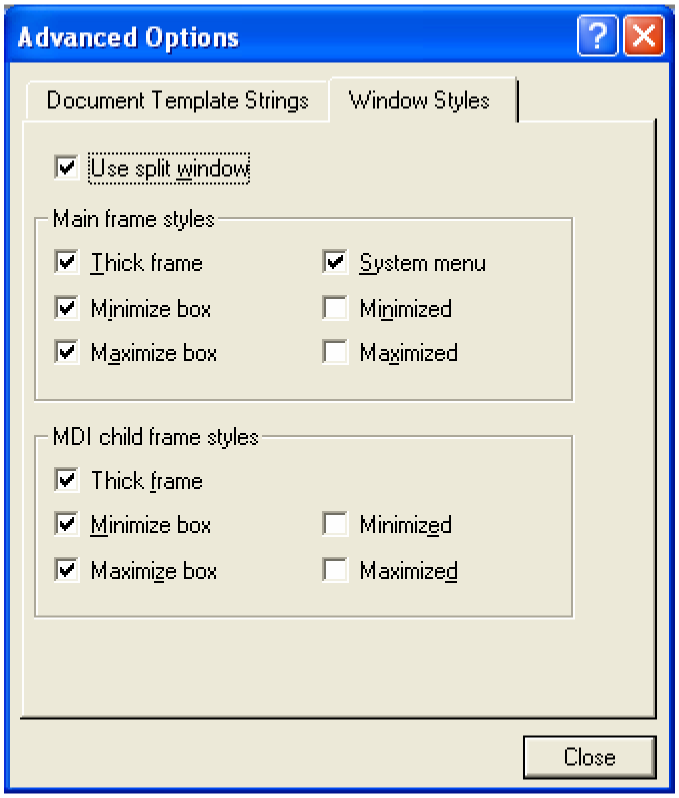
- [✓] Docking toolbar
- [✓] Initial status bar
- [✓] Printing and print preview
- [ ] Context-sensitive Help
- [✓] 3D controls
- [ ] MAPI (Messaging API)
- [ ] Windows Sockets

How do you want your toolbars to look?

- (●) Normal
- ( ) Internet Explorer ReBars

How many files would you like on your recent file list? 4

Figure 12: Step 4 of 6 AppWizard for MYMFC21B, invoking the **Advanced** options.


Advanced Options

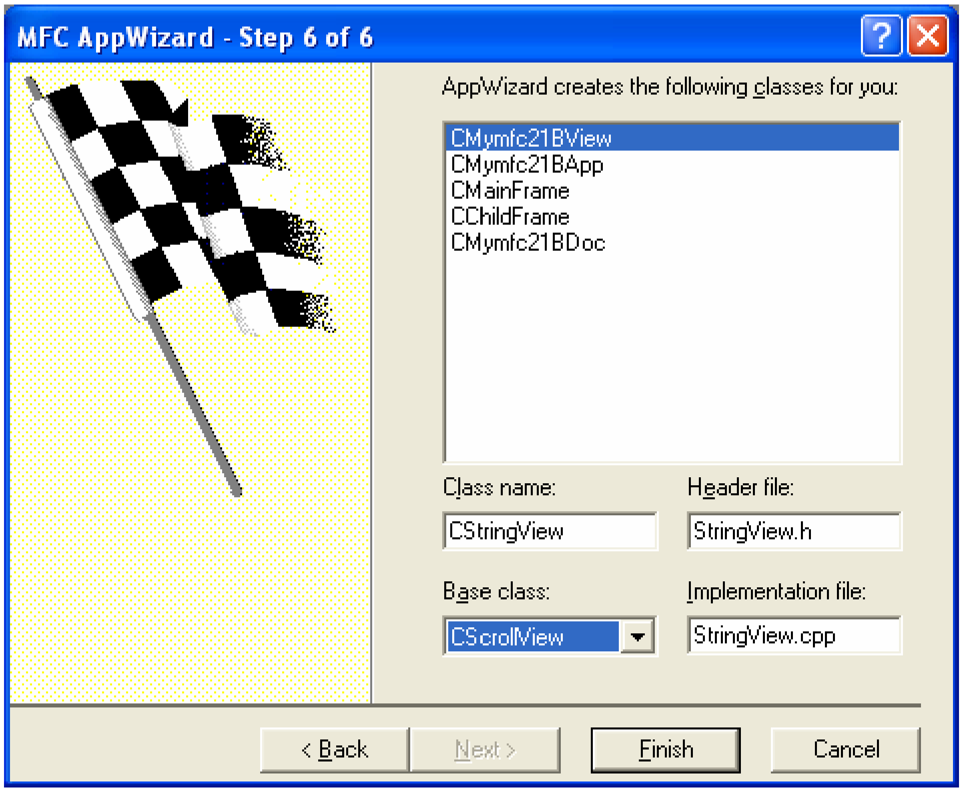
Document Template Strings | Window Styles

- [✓] Use split window

Main frame styles
- [✓] Thick frame
- [✓] System menu
- [✓] Minimize box
- [ ] Minimized
- [✓] Maximize box
- [ ] Maximized

MDI child frame styles
- [✓] Thick frame
- [✓] Minimize box
- [ ] Minimized
- [✓] Maximize box
- [ ] Maximized

Figure 13: Using the split window.


MFC AppWizard - Step 6 of 6

AppWizard creates the following classes for you:

CMymfc21BView
CMymfc21BApp
CMainFrame
CChildFrame
CMymfc21BDoc

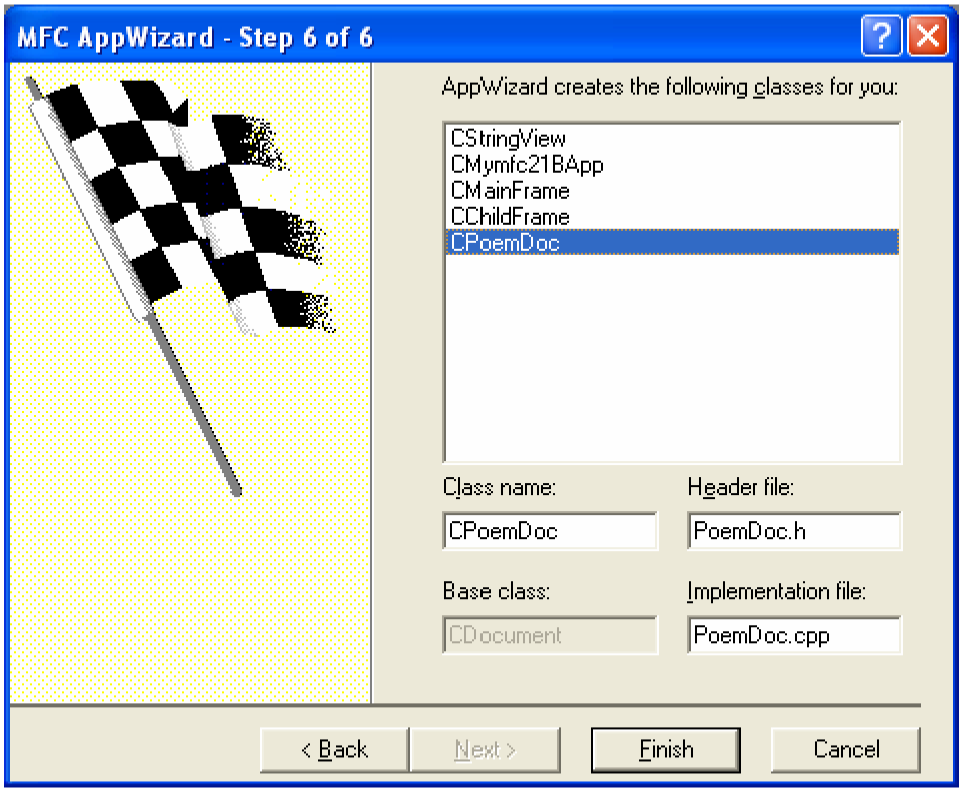
Class name: CStringView

Header file: StringView.h

Base class: CScrollView

Implementation file: StringView.cpp

Figure 14: Step 6 of 6, renaming the view files, class and using `CScrollView` as the base view class.


MFC AppWizard - Step 6 of 6

AppWizard creates the following classes for you:

CStringView
CMymfc21BApp
CMainFrame
CChildFrame
CPoemDoc

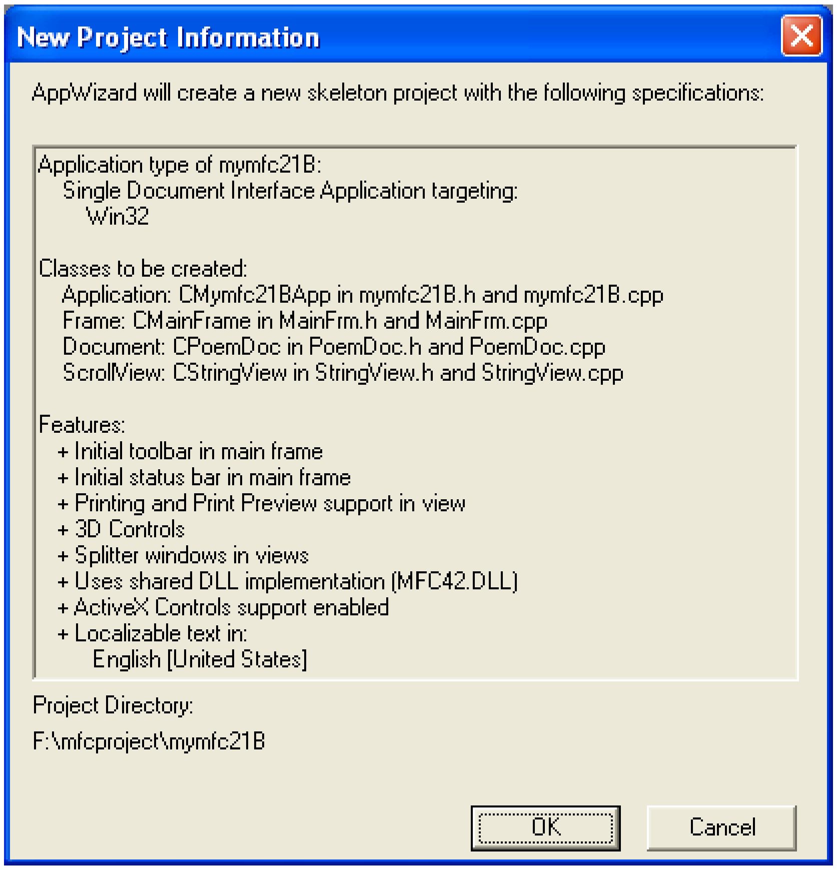
Class name: CPoemDoc

Header file: PoemDoc.h

Base class: CDocument

Implementation file: PoemDoc.cpp

Figure 15: Step 6 of 6, renaming the document files and class.



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

```
Application type of mymfc21B:
    Single Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CMymfc21BApp in mymfc21B.h and mymfc21B.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    Document: CPoemDoc in PoemDoc.h and PoemDoc.cpp
    ScrollView: CStringView in StringView.h and StringView.cpp

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

Project Directory:

F:\mfcproject\mymfc21B

Figure 16: MYMFC21B project summary.

Note that this is an SDI application. Add a `CStringArray` data member to the `CPoemDoc` class. Edit the **PoemDoc.h** header file or use ClassView.

```
public:
    CStringArray m_stringArray;
```

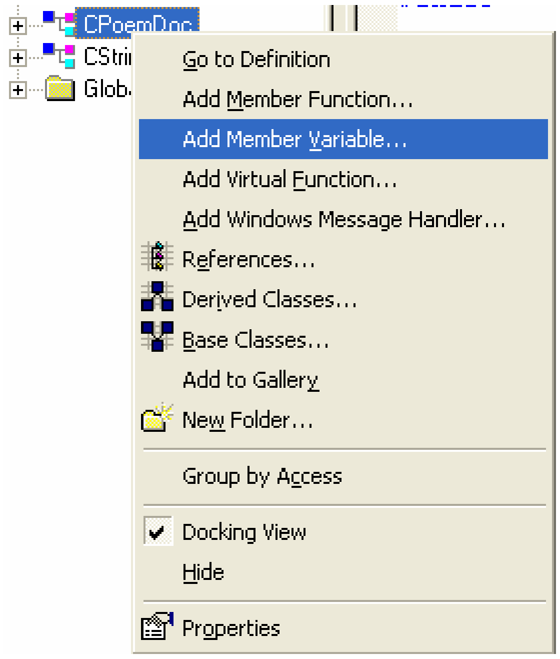
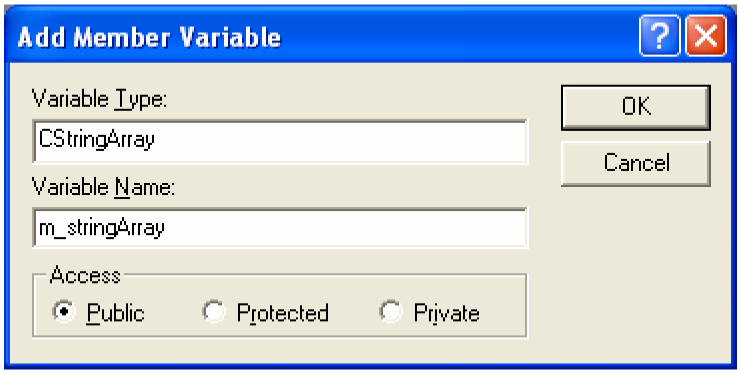


Figure 17: Using ClassView to add a `CStringArray` data member to the `CPoemDoc` class.



Variable Type: CStringArray

Variable Name: m_stringArray

Access: [✓ Public] Protected Private

Figure 18: Entering the member variable type and name.

```
// Implementation
public:
    CStringArray m_stringArray;
    virtual ~CPoemDoc();
#ifdef _DEBUG
```

Listing 12.

The document data is stored in a string array. The MFC library `CStringArray` class holds an array of `CString` objects, accessible by a zero-based subscript. You need not set a maximum dimension in the declaration because the array is dynamic.
Add a `CRect` data member to the `CStringView` class. Edit the **StringView.h** header file or use ClassView:

```
private:
    CRect m_rectPrint;
```

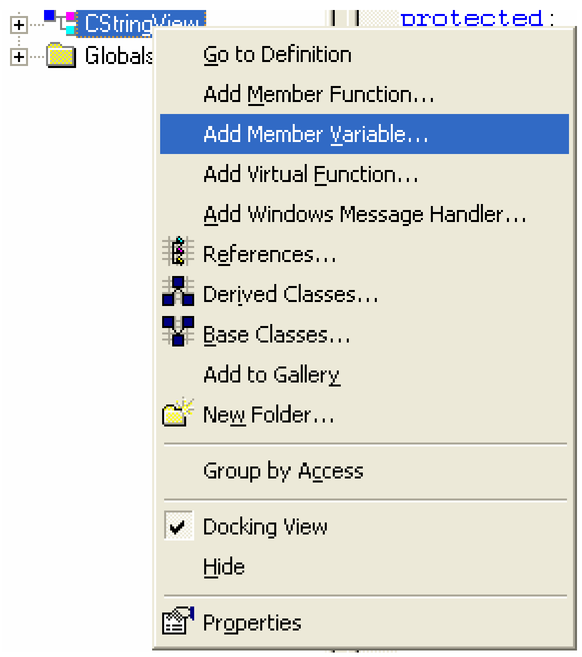
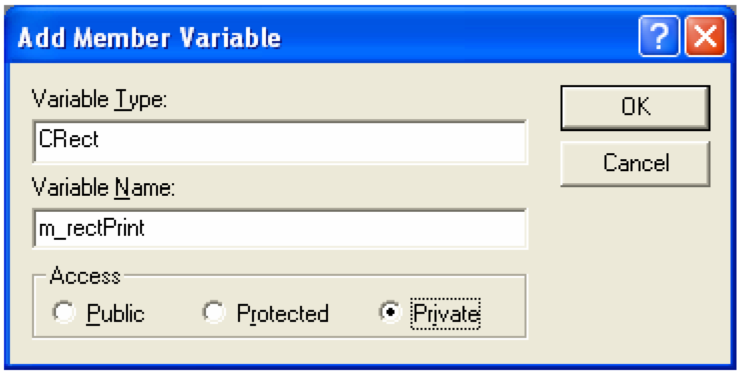


Figure 19: Using ClassView to add a `CRect` data member to the `CStringView` class.

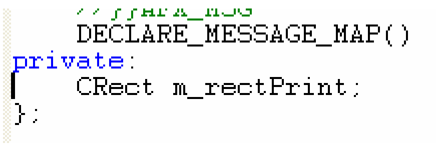


Variable Type: CRect

Variable Name: m_rectPrint

Access: Public [✓ Private] Protected

Figure 20: Entering the member variable type and name.

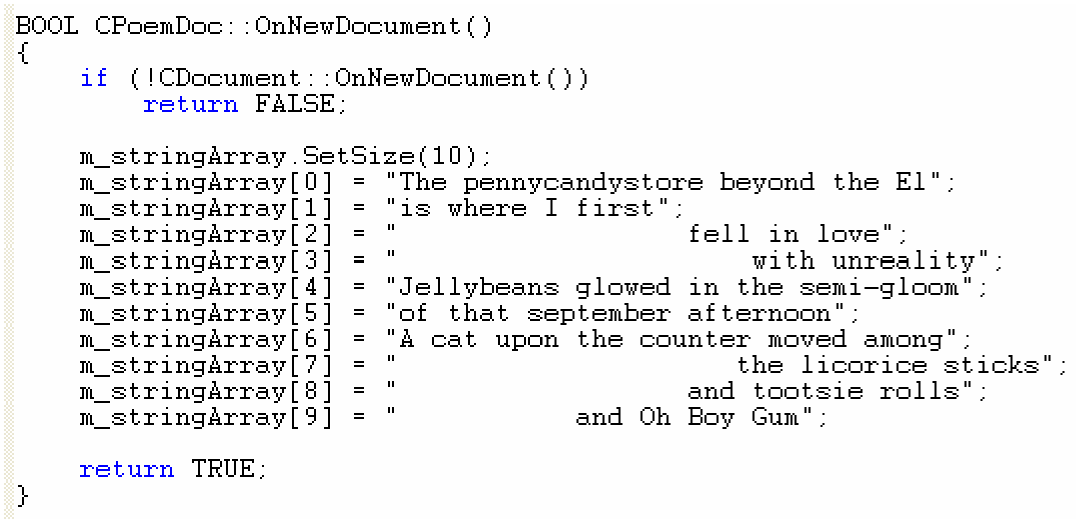


```
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CRect m_rectPrint;
};
```

Listing 13.

Edit three `CPoemDoc` member functions in the file **PoemDoc.cpp**. AppWizard generated skeleton `OnNewDocument()` and `Serialize()` functions, but we'll have to use ClassWizard to override the `DeleteContents()` function. We'll initialize the poem document in the overridden `OnNewDocument()` function. `DeleteContents()` is called in `CDocument::OnNewDocument,` so by calling the base class function first we're sure the poem won't be deleted. The text, by the way, is an excerpt from the twentieth poem in Lawrence Ferlinghetti's book A Coney Island of the Mind. Type 10 lines of your choice. You can substitute another poem or maybe your favorite Win32 function description. Add the following code:

```
BOOL CPoemDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    m_stringArray.SetSize(10);
    m_stringArray[0] = "The pennycandystore beyond the El";
    m_stringArray[1] = "is where I first";
    m_stringArray[2] = "                  fell in love";
    m_stringArray[3] = "                      with unreality";
    m_stringArray[4] = "Jellybeans glowed in the semi-gloom";
    m_stringArray[5] = "of that september afternoon";
    m_stringArray[6] = "A cat upon the counter moved among";
    m_stringArray[7] = "                     the licorice sticks";
    m_stringArray[8] = "                  and tootsie rolls";
    m_stringArray[9] = "           and Oh Boy Gum";

    return TRUE;
}
```

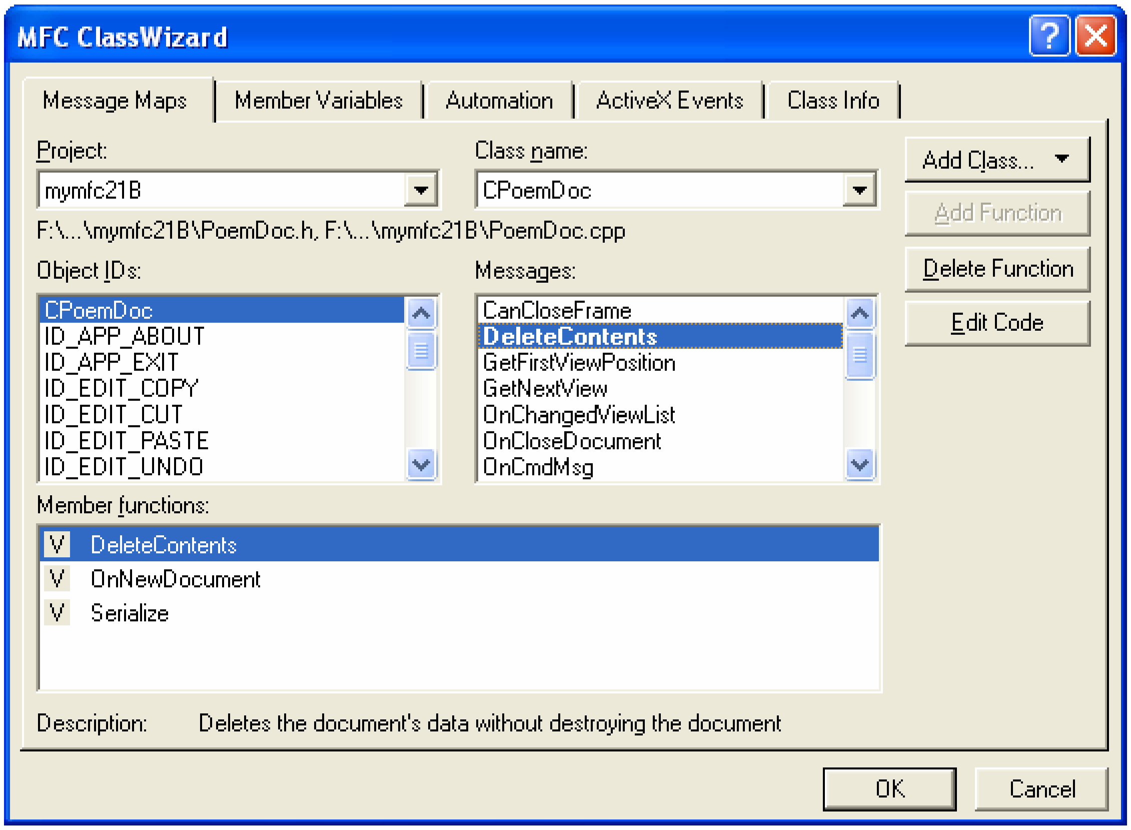


```
BOOL CPoemDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    m_stringArray.SetSize(10);
    m_stringArray[0] = "The pennycandystore beyond the El";
    m_stringArray[1] = "is where I first";
    m_stringArray[2] = "                  fell in love";
    m_stringArray[3] = "                      with unreality";
    m_stringArray[4] = "Jellybeans glowed in the semi-gloom";
    m_stringArray[5] = "of that september afternoon";
    m_stringArray[6] = "A cat upon the counter moved among";
    m_stringArray[7] = "                     the licorice sticks";
    m_stringArray[8] = "                  and tootsie rolls";
    m_stringArray[9] = "           and Oh Boy Gum";

    return TRUE;
}
```

Listing 14.

The `CStringArray` class supports dynamic arrays, but here we're using the `m_stringArray` object as though it were a static array of 10 elements. The application framework calls the document's virtual `DeleteContents()` function when it closes the document; this action deletes the strings in the array. A `CStringArray` contains actual objects, and a `CObArray` contains pointers to objects. This distinction is important when it's time to delete the array elements. Here, the `RemoveAll()` function actually deletes the string objects:

```
    void CPoemDoc::DeleteContents()
    {
        // called before OnNewDocument() and when document is closed
        m_stringArray.RemoveAll();
    }
```



Project: mymfc21B

Class name: CPoemDoc

F:\...\mymfc21B\PoemDoc.h, F:\...\mymfc21B\PoemDoc.cpp

Object IDs: CPoemDoc, ID_APP_ABOUT, ID_APP_EXIT, ID_EDIT_COPY, ID_EDIT_CUT, ID_EDIT_PASTE, ID_EDIT_UNDO

Messages: CanCloseFrame, DeleteContents, GetFirstViewPosition, GetNextView, OnChangedViewList, OnCloseDocument, OnCmdMsg

Member functions:
V DeleteContents
V OnNewDocument
V Serialize

Description: Deletes the document's data without destroying the document

Figure 21: As in MYMFC21, adding the `RemoveAll()` function to the document class.

```
void CPoemDoc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    // called before OnNewDocument() and when document is closed
    m_stringArray.RemoveAll();
}
```

Listing 15.

Serialization isn't important in this example, but the following function illustrates how easy it is to serialize strings. The application framework calls the `DeleteContents()` function before loading from the archive, so you don't have to worry about emptying the array. Add the following code:

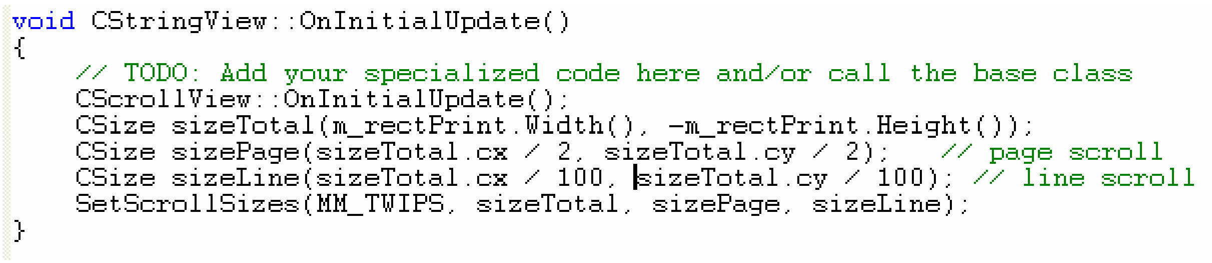
```
void CPoemDoc::Serialize(CArchive& ar)
{
    m_stringArray.Serialize(ar);
}
```

```
void CPoemDoc::Serialize(CArchive& ar)
{
    m_stringArray.Serialize(ar);
}
```

Listing 16.

Edit the `OnInitialUpdate()` function in **StringView.cpp**. You must override the function for all classes derived from `CScrollView`. This function's job is to set the logical window size and the mapping mode. Add the following code:

```
void CStringView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);
}
```



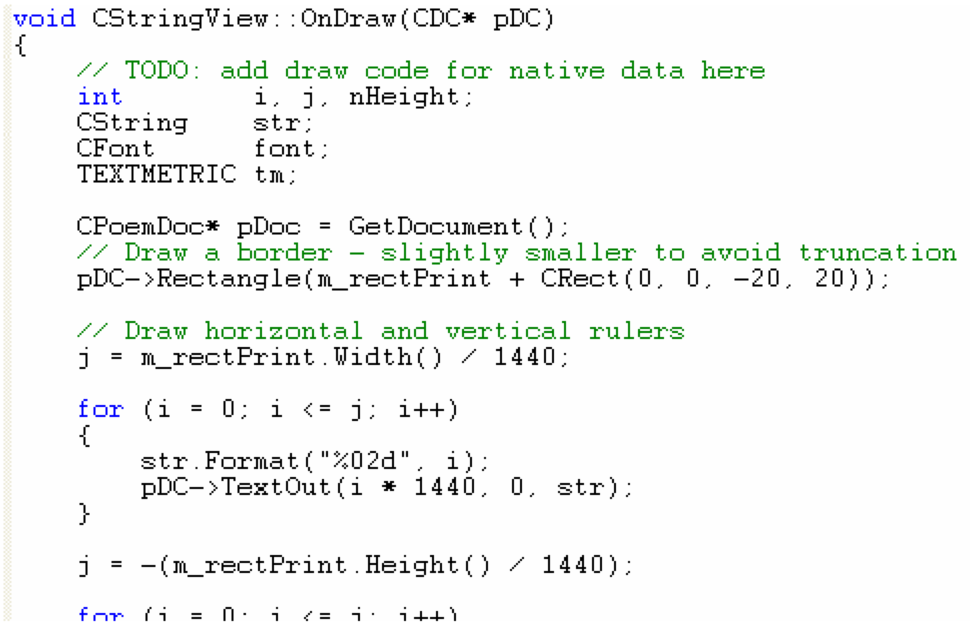
```
void CStringView::OnInitialUpdate()
{
    // TODO: Add your specialized code here and/or call the base class
    CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);
}
```

Listing 17.

Edit the `OnDraw()` function in **StringView.cpp**. The `OnDraw()` function of class `CStringView` draws on both the display and the printer. In addition to displaying the poem text lines in 10-point roman font, it draws a border around the printable area and a crude ruler along the top and left margins. `OnDraw()` assumes the `MM_TWIPS` mapping mode, in which 1 inch = 1440 units. Add the code shown below.

```
void CStringView::OnDraw(CDC* pDC)
{
    int        i, j, nHeight;
    CString    str;
    CFont      font;
    TEXTMETRIC tm;

    CPoemDoc* pDoc = GetDocument();
    // Draw a border – slightly smaller to avoid truncation
    pDC->Rectangle(m_rectPrint + CRect(0, 0, -20, 20));
    // Draw horizontal and vertical rulers
    j = m_rectPrint.Width() / 1440;
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(i * 1440, 0, str);
    }
    j = -(m_rectPrint.Height() / 1440);
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(0, -i * 1440, str);
    }
    // Print the poem 0.5 inch down and over;
    // use 10-point roman font
    font.CreateFont(-200, 0, 0, 0, 400, FALSE, FALSE, 0, ANSI_CHARSET,
                    OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
                    DEFAULT_QUALITY, DEFAULT_PITCH | FF_ROMAN, "Times New
Roman");
    CFont* pOldFont = (CFont*) pDC->SelectObject(&font);
    pDC->GetTextMetrics(&tm);
    nHeight = tm.tmHeight + tm.tmExternalLeading;
    TRACE("font height = %d, internal leading = %d\n", nHeight,
tm.tmInternalLeading);
```

```
        j = pDoc->m_stringArray.GetSize();
        for (i = 0; i < j; i++)
        {
            pDC->TextOut(720, -i * nHeight - 720, pDoc->m_stringArray[i]);
        }
        pDC->SelectObject(pOldFont);
        TRACE("LOGPIXELSX = %d, LOGPIXELSY = %d\n", pDC-
    >GetDeviceCaps(LOGPIXELSX),
              pDC->GetDeviceCaps(LOGPIXELSY));
        TRACE("HORZSIZE = %d, VERTSIZE = %d\n", pDC->GetDeviceCaps(HORZSIZE),
              pDC->GetDeviceCaps(VERTSIZE));
    }
```

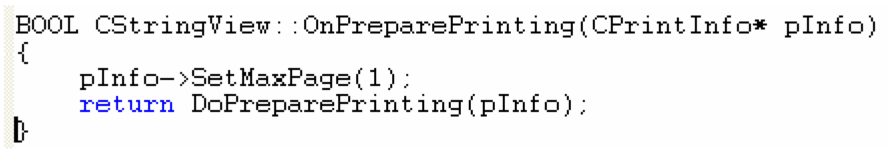


```
void CStringView::OnDraw(CDC* pDC)
{
    // TODO: add draw code for native data here
    int         i, j, nHeight;
    CString     str;
    CFont       font;
    TEXTMETRIC tm;

    CPoemDoc* pDoc = GetDocument();
    // Draw a border - slightly smaller to avoid truncation
    pDC->Rectangle(m_rectPrint + CRect(0, 0, -20, 20));

    // Draw horizontal and vertical rulers
    j = m_rectPrint.Width() / 1440;

    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(i * 1440, 0, str);
    }

    j = -(m_rectPrint.Height() / 1440);

    for (i = 0; i <= j; i++)
```

Listing 18.

Edit the `OnPreparePrinting()` function in **StringView.cpp**. This function sets the maximum number of pages in the print job. This example has only one page. It's absolutely necessary to call the base class `DoPreparePrinting()` function in your overridden `OnPreparePrinting()` function. Add the following code:

```
    BOOL CStringView::OnPreparePrinting(CPrintInfo* pInfo)
    {
        pInfo->SetMaxPage(1);
        return DoPreparePrinting(pInfo);
    }
```



```
BOOL CStringView::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```

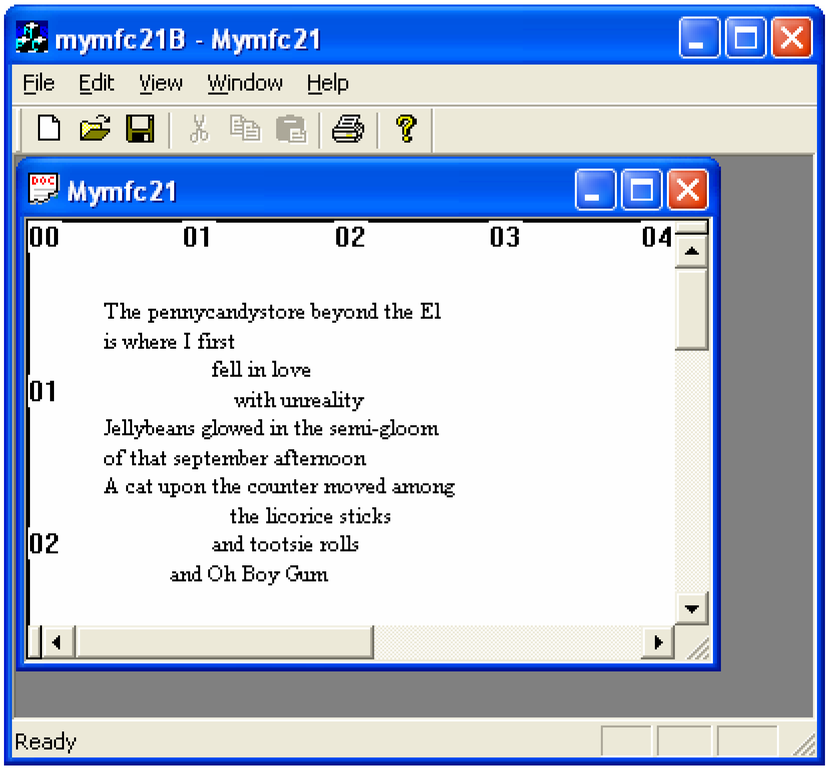
Listing 19.

Edit the constructor in **StringView.cpp**. The initial value of the print rectangle should be 8-by-15 inches, expressed in twips (1 inch = 1440 twips). Add the following code:

```
    CStringView::CStringView() : m_rectPrint(0, 0, 11520, -15120)
    {
        }

CStringView::CStringView() : m_rectPrint(0, 0, 11520, -15120)
{
    // TODO: add construction code here
}
```

Listing 20.

Build and test the application.



```
The pennycandystore beyond the El
is where I first
              fell in love
                 with unreality
Jellybeans glowed in the semi-gloom
of that september afternoon
A cat upon the counter moved among
                 the licorice sticks
              and tootsie rolls
         and Oh Boy Gum
```

Figure 22: MYMFC21B program output.

`CHexView` class, the cloned `CStringView` Class

Next, create `CHexView` by cloning the `CStringView`. Create the header and source files (**HexView.h** and **HexView.cpp**).

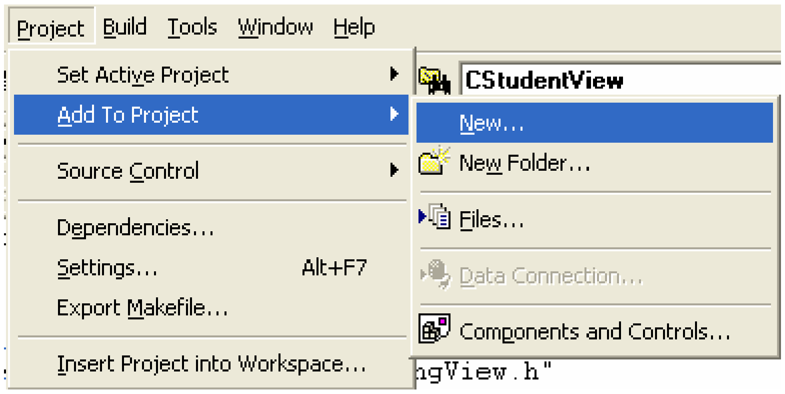
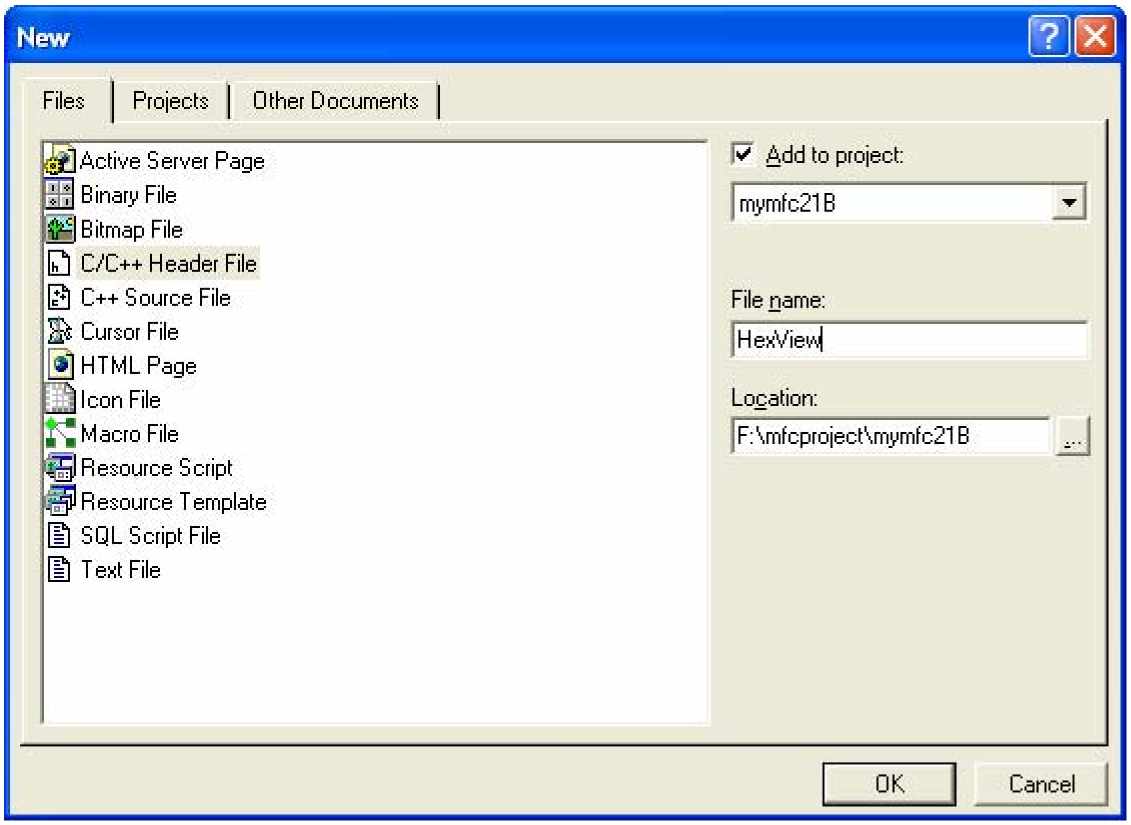


Figure 23: Creating and adding new `CHexView` class.



Add to project: mymfc21B

File name: HexView

Location: F:\mfcproject\mymfc21B

Figure 24: Creating the **HexView.h** header file.

Copy the cloned codes (**StringView.h** and **StringView.cpp**) and paste it to the **HexView.h** and **HexView.cpp** respectively. It is essentially the same code used for `CStringView` except for the `OnDraw()` member function

HEXVIEW.H

```
// HexView.h : interface of the CHexView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHEXVIEW_H__0A59E9CC_50F1_4B8E_8A3E_C3ED2955CE9D__INCLUDED_)
#define AFX_CHEXVIEW_H__0A59E9CC_50F1_4B8E_8A3E_C3ED2955CE9D__INCLUDED_

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


class CHexView : public CScrollView
{
protected: // create from serialization only
       CHexView();
       DECLARE_DYNCREATE(CHexView)

// Attributes
public:
       CPoemDoc* GetDocument();

// Operations
public:

// Overrides
```

```
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CHexView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual void OnInitialUpdate(); // called first time after construct
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL

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

protected:

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

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

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

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

#endif // !defined(AFX_CHEXVIEW_H__0A59E9CC_50F1_4B8E_8A3E_C3ED2955CE9D__INCLUDED_)
```

HEXVIEW.CPP

```
// HexView.cpp : implementation of the CHexView class
//

#include "stdafx.h"
#include "mymfc21B.h"

#include "PoemDoc.h"
#include "HexView.h"

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

/////////////////////////////////////////////////////////////////////////////
// CHexView

IMPLEMENT_DYNCREATE(CHexView, CScrollView)
```

```
BEGIN_MESSAGE_MAP(CHexView, CScrollView)
	//{{AFX_MSG_MAP(CHexView)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHexView construction/destruction

CHexView::CHexView() : m_rectPrint(0, 0, 11520, -15120)
{
	// TODO: add construction code here

}

CHexView::~CHexView()
{
}

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

	return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CHexView drawing

void CHexView::OnDraw(CDC* pDC)
{
    // hex dump of document strings
    int         i, j, k, l, n, nHeight;
    CString     outputLine, str;
    CFont       font;
    TEXTMETRIC tm;

    CPoemDoc* pDoc = GetDocument();
    font.CreateFont(-160, 80, 0, 0, 400, FALSE, FALSE, 0,
        ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
        DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS, "Arial");
    CFont* pOldFont = pDC->SelectObject(&font);
    pDC->GetTextMetrics(&tm);
    nHeight = tm.tmHeight + tm.tmExternalLeading;

    j = pDoc->m_stringArray.GetSize();
    for (i = 0; i < j; i++) {
        outputLine.Format("%02x   ", i);
        l = pDoc->m_stringArray[i].GetLength();
        for (k = 0; k < l; k++) {
            n = pDoc->m_stringArray[i][k] & 0x00ff;
            str.Format("%02x ", n);
            outputLine += str;
        }
        pDC->TextOut(720, -i * nHeight - 720, outputLine);
    }
    pDC->SelectObject(pOldFont);
}
```

```
void CHexView::OnInitialUpdate()
{
           CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);

}

/////////////////////////////////////////////////////////////////////////////
// CHexView printing

BOOL CHexView::OnPreparePrinting(CPrintInfo* pInfo)
{
       pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}

void CHexView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
       // TODO: add extra initialization before printing
}

void CHexView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
       // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CHexView diagnostics

#ifdef _DEBUG
void CHexView::AssertValid() const
{
       CScrollView::AssertValid();
}

void CHexView::Dump(CDumpContext& dc) const
{
       CScrollView::Dump(dc);
}

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

/////////////////////////////////////////////////////////////////////////////
// CHexView message handlers
```

Listing 21: `CHexView` class, the cloned `CStringView` class.

The `CHexView` class (**HexView.h** and **HexView.cpp**) was written to allow programmers to appreciate poetry. It is essentially the same code used for `CStringView` except for the `OnDraw()` member function:

```
void CHexView::OnDraw(CDC* pDC)
{
    // hex dump of document strings
    int         i, j, k, l, n, nHeight;
    CString     outputLine, str;
    CFont       font;
```

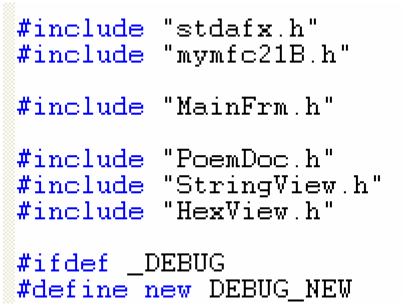
```
        TEXTMETRIC tm;

        CPoemDoc* pDoc = GetDocument();
        font.CreateFont(-160, 80, 0, 0, 400, FALSE, FALSE, 0,
            ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
            DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS, "Arial");
        CFont* pOldFont = pDC->SelectObject(&font);
        pDC->GetTextMetrics(&tm);
        nHeight = tm.tmHeight + tm.tmExternalLeading;

        j = pDoc->m_stringArray.GetSize();
        for (i = 0; i < j; i++) {
            outputLine.Format("%02x   ", i);
            l = pDoc->m_stringArray[i].GetLength();
            for (k = 0; k < l; k++) {
                n = pDoc->m_stringArray[i][k] & 0x00ff;
                str.Format("%02x ", n);
                outputLine += str;
            }
            pDC->TextOut(720, -i * nHeight - 720, outputLine);
        }
        pDC->SelectObject(pOldFont);
    }
```

This function displays a hexadecimal dump of all strings in the document's `m_stringArray` collection. Notice the use of the subscript operator to access individual characters in a `CString` object.

`CMainFrame` Class

As in MYMFC21, the MYMFC21B application's main frame window class needs a splitter window data member and a prototype for an overridden `OnCreateClient()` function. You can let AppWizard generate the code by specifying **Use Split Window**, as in MYMFC21. You won't have to modify the **MainFrm.h** file. The implementation file, **MainFrm.cpp**, needs both view class headers (and the prerequisite document header), as shown here:

```
    #include "PoemDoc.h"
    #include "StringView.h"
    #include "HexView.h"
```



```
#include "stdafx.h"
#include "mymfc21B.h"

#include "MainFrm.h"

#include "PoemDoc.h"
#include "StringView.h"
#include "HexView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
```

Listing 22.

AppWizard generates dynamic splitter code in the `OnCreateClient()` function, so you'll have to do some editing if you want a static splitter. Instead of calling `CSplitterWnd::Create`, you'll call the `CSplitterWnd::CreateStatic` function, which is tailored for multiple view classes. The following calls to `CSplitterWnd::CreateView` attach the two view classes. As the second and third `CreateStatic()` parameters (2, 1) dictate, this splitter window contains only two panes. The horizontal split is initially 100 device units from the top of the window. The top pane is the string view; the bottom pane is the hex dump view. The user can change the splitter bar position but the view configuration cannot be changed.

```
    BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT /*lpcs*/,
        CCreateContext* pContext)
```

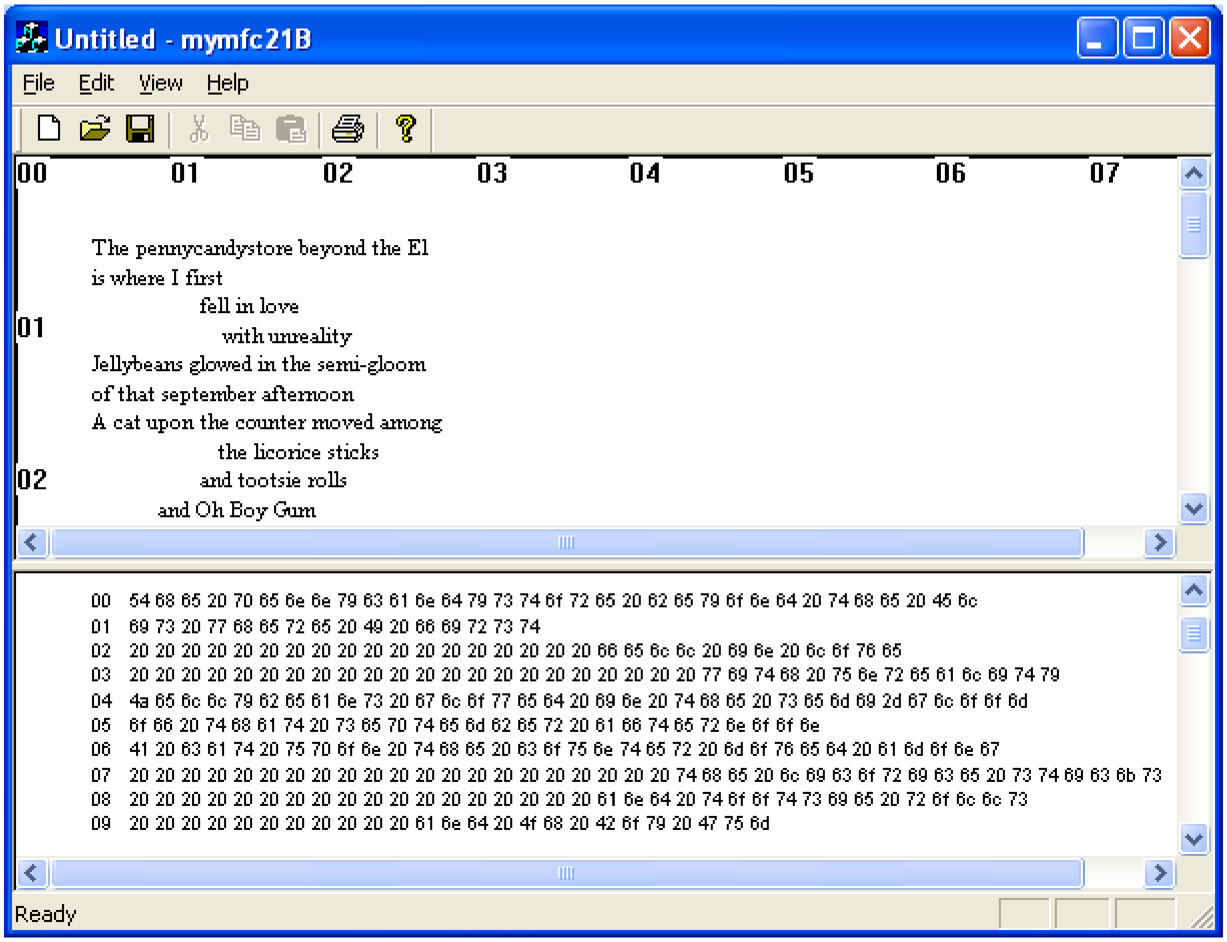
```
{
    VERIFY(m_wndSplitter.CreateStatic(this, 2, 1));
    VERIFY(m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CStringView), CSize(100,
100), pContext));
    VERIFY(m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CHexView), CSize(100, 100),
pContext));
    return TRUE;
}
```

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT /*lpcs*/,
    CCreateContext* pContext)
{
    VERIFY(m_wndSplitter.CreateStatic(this, 2, 1));
    VERIFY(m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CStringView),
                                    CSize(100, 100), pContext));
    VERIFY(m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CHexView),
                                    CSize(100, 100), pContext));
    return TRUE;
}
```

Listing 23.

Testing the MYMFC21B Application

When you start the MYMFC21B application, the window should look like the one shown below. Notice the separate horizontal scroll bars for the two views.



```
The pennycandystore beyond the El
is where I first
                fell in love
                   with unreality
Jellybeans glowed in the semi-gloom
of that september afternoon
A cat upon the counter moved among
                  the licorice sticks
                and tootsie rolls
          and Oh Boy Gum
```

```
00   54 68 65 20 70 65 6e 6e 79 63 61 6e 64 79 73 74 6f 72 65 20 62 65 79 6f 6e 64 20 74 68 65 20 45 6c
01   69 73 20 77 68 65 72 65 20 49 20 66 69 72 73 74
02   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 66 65 6c 6c 20 69 6e 20 6c 6f 76 65
03   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 77 69 74 68 20 75 6e 72 65 61 6c 69 74 79
04   4a 65 6c 6c 79 62 65 61 6e 73 20 67 6c 6f 77 65 64 20 69 6e 20 74 68 65 20 73 65 6d 69 2d 67 6c 6f 6f 6d
05   6f 66 20 74 68 61 74 20 73 65 70 74 65 6d 62 65 72 20 61 66 74 65 72 6e 6f 6f 6e
06   41 20 63 61 74 20 75 70 6f 6e 20 74 68 65 20 63 6f 75 6e 74 65 72 20 6d 6f 76 65 64 20 61 6d 6f 6e 67
07   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 74 68 65 20 6c 69 63 6f 72 69 63 65 20 73 74 69 63 6b 73
08   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 61 6e 64 20 74 6f 6f 74 73 69 65 20 72 6f 6c 6c 73
09   20 20 20 20 20 20 20 20 20 20 61 6e 64 20 4f 68 20 42 6f 79 20 47 75 6d
```

Figure 25: MYMFC21B program output with horizontal split views.

The MYMFC21C Example: Switching View Classes Without a Splitter

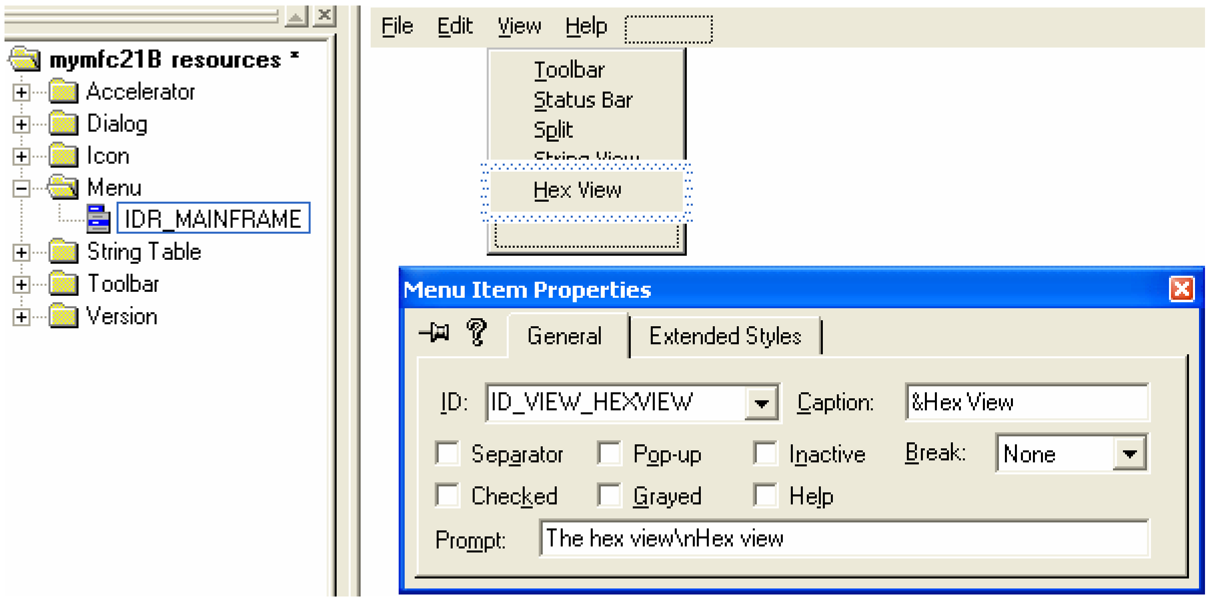
Sometimes you just want to switch view classes under program control and you don't want to be bothered with a splitter window. The MYMFC21C example is an SDI application that switches between `CStringView` and `CHexView` in response to selections on the **View** menu. Starting with what AppWizard generates, all you need to do is add two new menu commands and then add some code to the `CMainFrame` class. You also need to change the `CStringView` and `CHexView` constructors from `protected` to `public`.
We will use MYMFC21B project. Copy the MYMFC21B project folder to another folder as a backup and use the original one for this exercise. You can copy Visual C++ project (directory) to any other directory or computer, then you can edit and run the project as usual.

Resource Requirements

The following two items have been added to the **View** menu in the `IDR_MAINFRAME` menu resource.

Caption	Command ID	CMainFrame Function
St&ring View	`ID_VIEW_STRINGVIEW`	`OnViewStringView()`
&Hex View	`ID_VIEW_HEXVIEW`	`OnViewHexView()`

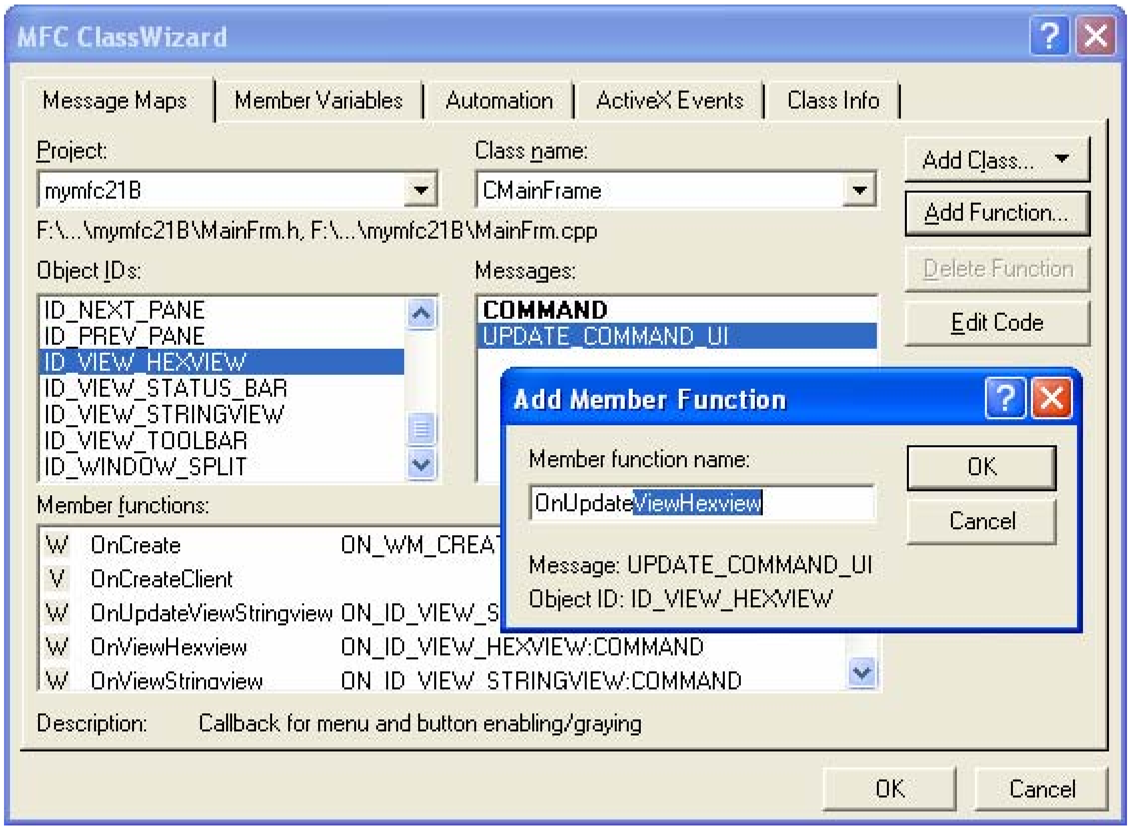
Table 1



ID: ID_VIEW_HEXVIEW
Caption: &Hex View
Break: None
Prompt: The hex view\nHex view

Figure 26: Adding and modifying menu items properties.

ClassWizard was used to add the command-handling functions (and corresponding update command UI handlers) to the `CMainFrame` class.



Project: mymfc21B
Class name: CMainFrame
F:\...\mymfc21B\MainFrm.h, F:\...\mymfc21B\MainFrm.cpp

Object IDs:
ID_NEXT_PANE
ID_PREV_PANE
ID_VIEW_HEXVIEW
ID_VIEW_STATUS_BAR
ID_VIEW_STRINGVIEW
ID_VIEW_TOOLBAR
ID_WINDOW_SPLIT

Messages:
COMMAND
UPDATE_COMMAND_UI

Member functions:

```
W  OnCreate                ON_WM_CREAT
V  OnCreateClient
W  OnUpdateViewStringview  ON_ID_VIEW_S
W  OnViewHexview           ON_ID_VIEW_HEXVIEW:COMMAND
W  OnViewStringview        ON_ID_VIEW_STRINGVIEW:COMMAND
```

Description: Callback for menu and button enabling/graying

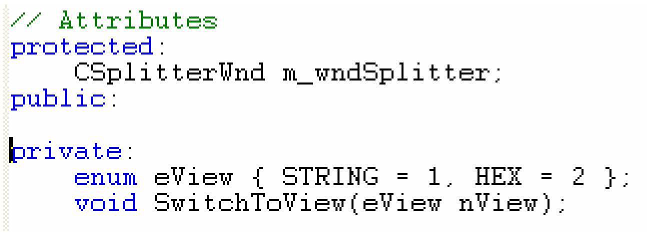
Member function name: OnUpdateViewHexview
Message: UPDATE_COMMAND_UI
Object ID: ID_VIEW_HEXVIEW

Figure 27: Adding the command-handling functions and corresponding update command UI handlers.

`CMainFrame` Class

The `CMainFrame` class gets a new private helper function, `SwitchToView()`, which is called from the two menu command handlers. The `enum` parameter tells the function which view to switch to. Here are the two added items in the **MainFrm.h** header file, added manually:

```
private:
    enum eView { STRING = 1, HEX = 2 };
    void SwitchToView(eView nView);
```

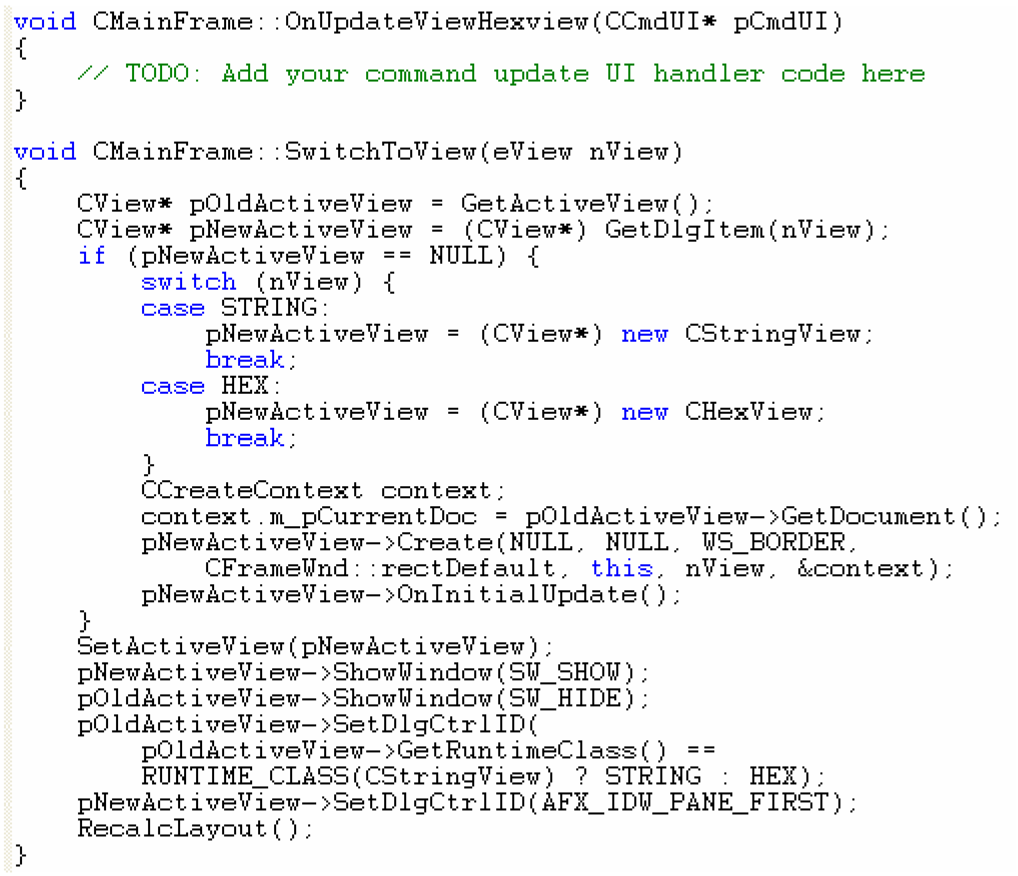


```
// Attributes
protected:
    CSplitterWnd m_wndSplitter;
public:

private:
    enum eView { STRING = 1, HEX = 2 };
    void SwitchToView(eView nView);
```

Listing 24.

The `SwitchToView()` function (in **MainFrm.cpp**) makes some low-level MFC calls to locate the requested view and to activate it. Don't worry about how it works. Just adapt it to your own applications when you want the view-switching feature. Add the following code manually:

```
void CMainFrame::SwitchToView(eView nView)
{
    CView* pOldActiveView = GetActiveView();
    CView* pNewActiveView = (CView*) GetDlgItem(nView);
    if (pNewActiveView == NULL) {
        switch (nView) {
        case STRING:
            pNewActiveView = (CView*) new CStringView;
            break;
        case HEX:
            pNewActiveView = (CView*) new CHexView;
            break;
        }
        CCreateContext context;
        context.m_pCurrentDoc = pOldActiveView->GetDocument();
        pNewActiveView->Create(NULL, NULL, WS_BORDER, CFrameWnd::rectDefault, this,
nView, &context);
        pNewActiveView->OnInitialUpdate();
    }
    SetActiveView(pNewActiveView);
    pNewActiveView->ShowWindow(SW_SHOW);
    pOldActiveView->ShowWindow(SW_HIDE);
    pOldActiveView->SetDlgCtrlID(
        pOldActiveView->GetRuntimeClass() == RUNTIME_CLASS(CStringView) ? STRING :
HEX);
    pNewActiveView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
    RecalcLayout();
}
```

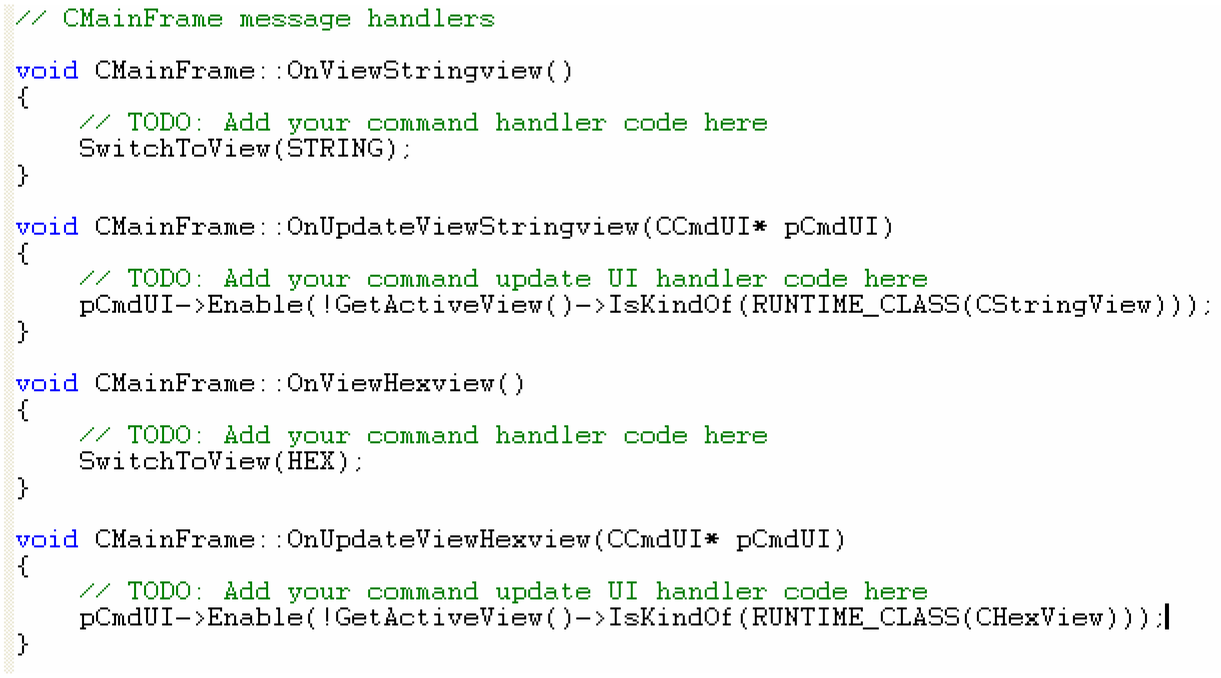


```
void CMainFrame::OnUpdateViewHexview(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
}

void CMainFrame::SwitchToView(eView nView)
{
    CView* pOldActiveView = GetActiveView();
    CView* pNewActiveView = (CView*) GetDlgItem(nView);
    if (pNewActiveView == NULL) {
        switch (nView) {
        case STRING:
            pNewActiveView = (CView*) new CStringView;
            break;
        case HEX:
            pNewActiveView = (CView*) new CHexView;
            break;
        }
        CCreateContext context;
        context.m_pCurrentDoc = pOldActiveView->GetDocument();
        pNewActiveView->Create(NULL, NULL, WS_BORDER,
            CFrameWnd::rectDefault, this, nView, &context);
        pNewActiveView->OnInitialUpdate();
    }
    SetActiveView(pNewActiveView);
    pNewActiveView->ShowWindow(SW_SHOW);
    pOldActiveView->ShowWindow(SW_HIDE);
    pOldActiveView->SetDlgCtrlID(
        pOldActiveView->GetRuntimeClass() ==
        RUNTIME_CLASS(CStringView) ? STRING : HEX);
    pNewActiveView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
    RecalcLayout();
}
```

Listing 25.

Finally, here are the menu command handlers and update command UI handlers that ClassWizard initially generated along with message map entries and prototypes. The update command UI handlers test the current view's class.

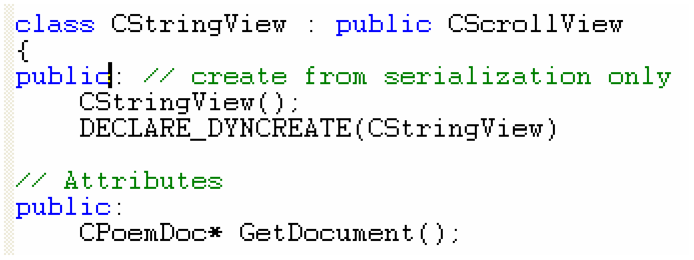
```
void CMainFrame::OnViewStringView()
{
    SwitchToView(STRING);
}

void CMainFrame::OnUpdateViewStringView(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(!GetActiveView()->IsKindOf(RUNTIME_CLASS(CStringView)));
}

void CMainFrame::OnViewHexView()
{
    SwitchToView(HEX);
}

void CMainFrame::OnUpdateViewHexView(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(!GetActiveView()->IsKindOf(RUNTIME_CLASS(CHexView)));
}
```



```
// CMainFrame message handlers

void CMainFrame::OnViewStringview()
{
    // TODO: Add your command handler code here
    SwitchToView(STRING);
}

void CMainFrame::OnUpdateViewStringview(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(!GetActiveView()->IsKindOf(RUNTIME_CLASS(CStringView)));
}

void CMainFrame::OnViewHexview()
{
    // TODO: Add your command handler code here
    SwitchToView(HEX);
}

void CMainFrame::OnUpdateViewHexview(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(!GetActiveView()->IsKindOf(RUNTIME_CLASS(CHexView)));
}
```

Listing 26.

Change the `CStringView` (**StringView.h**) and `CHexView` (**HexView.h**) constructors from `protected` to `public`.



```
class CStringView : public CScrollView
{
public: // create from serialization only
    CStringView();
    DECLARE_DYNCREATE(CStringView)

// Attributes
public:
    CPoemDoc* GetDocument();
```

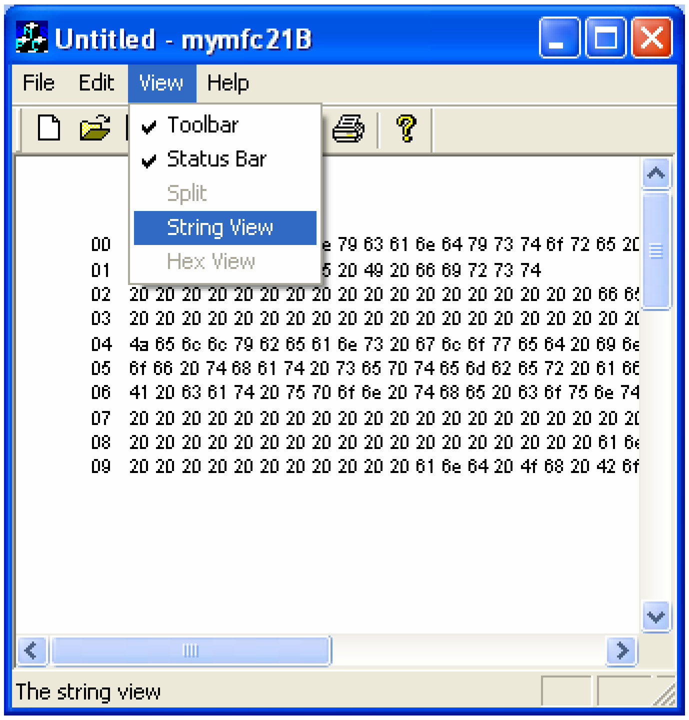
Listing 27.

```
class CHexView : public CScrollView
{
public: // create from serialization only
    CHexView();
    DECLARE_DYNCREATE(CHexView)

// Attributes
public:
    CPoemDoc* GetDocument();
```

Listing 28.

Testing the MYMFC21C Application

The MYMFC21C application initially displays the `CStringView` view of the document. You can toggle between the `CStringView` and `CHexView` views by choosing the appropriate command from the **View** menu. Both views of the document are shown side by side in Figure 28 and 29.



```
00                              e 79 63 61 6e 64 79 73 74 6f 72 65 20
01                              5 20 49 20 66 69 72 73 74
02  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 66 65
03  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
04  4a 65 6c 6c 79 62 65 61 6e 73 20 67 6c 6f 77 65 64 20 69 6e
05  6f 66 20 74 68 61 74 20 73 65 70 74 65 6d 62 65 72 20 61 66
06  41 20 63 61 74 20 75 70 6f 6e 20 74 68 65 20 63 6f 75 6e 74
07  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
08  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 61 6e
09  20 20 20 20 20 20 20 20 20 20 20 61 6e 64 20 4f 68 20 42 6f
```

The string view

Figure 28: MYMFC21C program output with two new view menus.

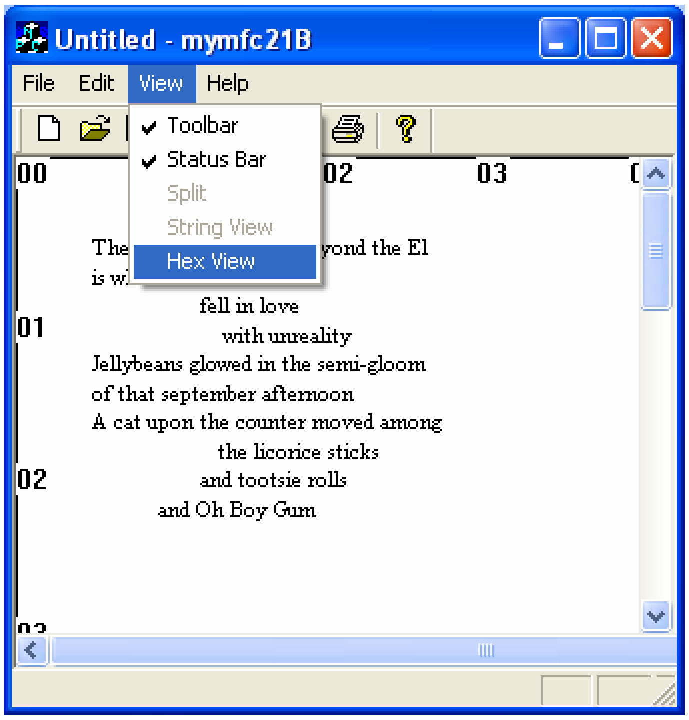


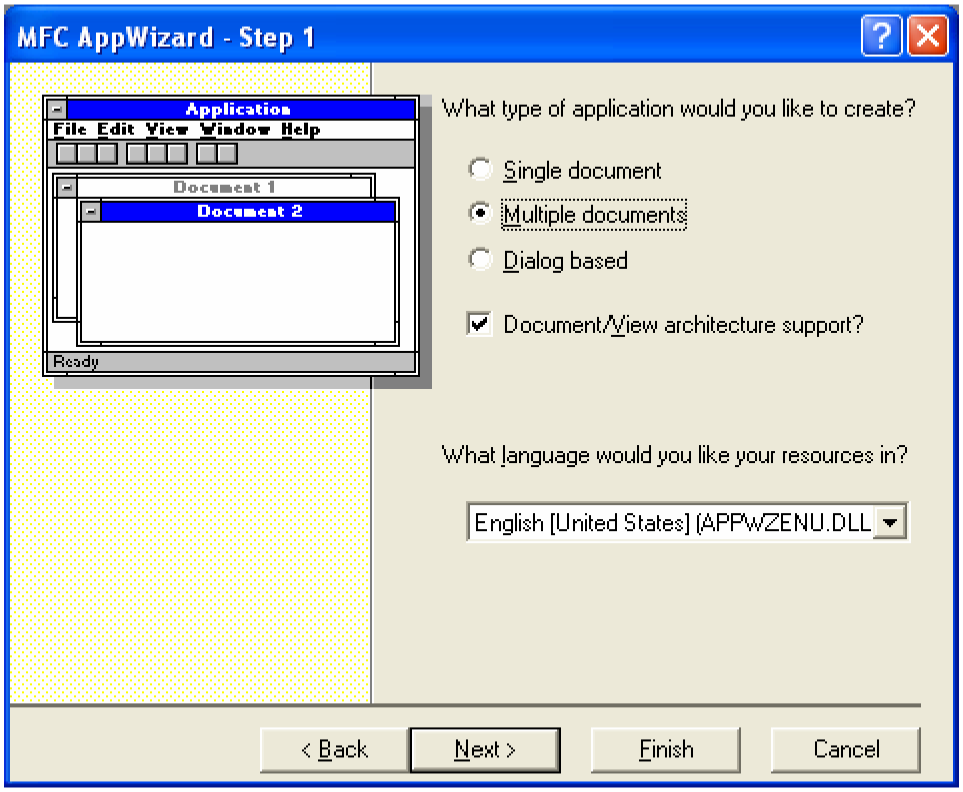
Figure 29: The `CStringView` view and the `CHexView` view of the document.

The MYMFC21D Example: A Multiple View Class MDI Application

The final example, MYMFC21D, uses the previous document and view classes to create a multiple view class MDI application without a splitter window. The logic is different from the logic in the other multiple view class applications. This time the action takes place in the application class in addition to the main frame class. As you study MYMFC21D, you'll gain more insight into the use of `CDocTemplate` objects.
This example was generated with the AppWizard **Context-Sensitive Help** option (step 4). In next Module, Module 15, you'll activate the context-sensitive help capability. If you're starting from scratch, use AppWizard to generate an ordinary MDI application with one of the view classes. Then add the second view class to the project and modify the application class files and main frame class files as described in the following sections.

The Steps

As usual, use AppWizard to create a MDI application. Follow the shown steps.



MFC AppWizard - Step 1

What type of application would you like to create?

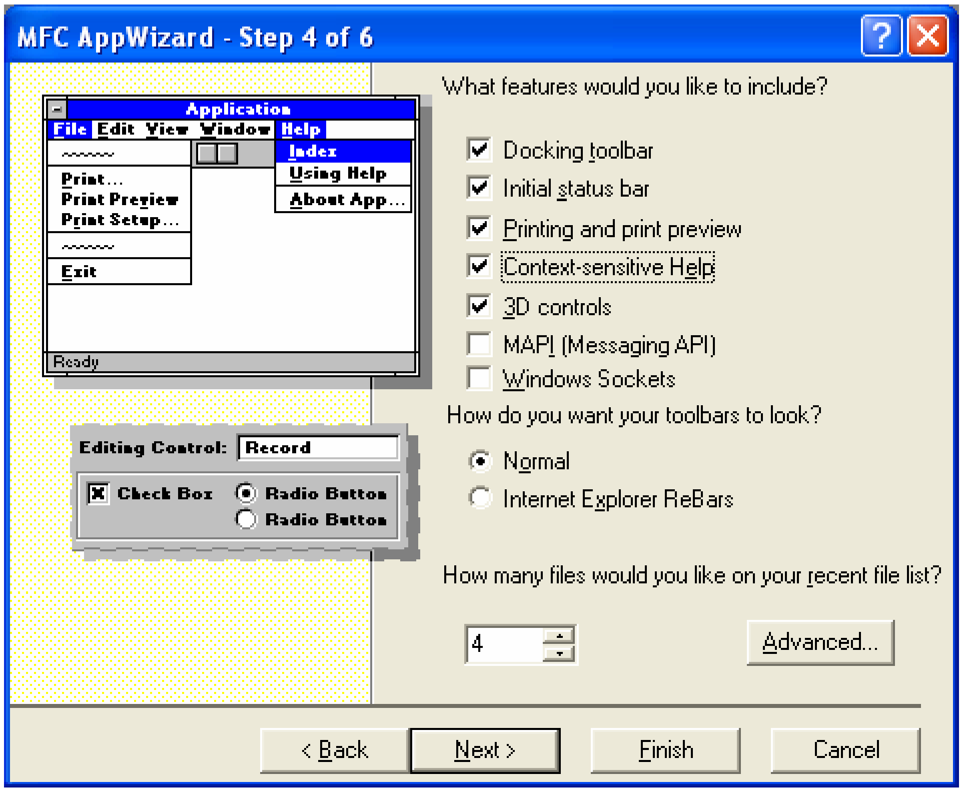
- Single document
- [✓ Multiple documents]
- Dialog based

[✓] Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 30: MYMFC21D AppWizard step 1 of 6.



MFC AppWizard - Step 4 of 6

What features would you like to include?

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

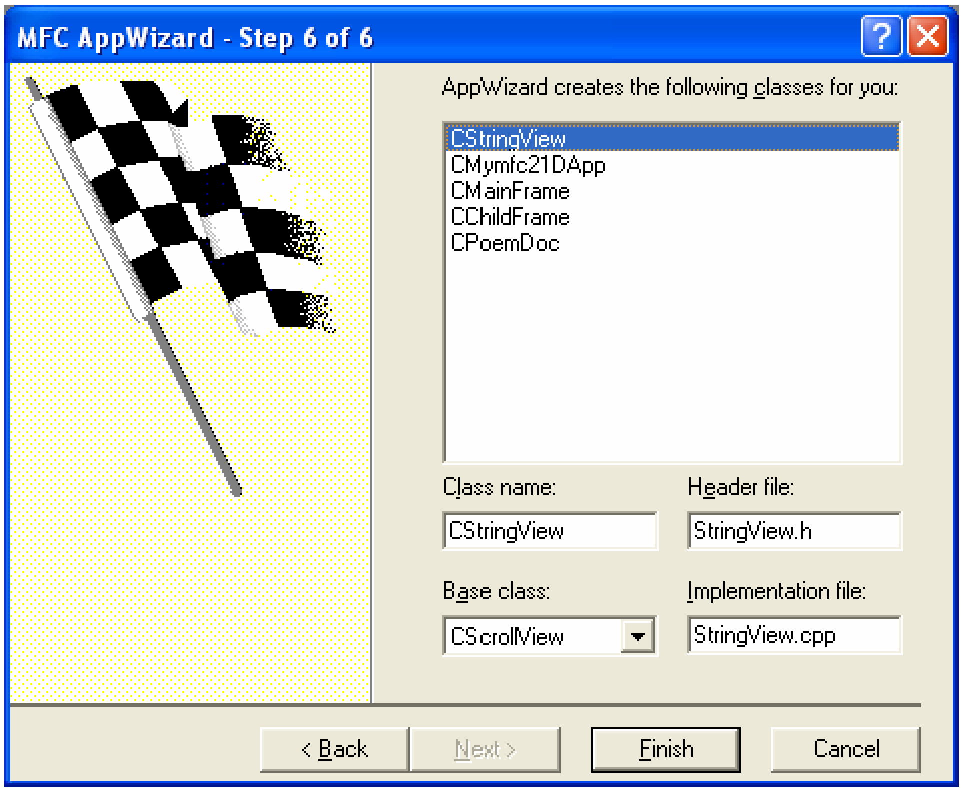
How do you want your toolbars to look?

- [✓ Normal]
- Internet Explorer ReBars

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

4

Figure 31: MYMFC21D step 4 of 6 AppWizard. Don’t forget to tick the Context-sensitive Help; this option will be used in the next module’s project.



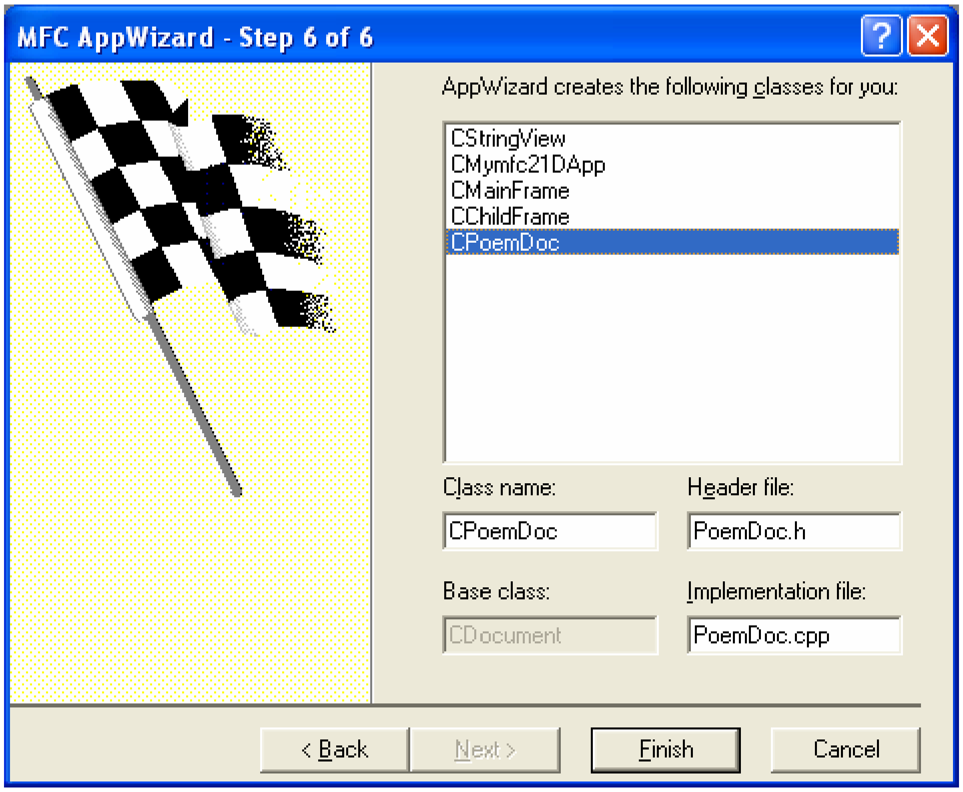
MFC AppWizard - Step 6 of 6

AppWizard creates the following classes for you:

CStringView
CMymfc21DApp
CMainFrame
CChildFrame
CPoemDoc

Class name: CStringView
Header file: StringView.h
Base class: CScrollView
Implementation file: StringView.cpp

Figure 32: MYMFC21D step 6 of 6 AppWizard, renaming the view class, their related files and using `CScrollView` as the view base class.



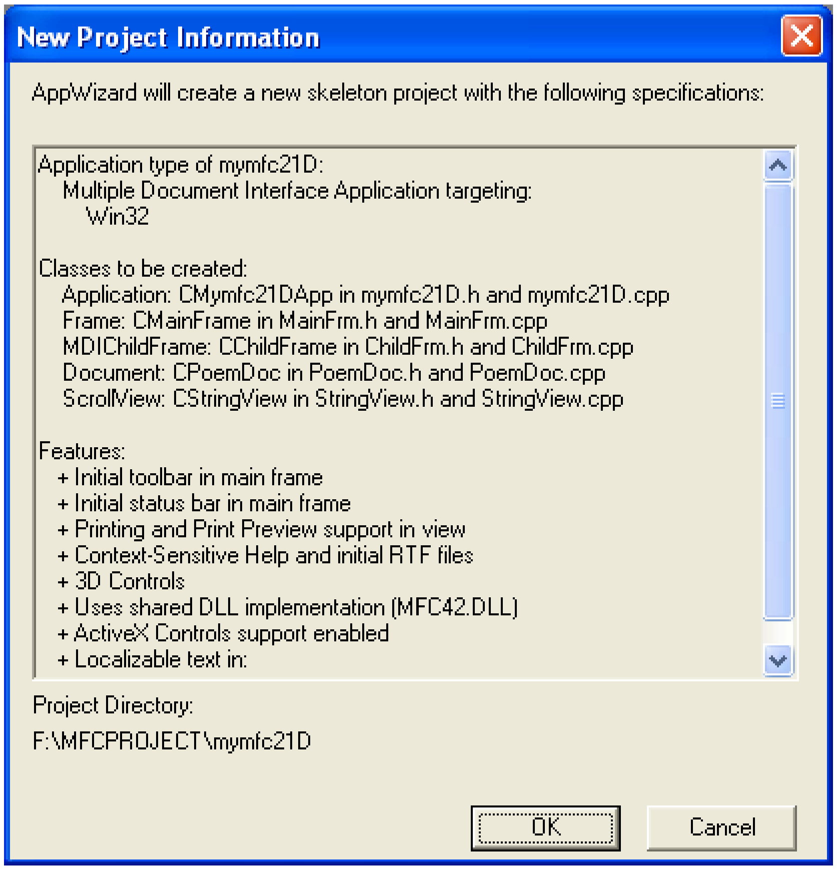
MFC AppWizard - Step 6 of 6

AppWizard creates the following classes for you:

CStringView
CMymfc21DApp
CMainFrame
CChildFrame
CPoemDoc

Class name: CPoemDoc
Header file: PoemDoc.h
Base class: CDocument
Implementation file: PoemDoc.cpp

Figure 33: MYMFC21D step 6 of 6 AppWizard, renaming the document class and their related files.


New Project Information

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

```
Application type of mymfc21D:
   Multiple Document Interface Application targeting:
      Win32

Classes to be created:
   Application: CMymfc21DApp in mymfc21D.h and mymfc21D.cpp
   Frame: CMainFrame in MainFrm.h and MainFrm.cpp
   MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
   Document: CPoemDoc in PoemDoc.h and PoemDoc.cpp
   ScrollView: CStringView in StringView.h and StringView.cpp

Features:
  + Initial toolbar in main frame
  + Initial status bar in main frame
  + Printing and Print Preview support in view
  + Context-Sensitive Help and initial RTF files
  + 3D Controls
  + Uses shared DLL implementation (MFC42.DLL)
  + ActiveX Controls support enabled
  + Localizable text in:
```

Project Directory:

F:\MFCPROJECT\mymfc21D

Figure 34: MYMFC21D project summary.

Note that this is an MDI application. Add a `CStringArray` data member to the `CPoemDoc` class. Edit the **PoemDoc.h** header file or use ClassView.

```
public:
    CStringArray m_stringArray;
```

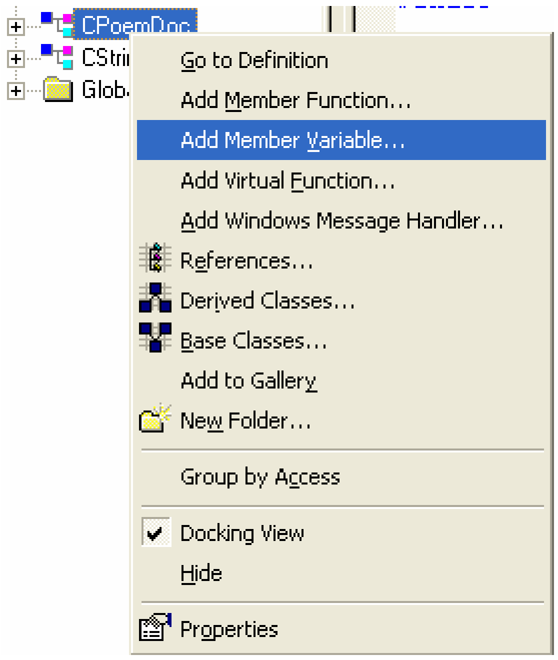
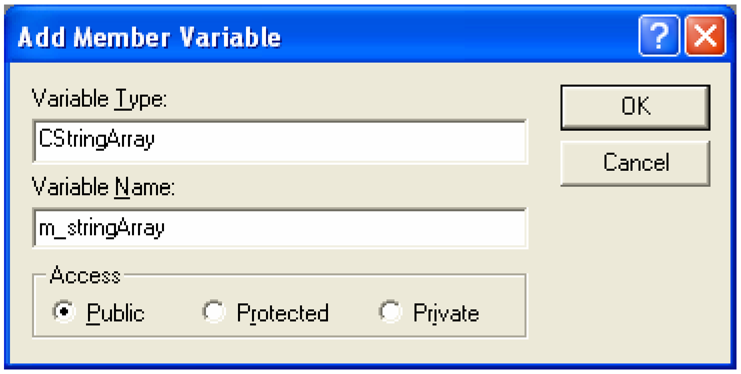


Figure 35: Adding a `CStringArray` data member to the `CPoemDoc` class using ClassView.


Add Member Variable

Variable Type: CStringArray

Variable Name: m_stringArray

Access: [✓ Public] Protected Private

Figure 36: Entering the member variable type and name.

```
// Implementation
public:
    CStringArray m_stringArray;
    virtual ~CPoemDoc();
#ifdef _DEBUG
```

Listing 29.

The document data is stored in a string array. The MFC library `CStringArray` class holds an array of `CString` objects, accessible by a zero-based subscript. You need not set a maximum dimension in the declaration because the array is dynamic.
Add a `CRect` data member to the `CStringView` class. Edit the **StringView.h** header file or use ClassView:

```
    private:
        CRect m_rectPrint;
```

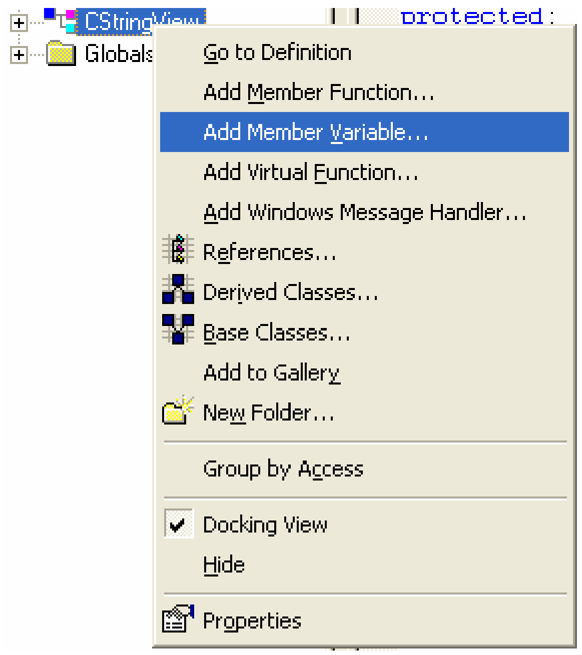
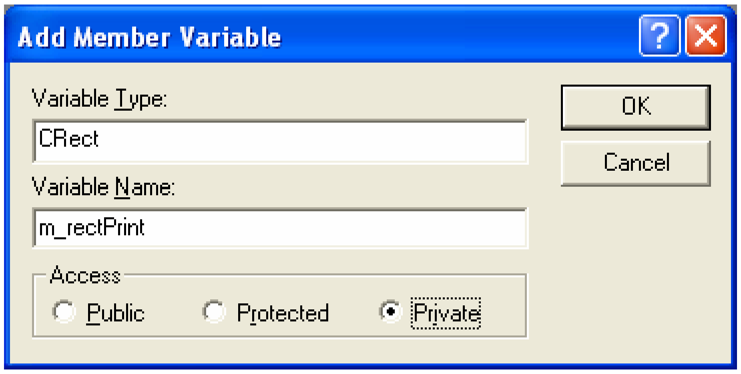


Figure 37: Adding a `CRect` data member to the `CStringView` class.

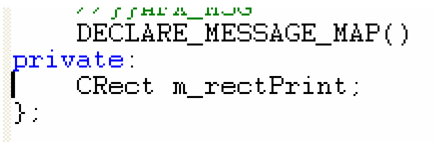

Add Member Variable

Variable Type: CRect

Variable Name: m_rectPrint

Access: Public [✓ Private] Protected

Figure 38: Entering the member variable type and name.



```
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CRect m_rectPrint;
};
```

Listing 30.

Edit three `CPoemDoc` member functions in the file **PoemDoc.cpp**. AppWizard generated skeleton `OnNewDocument()` and `Serialize()` functions, but we'll have to use ClassWizard to override the `DeleteContents()` function. We'll initialize the poem document in the overridden `OnNewDocument()` function. `DeleteContents()` is called in `CDocument::OnNewDocument`, so by calling the base class function first we're sure the poem won't be deleted. The text, by the way, is an excerpt from the twentieth poem in Lawrence Ferlinghetti's book A Coney Island of the Mind. Type 10 lines of your choice. You can substitute another poem or maybe your favorite Win32 function description. Add the following code:

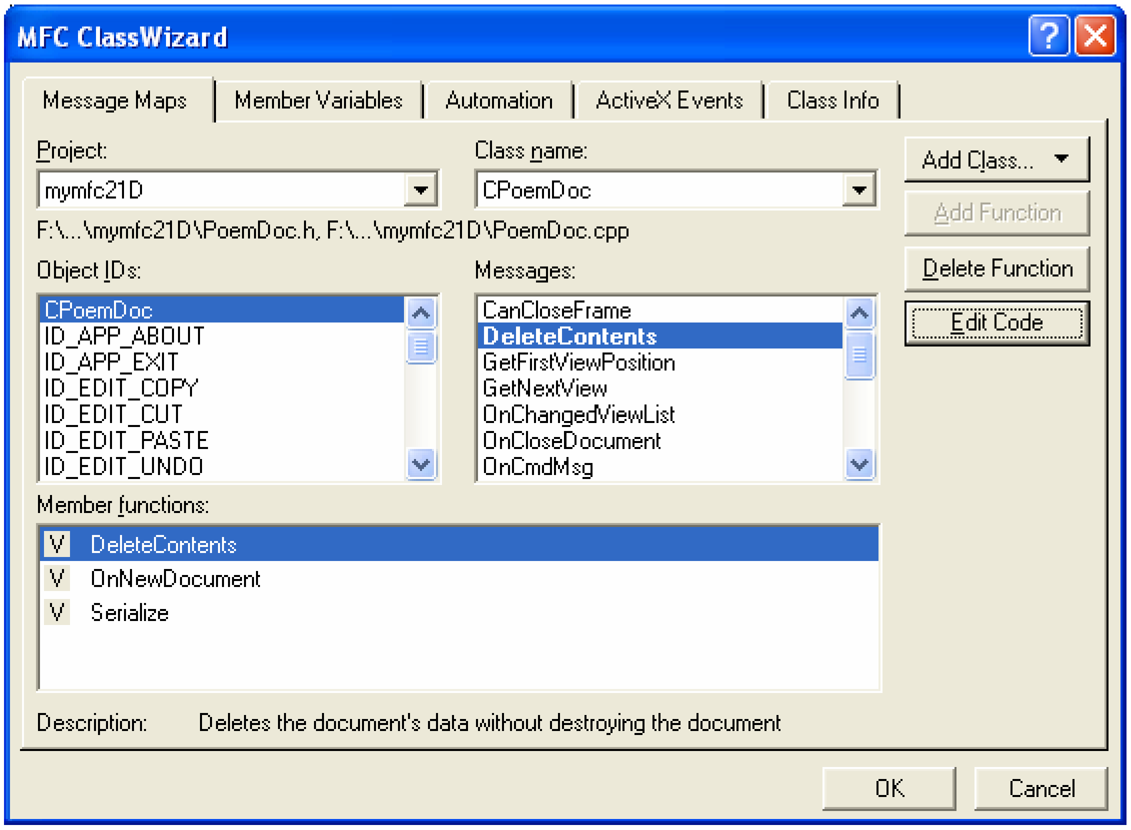
```
BOOL CPoemDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    m_stringArray.SetSize(10);
    m_stringArray[0] = "The pennycandystore beyond the El";
    m_stringArray[1] = "is where I first";
    m_stringArray[2] = "                  fell in love";
    m_stringArray[3] = "                      with unreality";
    m_stringArray[4] = "Jellybeans glowed in the semi-gloom";
    m_stringArray[5] = "of that september afternoon";
    m_stringArray[6] = "A cat upon the counter moved among";
    m_stringArray[7] = "                     the licorice sticks";
    m_stringArray[8] = "                  and tootsie rolls";
    m_stringArray[9] = "           and Oh Boy Gum";

    return TRUE;
}
```

```
BOOL CPoemDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    m_stringArray.SetSize(10);
    m_stringArray[0] = "The pennycandystore beyond the El";
    m_stringArray[1] = "is where I first";
    m_stringArray[2] = "                  fell in love";
    m_stringArray[3] = "                      with unreality";
    m_stringArray[4] = "Jellybeans glowed in the semi-gloom";
    m_stringArray[5] = "of that september afternoon";
    m_stringArray[6] = "A cat upon the counter moved among";
    m_stringArray[7] = "                     the licorice sticks";
    m_stringArray[8] = "                  and tootsie rolls";
    m_stringArray[9] = "           and Oh Boy Gum";

    return TRUE;
}
```

Listing 31.

The `CStringArray` class supports dynamic arrays, but here we're using the `m_stringArray` object as though it were a static array of 10 elements. The application framework calls the document's virtual `DeleteContents()` function when it closes the document; this action deletes the strings in the array. A `CStringArray` contains actual objects, and a `CObArray` contains pointers to objects. This distinction is important when it's time to delete the array elements. Here the `RemoveAll()` function actually deletes the string objects:

```
void CPoemDoc::DeleteContents()
{
    // called before OnNewDocument() and when document is closed
    m_stringArray.RemoveAll();
}
```



Project: mymfc21D

Class name: CPoemDoc

F:\...\mymfc21D\PoemDoc.h, F:\...\mymfc21D\PoemDoc.cpp

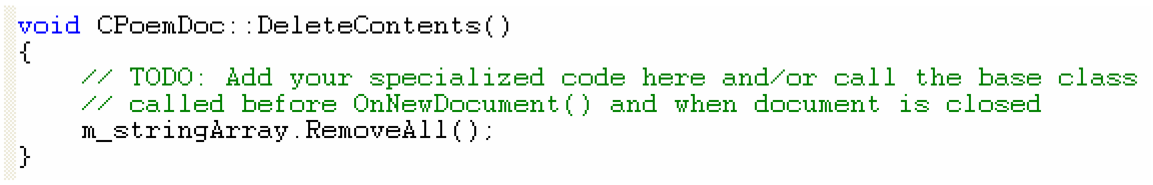
Object IDs:
CPoemDoc
ID_APP_ABOUT
ID_APP_EXIT
ID_EDIT_COPY
ID_EDIT_CUT
ID_EDIT_PASTE
ID_EDIT_UNDO

Messages:
CanCloseFrame
DeleteContents
GetFirstViewPosition
GetNextView
OnChangedViewList
OnCloseDocument
OnCmdMsg

Member functions:
V DeleteContents
V OnNewDocument
V Serialize

Description: Deletes the document's data without destroying the document

Figure 39: Similar to the previous example, adding the `RemoveAll()` function.

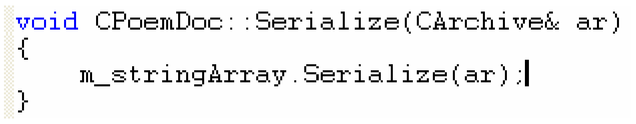


```
void CPoemDoc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    // called before OnNewDocument() and when document is closed
    m_stringArray.RemoveAll();
}
```

Listing 32.

Serialization isn't important in this example, but the following function illustrates how easy it is to serialize strings. The application framework calls the `DeleteContents()` function before loading from the archive, so you don't have to worry about emptying the array. Add the following code:

```
void CPoemDoc::Serialize(CArchive& ar)
{
    m_stringArray.Serialize(ar);
}
```

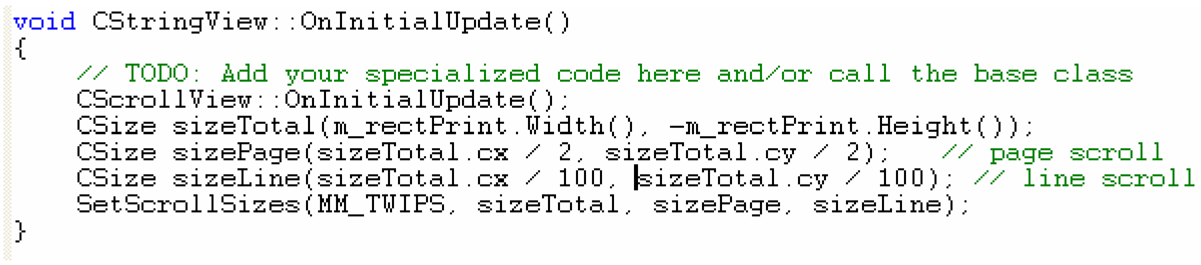


```
void CPoemDoc::Serialize(CArchive& ar)
{
    m_stringArray.Serialize(ar);
}
```

Listing 33.

Edit the `OnInitialUpdate()` function in **StringView.cpp**. You must override the function for all classes derived from `CScrollView`. This function's job is to set the logical window size and the mapping mode. Add the following boldface code:

```
void CStringView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);
}
```

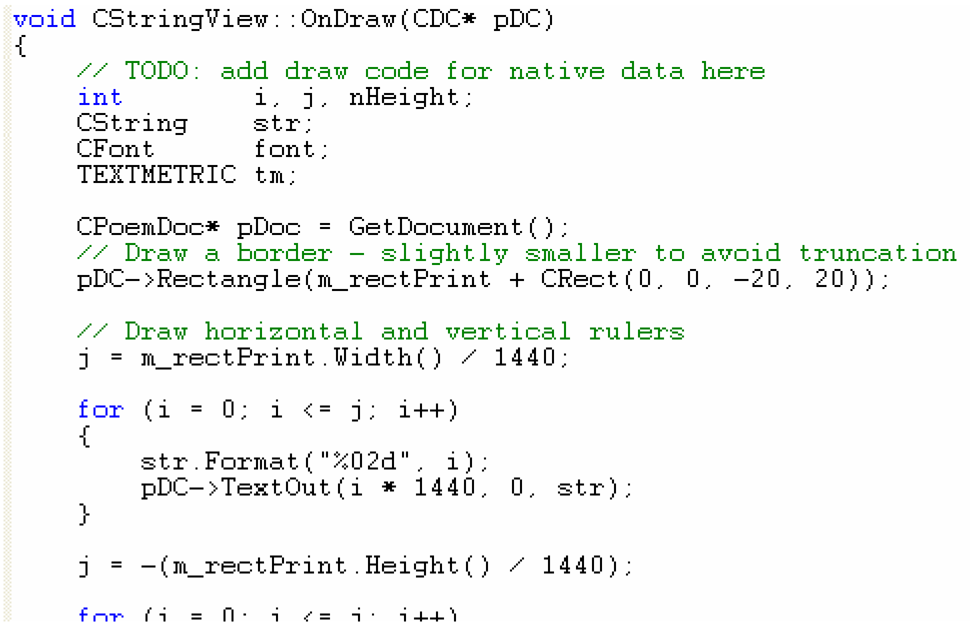


```
void CStringView::OnInitialUpdate()
{
    // TODO: Add your specialized code here and/or call the base class
    CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);
}
```

Listing 34.

Edit the `OnDraw()` function in **StringView.cpp**. The `OnDraw()` function of class `CStringView` draws on both the display and the printer. In addition to displaying the poem text lines in 10-point roman font, it draws a border around the printable area and a crude ruler along the top and left margins. `OnDraw()` assumes the `MM_TWIPS` mapping mode, in which 1 inch = 1440 units. Add the code shown below.

```
void CStringView::OnDraw(CDC* pDC)
{
    int        i, j, nHeight;
    CString    str;
    CFont      font;
    TEXTMETRIC tm;

    CPoemDoc* pDoc = GetDocument();
    // Draw a border – slightly smaller to avoid truncation
    pDC->Rectangle(m_rectPrint + CRect(0, 0, -20, 20));
    // Draw horizontal and vertical rulers
    j = m_rectPrint.Width() / 1440;
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(i * 1440, 0, str);
    }
    j = -(m_rectPrint.Height() / 1440);
    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(0, -i * 1440, str);
    }
    // Print the poem 0.5 inch down and over
    // use 10-point roman font
    font.CreateFont(-200, 0, 0, 0, 400, FALSE, FALSE, 0, ANSI_CHARSET,
                    OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
                    DEFAULT_QUALITY, DEFAULT_PITCH | FF_ROMAN, "Times New
Roman");
    CFont* pOldFont = (CFont*) pDC->SelectObject(&font);
    pDC->GetTextMetrics(&tm);
    nHeight = tm.tmHeight + tm.tmExternalLeading;
    TRACE("font height = %d, internal leading = %d\n", nHeight,
tm.tmInternalLeading);
    j = pDoc->m_stringArray.GetSize();
    for (i = 0; i < j; i++)
    {
        pDC->TextOut(720, -i * nHeight - 720, pDoc->m_stringArray[i]);
    }
    pDC->SelectObject(pOldFont);
    TRACE("LOGPIXELSX = %d, LOGPIXELSY = %d\n", pDC-
>GetDeviceCaps(LOGPIXELSX),
          pDC->GetDeviceCaps(LOGPIXELSY));
    TRACE("HORZSIZE = %d, VERTSIZE = %d\n", pDC->GetDeviceCaps(HORZSIZE),
          pDC->GetDeviceCaps(VERTSIZE));
}
```

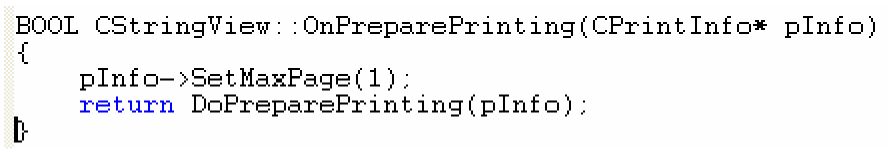


```
void CStringView::OnDraw(CDC* pDC)
{
    // TODO: add draw code for native data here
    int         i, j, nHeight;
    CString     str;
    CFont       font;
    TEXTMETRIC tm;

    CPoemDoc* pDoc = GetDocument();
    // Draw a border - slightly smaller to avoid truncation
    pDC->Rectangle(m_rectPrint + CRect(0, 0, -20, 20));

    // Draw horizontal and vertical rulers
    j = m_rectPrint.Width() / 1440;

    for (i = 0; i <= j; i++)
    {
        str.Format("%02d", i);
        pDC->TextOut(i * 1440, 0, str);
    }

    j = -(m_rectPrint.Height() / 1440);

    for (i = 0; i <= j; i++)
```

Listing 35.

Edit the `OnPreparePrinting()` function in **StringView.cpp**. This function sets the maximum number of pages in the print job. This example has only one page. It's absolutely necessary to call the base class `DoPreparePrinting()` function in your overridden `OnPreparePrinting()` function. Add the following code:

```
BOOL CStringView::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```

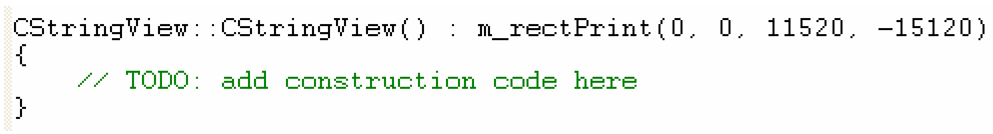


```
BOOL CStringView::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```

Listing 36.

Edit the constructor in **StringView.cpp**. The initial value of the print rectangle should be 8-by-15 inches, expressed in twips (1 inch = 1440 twips). Add the following code:

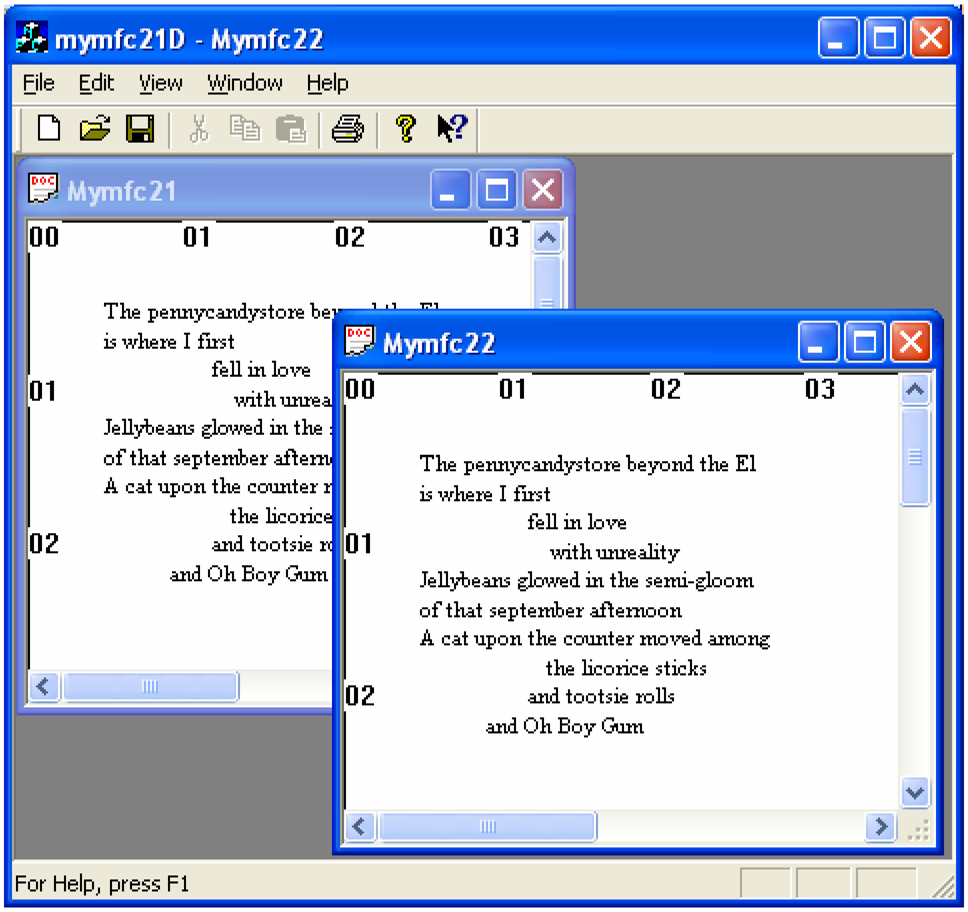
```
CStringView::CStringView() : m_rectPrint(0, 0, 11520, -15120)
{
}
```



```
CStringView::CStringView() : m_rectPrint(0, 0, 11520, -15120)
{
    // TODO: add construction code here
}
```

Listing 37.

Build and test the application.



```
The pennycandystore beyond the El
is where I first
                fell in love
                    with unreality
Jellybeans glowed in the semi-gloom
of that september afternoon
A cat upon the counter moved among
                    the licorice sticks
                and tootsie rolls
        and Oh Boy Gum
```

Figure 40: MYMFC21D program output, the MDI.

Next, create `CHexView` by cloning the `CStringView` as shown in the previous example.

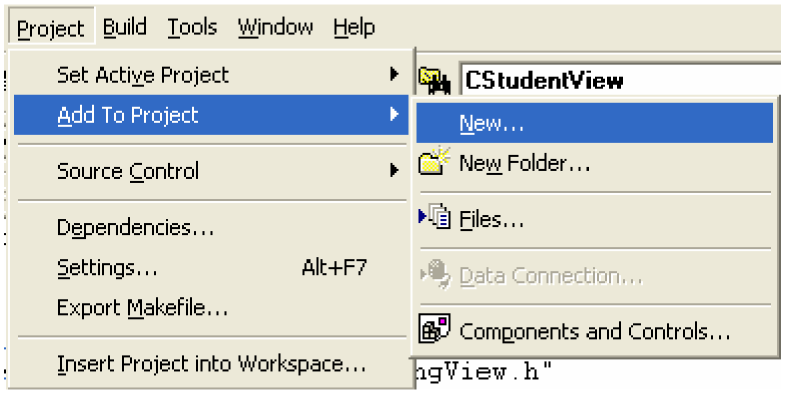
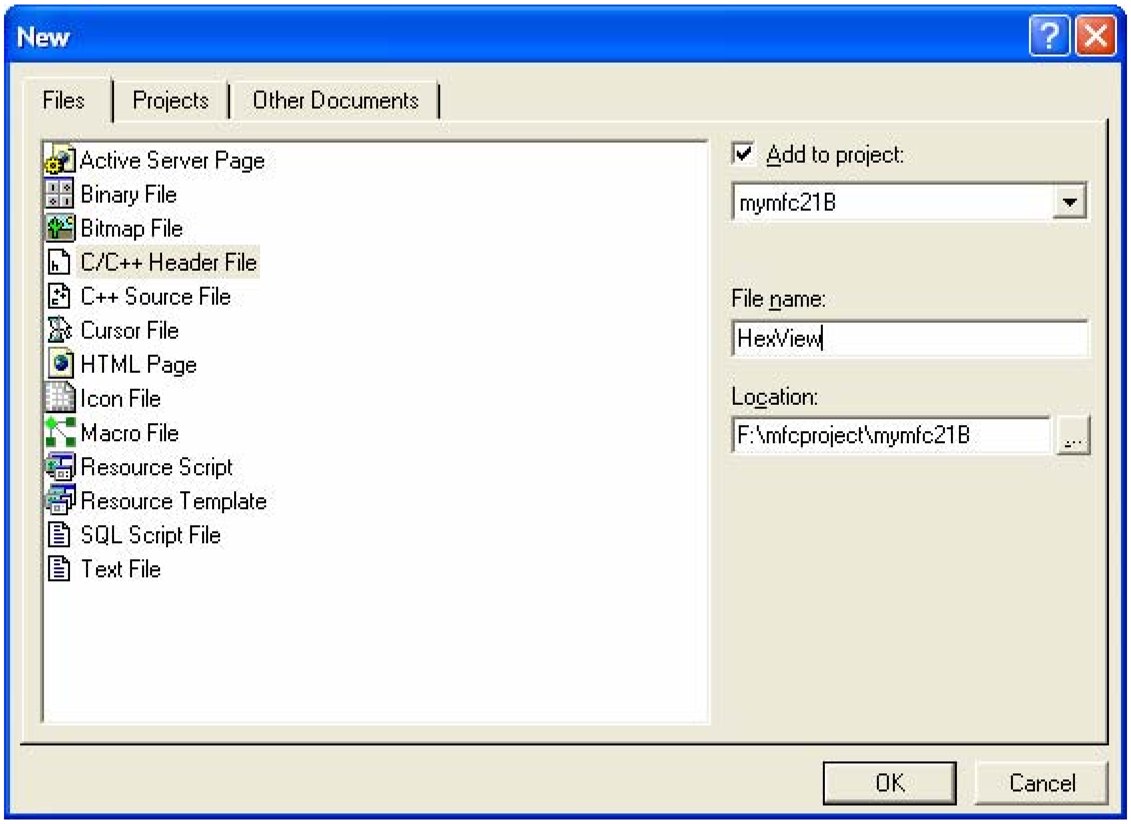


Figure 41: Creating and adding new files (class) to project.



Add to project: [✓] mymfc21B

File name: HexView

Location: F:\mfcproject\mymfc21B

Figure 42: Adding empty **HexView.h** header file to the project.

Copy the following cloned codes (**StringView.h** and **StringView.cpp**) into the **HexView.h** and **HexView.cpp** respectively.

HEXVIEW.H

```
// HexView.h : interface of the CHexView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHEXVIEW_H__0A59E9CC_50F1_4B8E_8A3E_C3ED2955CE9D__INCLUDED_)
#define AFX_CHEXVIEW_H__0A59E9CC_50F1_4B8E_8A3E_C3ED2955CE9D__INCLUDED_

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


class CHexView : public CScrollView
{
protected: // create from serialization only
       CHexView();
       DECLARE_DYNCREATE(CHexView)

// Attributes
public:
       CPoemDoc* GetDocument();

// Operations
public:

// Overrides
       // ClassWizard generated virtual function overrides
       //{{AFX_VIRTUAL(CHexView)
       public:
       virtual void OnDraw(CDC* pDC);  // overridden to draw this view
       virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
       protected:
       virtual void OnInitialUpdate(); // called first time after construct
       virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
       virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
       virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
       //}}AFX_VIRTUAL

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

protected:

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

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

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

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

#endif // !defined(AFX_CHEXVIEW_H__0A59E9CC_50F1_4B8E_8A3E_C3ED2955CE9D__INCLUDED_)
```

HEXVIEW.CPP

```
// HexView.cpp : implementation of the CHexView class
//

#include "stdafx.h"
#include "mymfc21D.h"

#include "PoemDoc.h"
#include "HexView.h"

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

/////////////////////////////////////////////////////////////////////////////
// CHexView

IMPLEMENT_DYNCREATE(CHexView, CScrollView)

BEGIN_MESSAGE_MAP(CHexView, CScrollView)
```

```
	//{{AFX_MSG_MAP(CHexView)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHexView construction/destruction

CHexView::CHexView() : m_rectPrint(0, 0, 11520, -15120)
{
	// TODO: add construction code here

}

CHexView::~CHexView()
{
}

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

	return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CHexView drawing

void CHexView::OnDraw(CDC* pDC)
{
// hex dump of document strings
    int        i, j, k, l, n, nHeight;
    CString    outputLine, str;
    CFont      font;
    TEXTMETRIC tm;

    CPoemDoc* pDoc = GetDocument();
    font.CreateFont(-160, 80, 0, 0, 400, FALSE, FALSE, 0,
        ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
        DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS, "Arial");
    CFont* pOldFont = pDC->SelectObject(&font);
    pDC->GetTextMetrics(&tm);
    nHeight = tm.tmHeight + tm.tmExternalLeading;

    j = pDoc->m_stringArray.GetSize();
    for (i = 0; i < j; i++) {
        outputLine.Format("%02x   ", i);
        l = pDoc->m_stringArray[i].GetLength();
        for (k = 0; k < l; k++) {
            n = pDoc->m_stringArray[i][k] & 0x00ff;
            str.Format("%02x ", n);
            outputLine += str;
        }
        pDC->TextOut(720, -i * nHeight - 720, outputLine);
    }
    pDC->SelectObject(pOldFont);
}

void CHexView::OnInitialUpdate()
{
```

```
           CScrollView::OnInitialUpdate();
    CSize sizeTotal(m_rectPrint.Width(), -m_rectPrint.Height());
    CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);   // page scroll
    CSize sizeLine(sizeTotal.cx / 100, sizeTotal.cy / 100); // line scroll
    SetScrollSizes(MM_TWIPS, sizeTotal, sizePage, sizeLine);

}

/////////////////////////////////////////////////////////////////////////////
// CHexView printing

BOOL CHexView::OnPreparePrinting(CPrintInfo* pInfo)
{
       pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}

void CHexView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
       // TODO: add extra initialization before printing
}

void CHexView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
       // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CHexView diagnostics

#ifdef _DEBUG
void CHexView::AssertValid() const
{
       CScrollView::AssertValid();
}

void CHexView::Dump(CDumpContext& dc) const
{
       CScrollView::Dump(dc);
}

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

/////////////////////////////////////////////////////////////////////////////
// CHexView message handlers
```

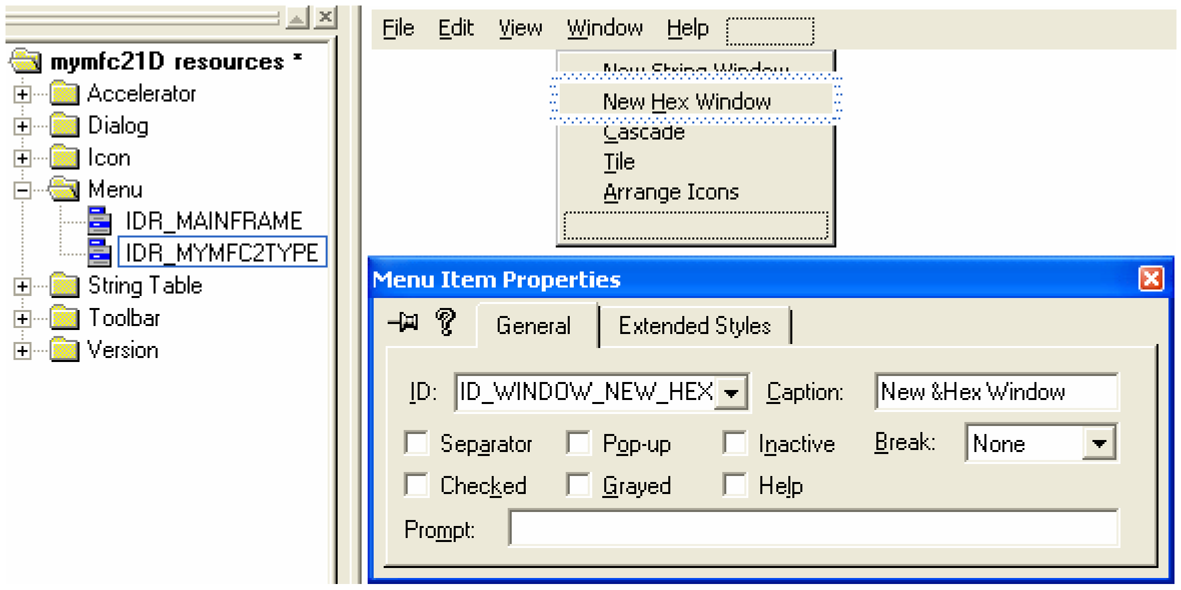
Listing 38: The `CHexView` class, a cloned `CStringView` class.

Resource Requirements

The two items below have been added to the **Window** menu in the `IDR_MYMFC2TYPE` menu resource.

Caption	Command ID	CMainFrame Function
New &String Window (replaces **New Window** item)	`ID_WINDOW_NEW_STRING`	`CMDIFrameWnd::OnWindowNew` (default when the program start)
New &Hex Window	`ID_WINDOW_NEW_HEX`	`OnWindowNewHex`

Table 2.



mymfc21D resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
IDR_MYMFC2TYPE
String Table
Toolbar
Version

File Edit View Window Help

New String Window
New Hex Window
Cascade
Tile
Arrange Icons

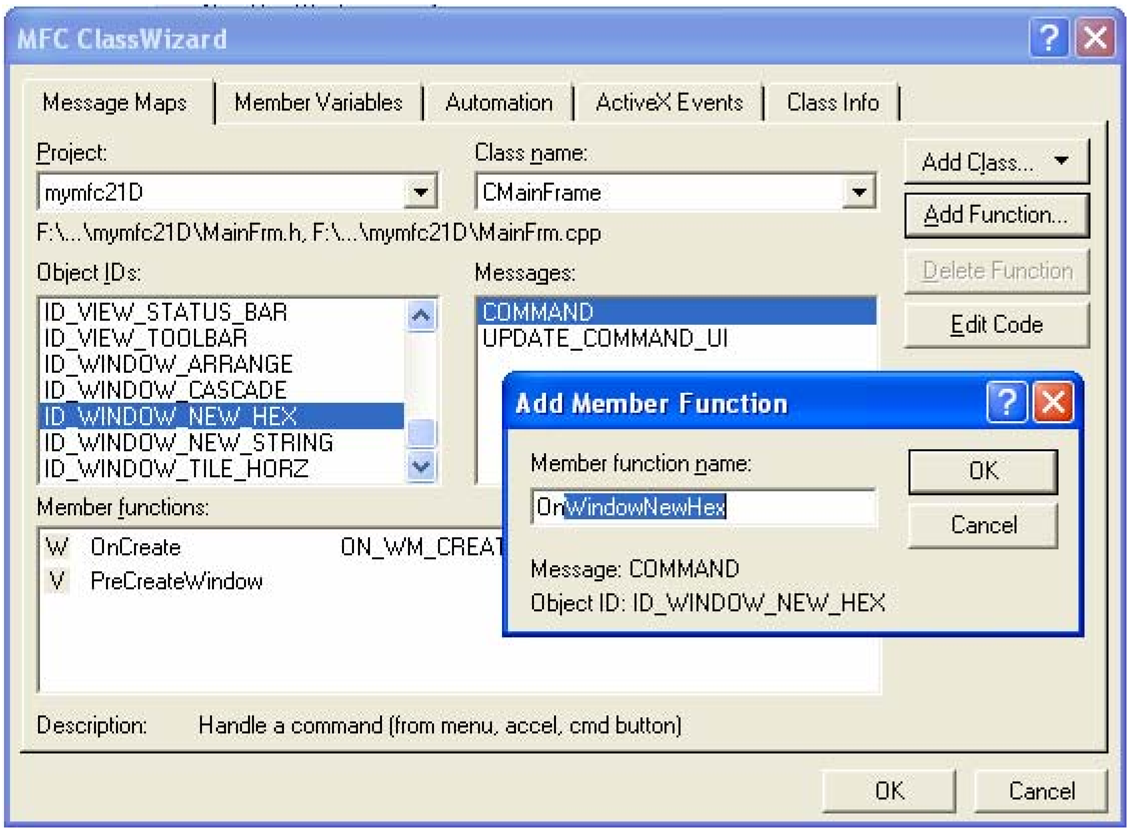
Menu Item Properties

General | Extended Styles

ID: ID_WINDOW_NEW_HEX
Caption: New &Hex Window
Separator [ ] Pop-up [ ] Inactive [ ] Break: None
Checked [ ] Grayed [ ] Help [ ]
Prompt:

Figure 43: Adding and modifying menu items properties.

ClassWizard was used to add the command-handling function `OnWindowNewHex()` to the `CMainFrame` class.



MFC ClassWizard

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

Project: mymfc21D
Class name: CMainFrame
F:\...\mymfc21D\MainFrm.h, F:\...\mymfc21D\MainFrm.cpp

Object IDs:
ID_VIEW_STATUS_BAR
ID_VIEW_TOOLBAR
ID_WINDOW_ARRANGE
ID_WINDOW_CASCADE
ID_WINDOW_NEW_HEX
ID_WINDOW_NEW_STRING
ID_WINDOW_TILE_HORZ

Messages:
COMMAND
UPDATE_COMMAND_UI

Member functions:
W OnCreate ON_WM_CREAT
V PreCreateWindow

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

Add Class... | Add Function... | Delete Function | Edit Code | OK | Cancel

Add Member Function

Member function name: OnWindowNewHex
Message: COMMAND
Object ID: ID_WINDOW_NEW_HEX

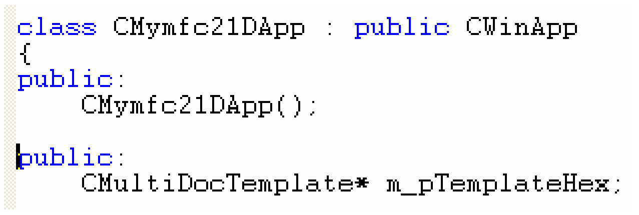
OK | Cancel

Figure 44: Adding the command-handling function `OnWindowNewHex()` to the `CMainFrame` class.

`CMymfc21DApp` Class

In the application class header file, **mymfc21D.h**, the following data member and function prototype have been added:

```
public:
    CMultiDocTemplate* m_pTemplateHex;
```

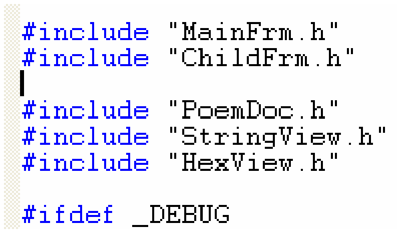


```
class CMymfc21DApp : public CWinApp
{
public:
    CMymfc21DApp();

public:
    CMultiDocTemplate* m_pTemplateHex;
```

Listing 39.

The implementation file, **mymfc21D.cpp**, contains the `#include` statements shown here:

```
#include "HexView.h"
```

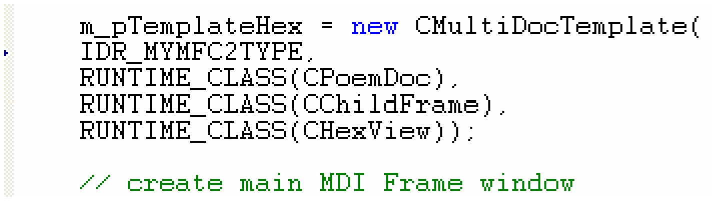


```
#include "MainFrm.h"
#include "ChildFrm.h"

#include "PoemDoc.h"
#include "StringView.h"
#include "HexView.h"

#ifdef _DEBUG
```

Listing 40.

The `CMymfc21DApp InitInstance()` member function has the code shown below inserted immediately after the `AddDocTemplate()` function call.

```
m_pTemplateHex = new CMultiDocTemplate(
    IDR_MYMFC2TYPE,
    RUNTIME_CLASS(CPoemDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CHexView));
```



```
    m_pTemplateHex = new CMultiDocTemplate(
    IDR_MYMFC2TYPE,
    RUNTIME_CLASS(CPoemDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CHexView));

    // create main MDI Frame window
```

Listing 41.

The `AddDocTemplate()` call generated by AppWizard established the primary document/frame/view combination for the application that is effective when the program starts. The template object above is a secondary template that can be activated in response to the **New Hex Window** menu item.
Now all you need is an `ExitInstance()` member function that cleans up the secondary template:

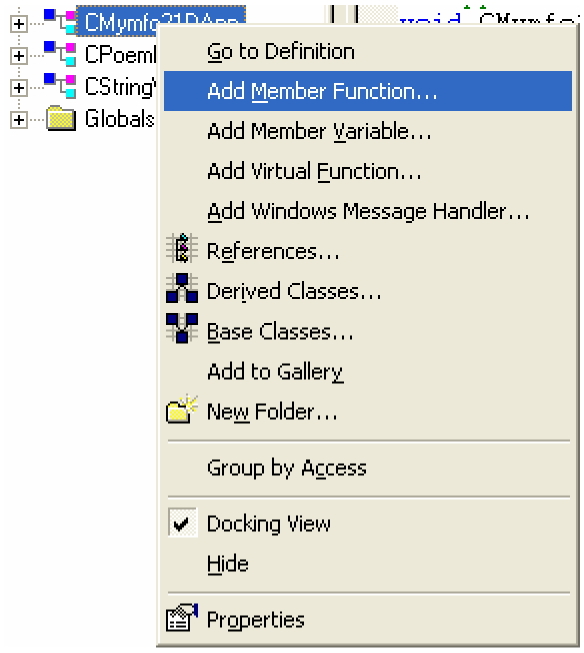
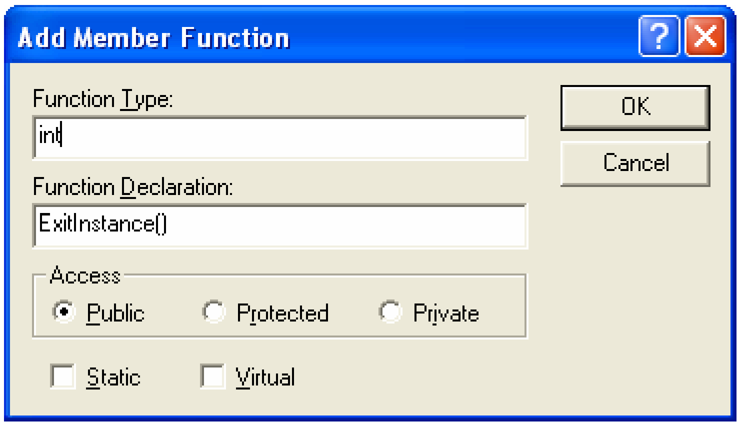


Figure 45: Adding an `ExitInstance()` member function that cleans up the secondary template.



Function Type: int

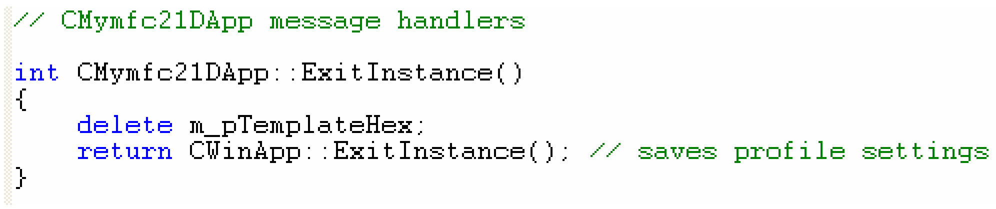
Function Declaration: ExitInstance()

Access: [✓ Public] Protected Private

Static Virtual

Figure 46: Entering the member function's type and declaration.

```
int CMymfc21DApp::ExitInstance()
{
    delete m_pTemplateHex;
    return CWinApp::ExitInstance(); // saves profile settings
}
```



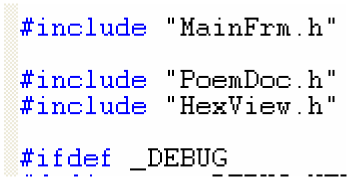
```
// CMymfc21DApp message handlers

int CMymfc21DApp::ExitInstance()
{
    delete m_pTemplateHex;
    return CWinApp::ExitInstance(); // saves profile settings
}
```

Listing 42.

`CMainFrame`

The main frame class implementation file, **MainFrm.cpp**, has the `CHexView` class header (and the prerequisite document header) included:

```
        #include "PoemDoc.h"
        #include "HexView.h"
```

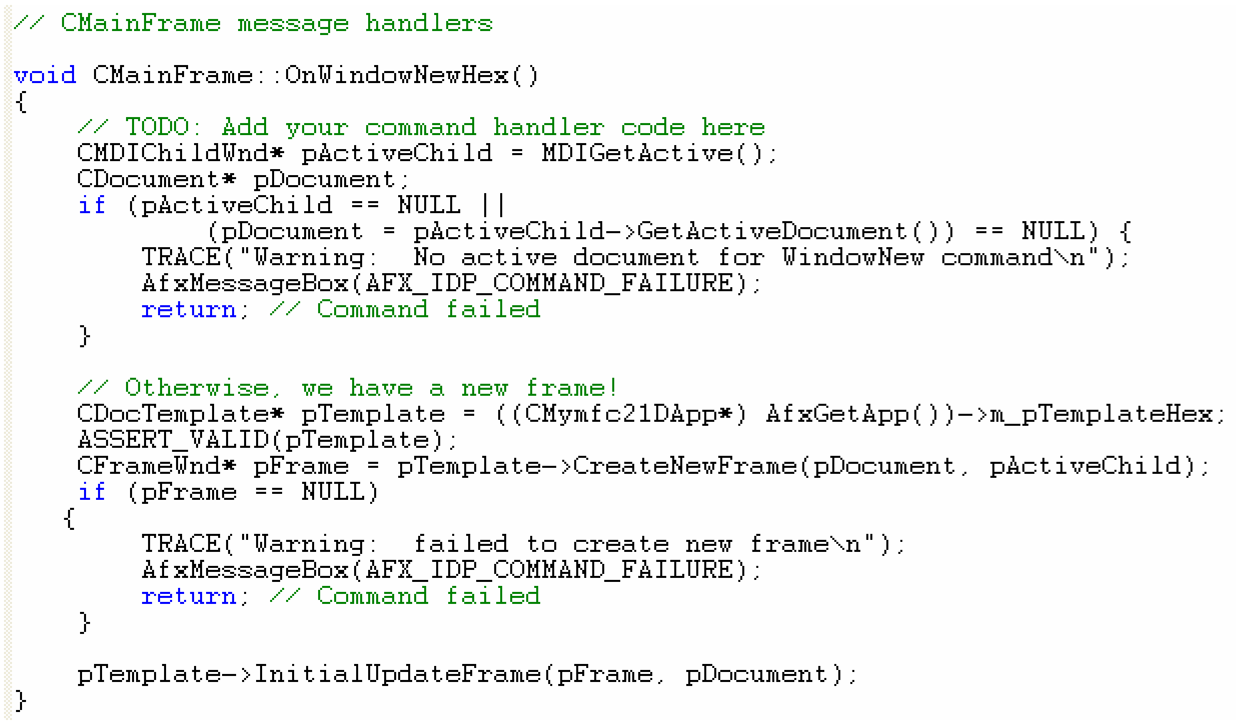


```
#include "MainFrm.h"

#include "PoemDoc.h"
#include "HexView.h"

#ifdef _DEBUG
```

Listing 43.

The base frame window class, `CMDIFrameWnd`, has an `OnWindowNew()` function that is normally connected to the standard **New Window** menu item on the **Window** menu. The **New String Window** menu item is mapped to this function in MYMFC21D. The **New Hex Window** menu item is mapped to the command handler function below to create new hex child windows. The function is a clone of `OnWindowNew()`, adapted for the hex view-specific template defined in `InitInstance()`.

```
        void CMainFrame::OnWindowNewHex()
        {
            CMDIChildWnd* pActiveChild = MDIGetActive();
            CDocument* pDocument;
            if (pActiveChild == NULL ||
                    (pDocument = pActiveChild->GetActiveDocument()) == NULL) {
                TRACE("Warning:  No active document for WindowNew command\n");
                AfxMessageBox(AFX_IDP_COMMAND_FAILURE);
                return; // Command failed
            }

            // Otherwise, we have a new frame!
            CDocTemplate* pTemplate = ((CMymfc21DApp*) AfxGetApp())->m_pTemplateHex;
            ASSERT_VALID(pTemplate);
            CFrameWnd* pFrame = pTemplate->CreateNewFrame(pDocument, pActiveChild);
            if (pFrame == NULL)
           {
                TRACE("Warning:  failed to create new frame\n");
                AfxMessageBox(AFX_IDP_COMMAND_FAILURE);
                return; // Command failed
            }

            pTemplate->InitialUpdateFrame(pFrame, pDocument);
        }
```



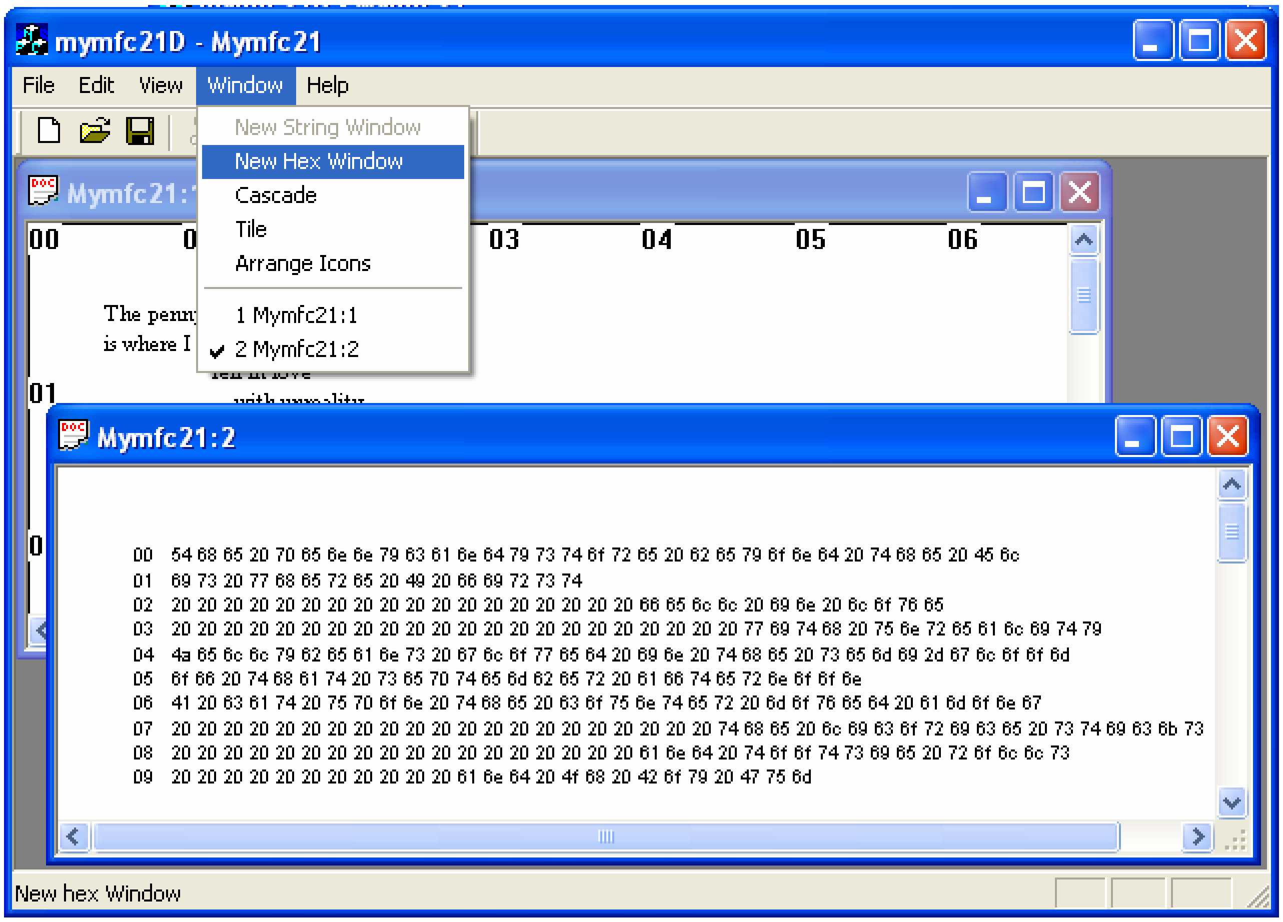
```
// CMainFrame message handlers

void CMainFrame::OnWindowNewHex()
{
    // TODO: Add your command handler code here
    CMDIChildWnd* pActiveChild = MDIGetActive();
    CDocument* pDocument;
    if (pActiveChild == NULL ||
            (pDocument = pActiveChild->GetActiveDocument()) == NULL) {
        TRACE("Warning:  No active document for WindowNew command\n");
        AfxMessageBox(AFX_IDP_COMMAND_FAILURE);
        return; // Command failed
    }

    // Otherwise, we have a new frame!
    CDocTemplate* pTemplate = ((CMymfc21DApp*) AfxGetApp())->m_pTemplateHex;
    ASSERT_VALID(pTemplate);
    CFrameWnd* pFrame = pTemplate->CreateNewFrame(pDocument, pActiveChild);
    if (pFrame == NULL)
   {
        TRACE("Warning:  failed to create new frame\n");
        AfxMessageBox(AFX_IDP_COMMAND_FAILURE);
        return; // Command failed
    }

    pTemplate->InitialUpdateFrame(pFrame, pDocument);
}
```

Listing 44.

The function cloning above is a **useful MFC programming technique**. You must first find a base class function that does almost what you want, and then copy it from the **\VC98\mfc\src** subdirectory into your derived class, changing it as required. The only danger of cloning is that subsequent versions of the MFC library might implement the original function differently.

Testing the MYMFC21D Application

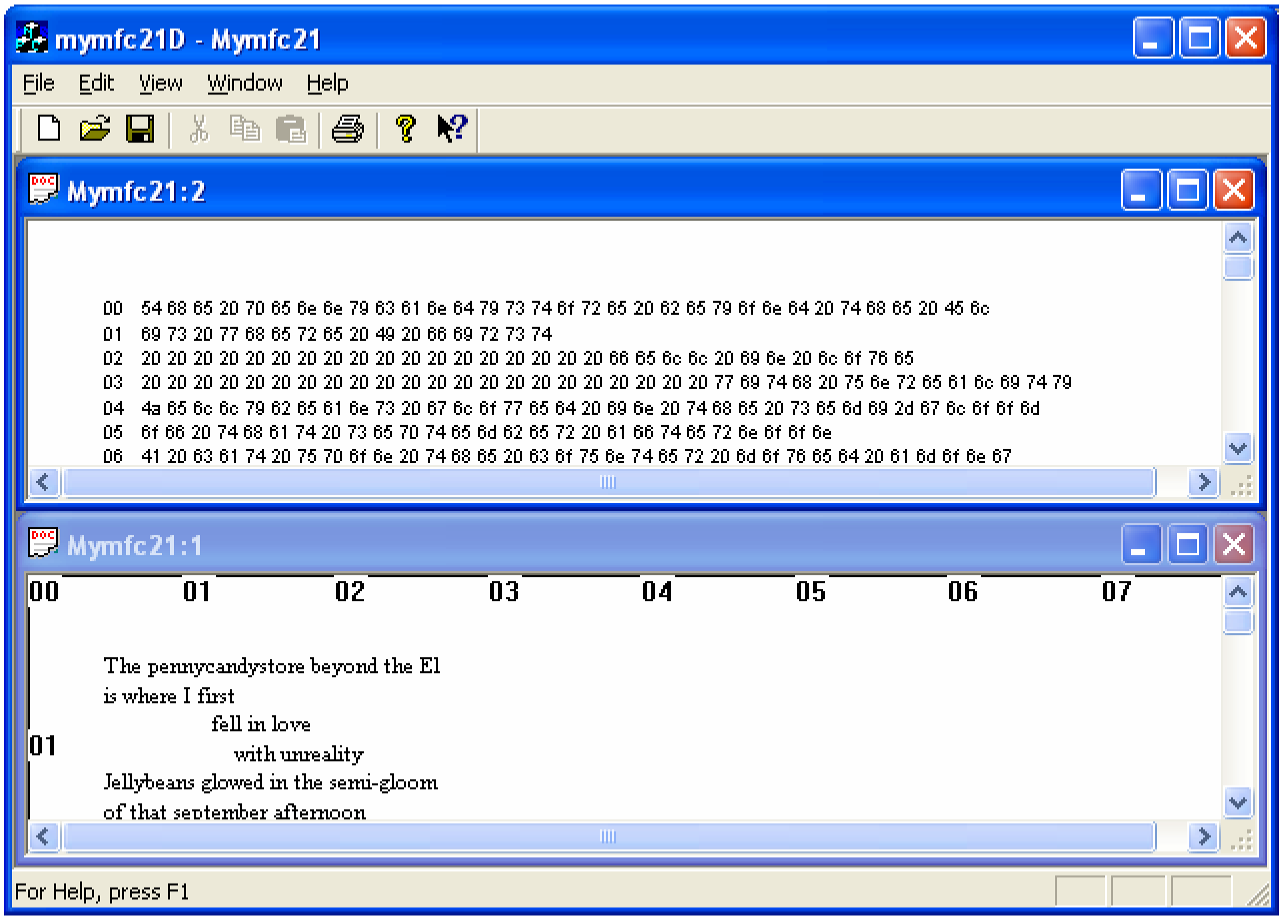
When you start the MYMFC21D application, a text view child window appears. Choose **New Hex Window** from the Window menu. The application should look like this.



```
00   54 68 65 20 70 65 6e 6e 79 63 61 6e 64 79 73 74 6f 72 65 20 62 65 79 6f 6e 64 20 74 68 65 20 45 6c
01   69 73 20 77 68 65 72 65 20 49 20 66 69 72 73 74
02   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 66 65 6c 6c 20 69 6e 20 6c 6f 76 65
03   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 77 69 74 68 20 75 6e 72 65 61 6c 69 74 79
04   4a 65 6c 6c 79 62 65 61 6e 73 20 67 6c 6f 77 65 64 20 69 6e 20 74 68 65 20 73 65 6d 69 2d 67 6c 6f 6f 6d
05   6f 66 20 74 68 61 74 20 73 65 70 74 65 6d 62 65 72 20 61 66 74 65 72 6e 6f 6f 6e
06   41 20 63 61 74 20 75 70 6f 6e 20 74 68 65 20 63 6f 75 6e 74 65 72 20 6d 6f 76 65 64 20 61 6d 6f 6e 67
07   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 74 68 65 20 6c 69 63 6f 72 69 63 65 20 73 74 69 63 6b 73
08   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 61 6e 64 20 74 6f 6f 74 73 69 65 20 72 6f 6c 6c 73
09   20 20 20 20 20 20 20 20 20 20 20 61 6e 64 20 4f 68 20 42 6f 79 20 47 75 6d
```

Figure 47: MYMFC21D MDI program output with cascade view.

Try the **Cascade** and **Tile** menus.



```
00  54 68 65 20 70 65 6e 6e 79 63 61 6e 64 79 73 74 6f 72 65 20 62 65 79 6f 6e 64 20 74 68 65 20 45 6c
01  69 73 20 77 68 65 72 65 20 49 20 66 69 72 73 74
02  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 66 65 6c 6c 20 69 6e 20 6c 6f 76 65
03  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 77 69 74 68 20 75 6e 72 65 61 6c 69 74 79
04  4a 65 6c 6c 79 62 65 61 6e 73 20 67 6c 6f 77 65 64 20 69 6e 20 74 68 65 20 73 65 6d 69 2d 67 6c 6f 6f 6d
05  6f 66 20 74 68 61 74 20 73 65 70 74 65 6d 62 65 72 20 61 66 74 65 72 6e 6f 6f 6e
06  41 20 63 61 74 20 75 70 6f 6e 20 74 68 65 20 63 6f 75 6e 74 65 72 20 6d 6f 76 65 64 20 61 6d 6f 6e 67
```

```
00        01        02        03        04        05        06        07

      The pennycandystore beyond the El
      is where I first
                    fell in love
01                      with unreality
      Jellybeans glowed in the semi-gloom
      of that september afternoon
```

Figure 48: MYMFC21D MDI program output with tile view.

Further reading and digging:

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