# Module 7a: Menus, Keyboard Accelerators, the Rich Edit Control, and Property Sheets – Part 2

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:



## Property Sheets

You've already seen property sheets in Visual C++ and in many other modern Windows-based programs. A property sheet is a nice UI element that allows you to cram lots of categorized information into a small dialog. The user selects pages by clicking on their tabs. Windows offers a tab control that you can insert in a dialog, but it's more likely that you'll want to put dialogs inside the tab control. The MFC library supports this, and the result is called a **property sheet**. The individual dialogs are called **property pages**.

## Building a Property Sheet

Follow these general steps to build a property sheet using the Visual C++ tools:

1. Use the resource editor to create a series of dialog templates that are all approximately the same size. The captions are the strings that you want to display on the tabs.
2. Use ClassWizard to generate a class for each template. Select `CPropertyPage` as the base class. Add data members for the controls.
3. Use ClassWizard to generate a single class derived from `CPropertySheet`.
4. To the sheet class, add one data member for each page class.
5. In the sheet class constructor, call the `AddPage()` member function for each page, specifying the address of the embedded page object.
6. In your application, construct an object of the derived `CPropertySheet` class, and then call `DoModal()`. You must specify a caption in the constructor call, but you can change the caption later by calling `CPropertySheet::SetTitle`.
7. Take care of programming for the **Apply** button.

## Property Sheet Data Exchange

The framework puts three buttons on a property sheet (See Figure 1 for example). Be aware that the framework calls the **Dialog Data Exchange** (DDX) code for a property page each time the user switches to and from that page. As you would expect, the framework calls the DDX code for a page when the user clicks OK, thus updating that page's data members. From these statements, you can conclude that all data members for all pages are updated when the user clicks OK to exit the sheet. All this with no C++ programming on your part!

With a normal modal dialog, if the user clicks the Cancel button, the changes are discarded and the dialog class data members remain unchanged. With a property sheet, however, the data members are updated if the user changes one page and then moves to another, even if the user exits by clicking the Cancel button.

What does the Apply button do? Nothing at all if you don't write some code. It won't even be enabled. To enable it for a given page, you must set the page's modified flag by calling `SetModified(TRUE)` when you detect that the user has made changes on the page.

If you've enabled the Apply button, you can write a handler function for it in your page class by overriding the virtual `CPropertyPage::OnApply` function. Don't try to understand property page message processing in the context of

normal modal dialogs; it's quite different. The framework gets a `WM_NOTIFY` message for all button clicks. It calls the DDX code for the page if the OK or Apply button was clicked. It then calls the virtual `OnApply()` functions for all the pages, and it resets the modified flag, which disables the Apply button. Don't forget that the DDX code has already been called to update the data members in all pages, so you need to override `OnApply()` in only one page class. What you put in your `OnApply()` function is your business, but one option is to send a user-defined message to the object that created the property sheet. The message handler can get the property page data members and process them. Meanwhile, the property sheet stays on the screen.

### The MYMFCPRO Example Revisited

Now we'll add a property sheet to MYMFCPRO that allows the user to change the rich edit control's font characteristics. Of course, we could have used the standard MFC `CFontDialog()` function, but then you wouldn't have learned how to create property sheets. Figure 1 shows the property sheet that you'll build as you continue with MYMFCPRO.

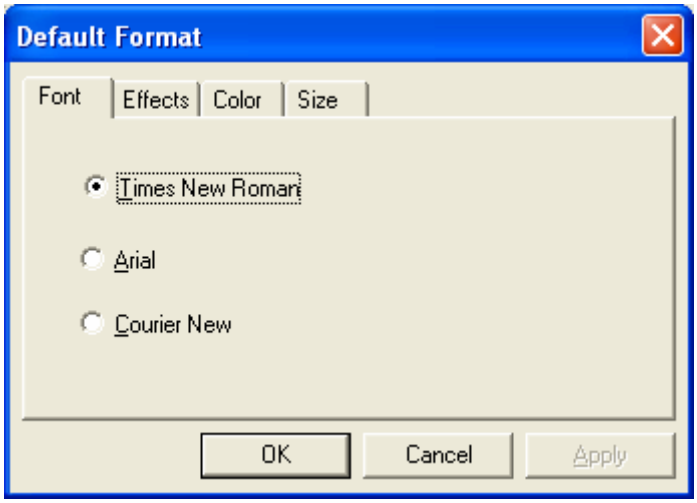


Figure 1: The property sheet from MYMFCPRO.

If you haven't built MYMFCPRO, follow the instructions that begin under the MYMFCPRO example to build it. If you already have MYMFCPRO working with the **Transfer** menu commands, just continue on with the following long steps:

Use the resource editor to edit the application's main menu. Click on the **ResourceView** tab in the **Workspace** window. Edit the `IDR_MAINFRAME` menu resource to add a **Format** menu that looks something like this.

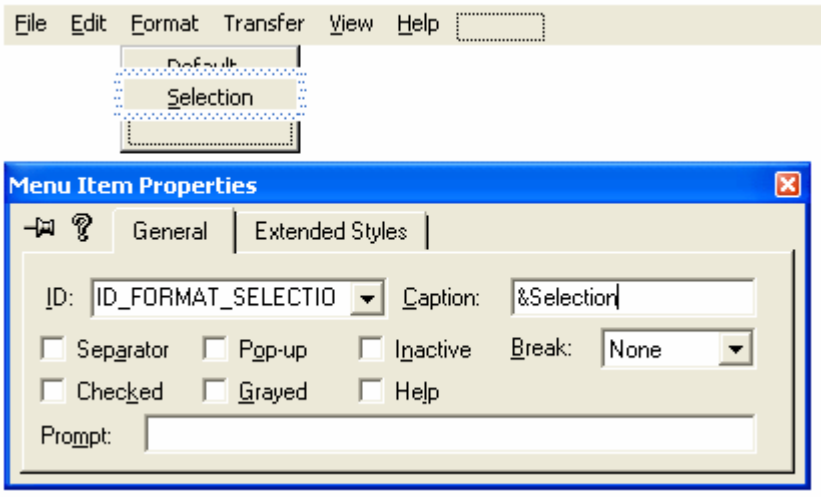


Figure 2: Adding new menus and its items.

Use the following command IDs for the new **Format** menu items.

| Caption | Command ID |
|---|---|
| &Default | ID_FORMAT_DEFAULT |
| &Selection | ID_FORMAT_SELECTION |

Table 1

Add appropriate **prompt strings** for the two menu items as needed.

Use ClassWizard to add the view class command and update command UI message handlers. Select the `CMymfcproView` class, and then add the following member functions.

| Object ID | Message | Member Function |
|---|---|---|
| ID_FORMAT_DEFAULT | COMMAND | OnFormatDefault() |
| ID_FORMAT_SELECTION | COMMAND | OnFormatSelection() |
| ID_FORMAT_SELECTION | UPDATE_COMMAND_UI | OnUpdateFormatSelection() |

Table 1

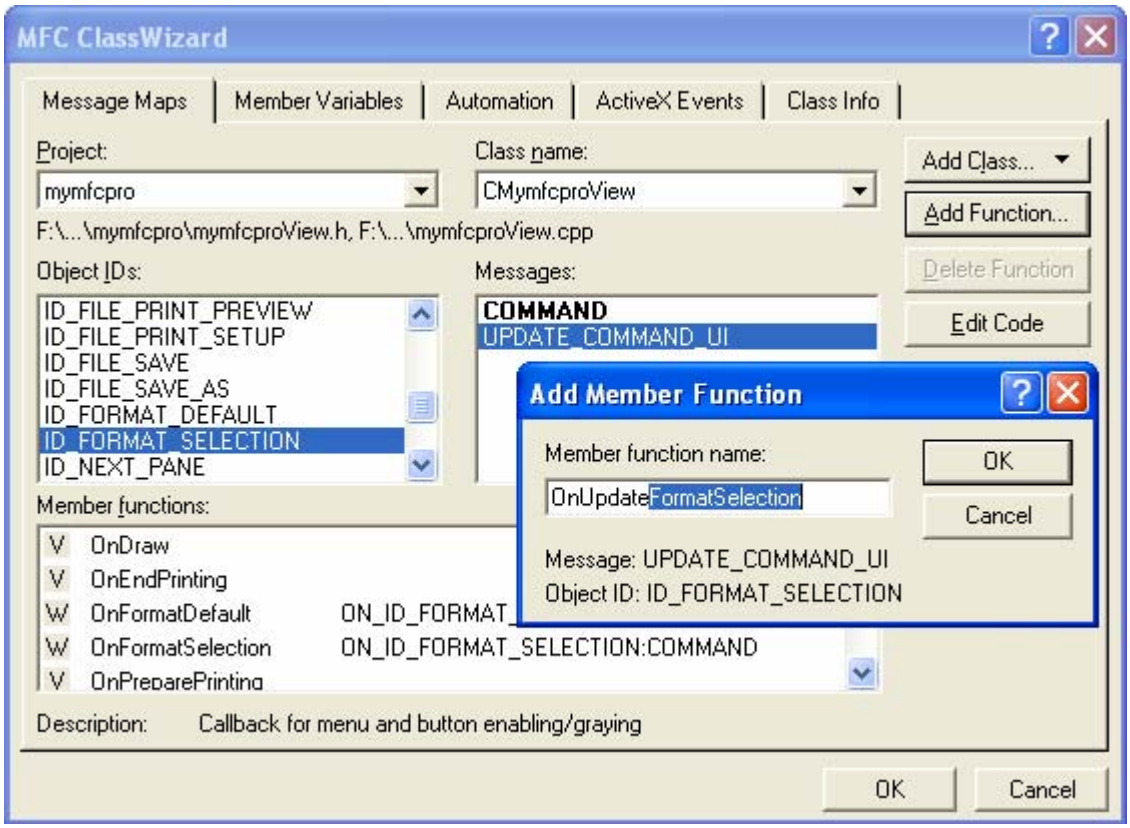


Figure 3: Using ClassWizard to add the view class commands and update command UI message handlers.

Use the resource editor to add four property page dialog templates. The templates are shown here with their associated IDs.

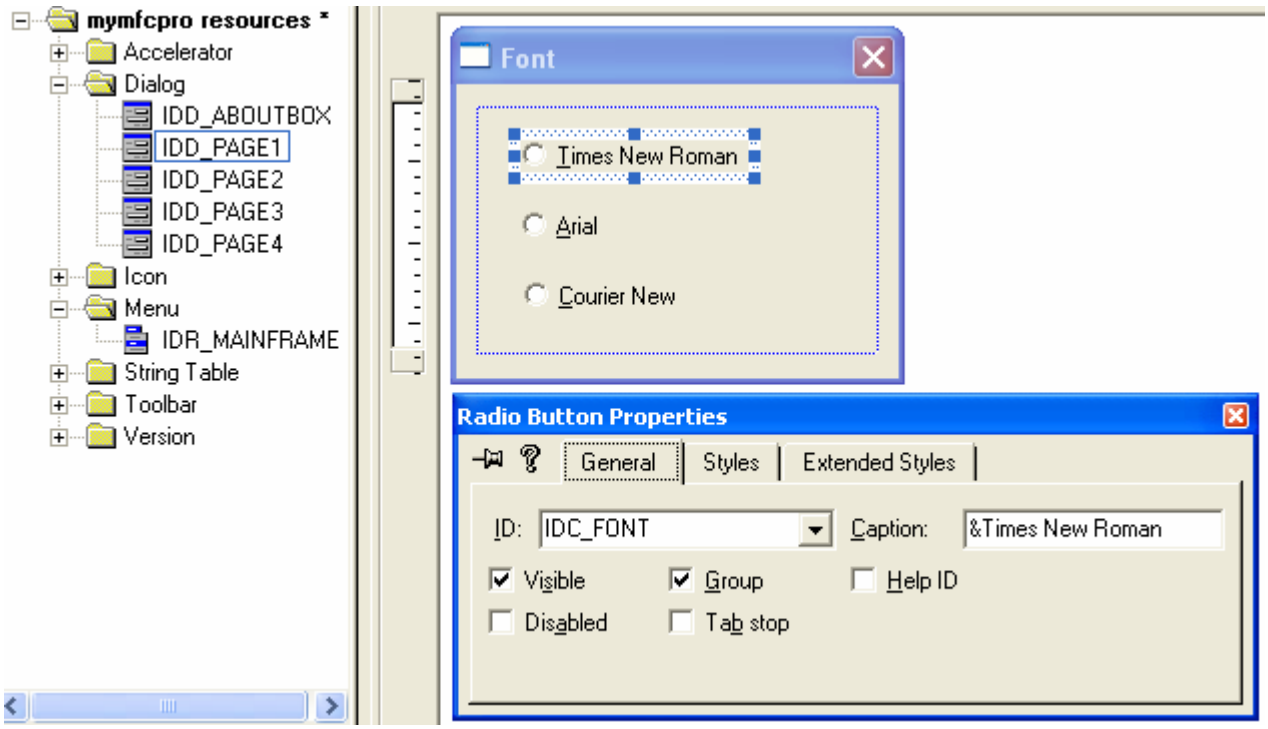


Figure 4: Using the resource editor to add **Font** property page dialog templates.

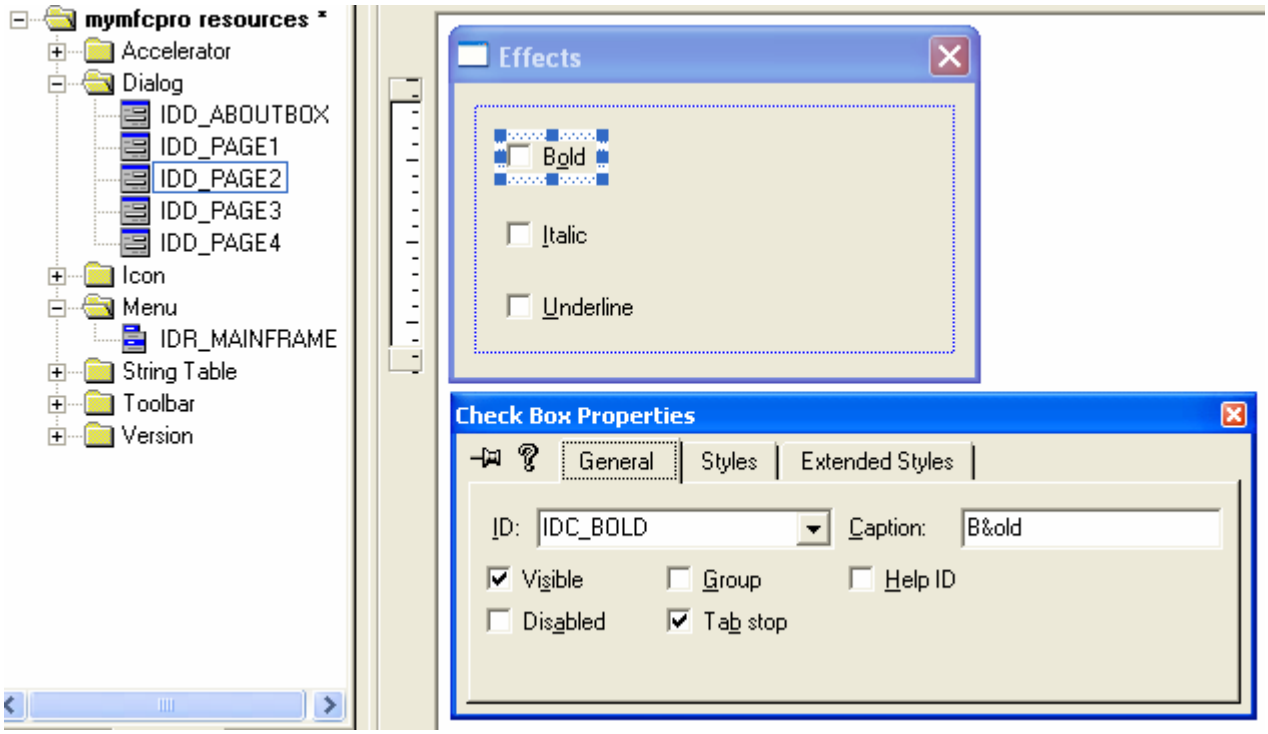


Figure 5: Using the resource editor to add **Effects** property page dialog templates.

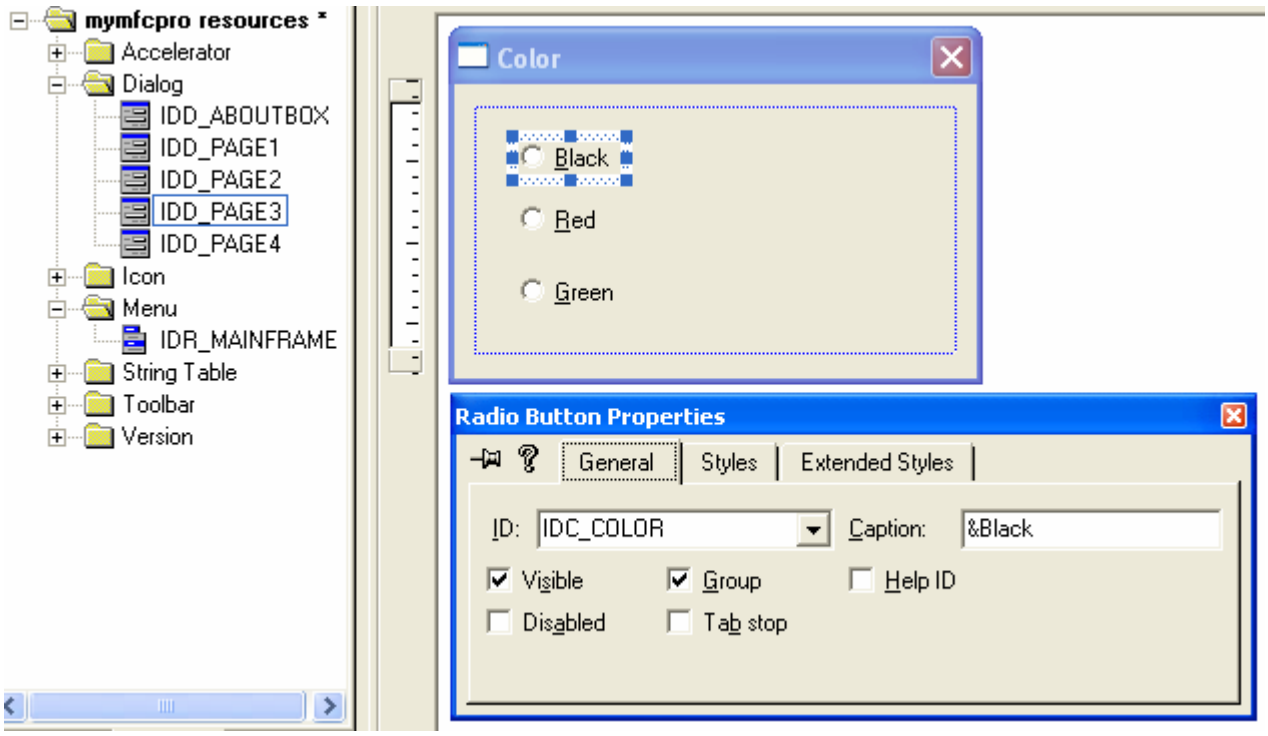


Figure 6: Using the resource editor to add **Color** property page dialog templates.

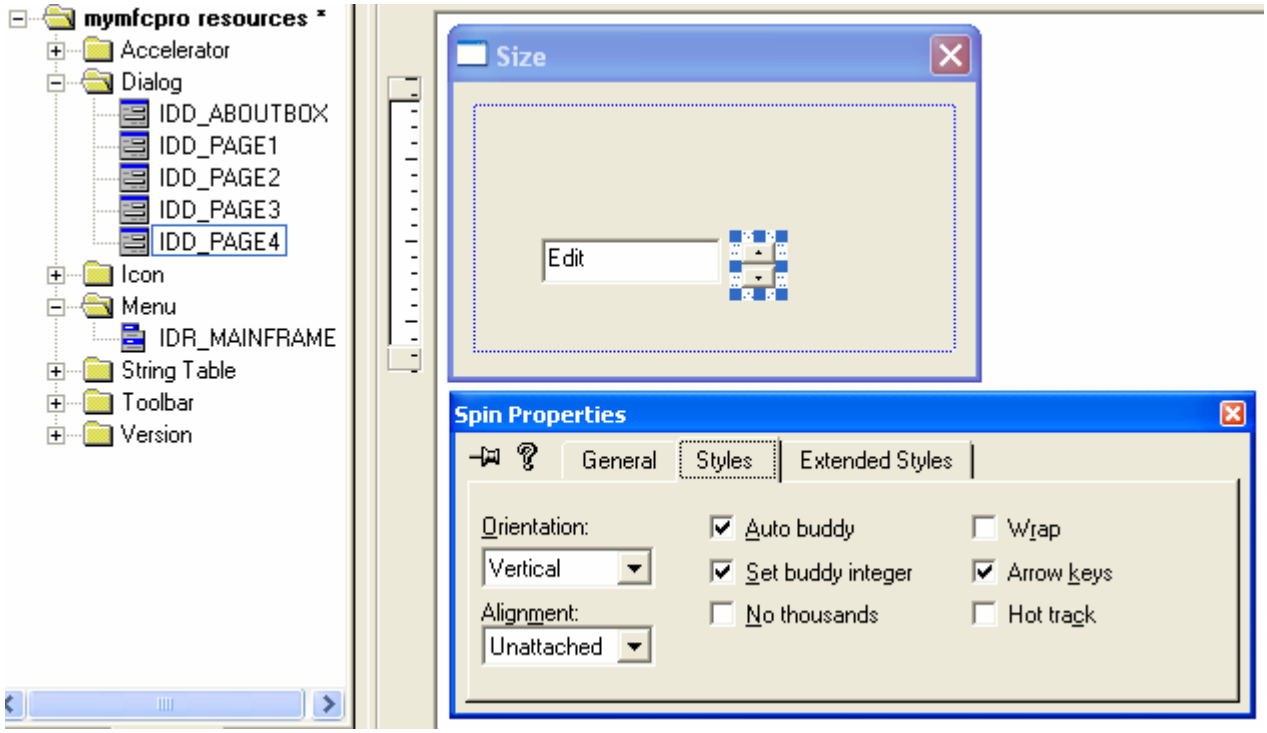


Figure 7: Using the resource editor to add **Size** property page dialog templates.

Use the IDs in the table below for the controls in the dialogs. Set the **Auto Buddy** and the **Set Buddy Integer** properties for the spin button control, and set the **Group** property for the `IDC_FONT` and `IDC_COLOR` radio buttons. Use ClassWizard to create the classes `CPage1`, `CPage2`, `CPage3`, and `CPage4`. In each case, select:

1. `CPropertyPage` as the base class.

2. Click the **Change** button in ClassWizard's **New Class** dialog to generate the code for all these classes in the files **Property.h** and **Property.cpp**.

Then add the data members shown here.

| Dialog ID | Control | Control ID | Type | Data Member |
|---|---|---|---|---|
| IDD_PAGE1 | First radio button | IDC_FONT | int | m_nFont |
| IDD_PAGE2 | Bold check box | IDC_BOLD | BOOL | m_bBold |
| IDD_PAGE2 | Italic check box | IDC_ITALIC | BOOL | m_bItalic |
| IDD_PAGE2 | Underline check box | IDC_UNDERLINE | BOOL | m_bUnderline |
| IDD_PAGE3 | First radio button | IDC_COLOR | int | m_nColor |
| IDD_PAGE4 | Edit control | IDC_FONTSIZE | int | m_nFontSize |
| IDD_PAGE4 | Spin button control | IDC_SPIN1 | - | - |

Table 2

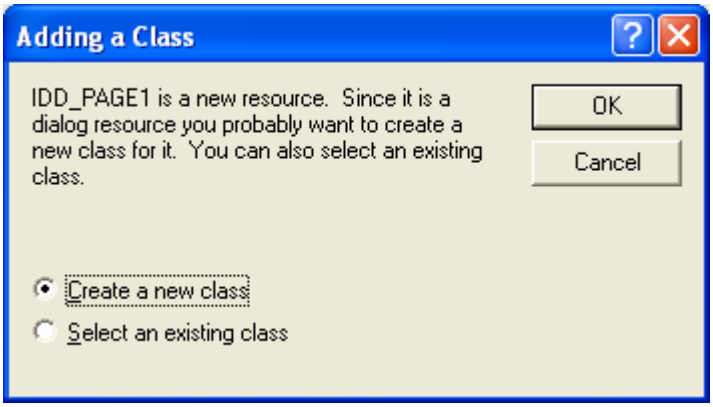


Figure 8: New class creation dialog prompt.

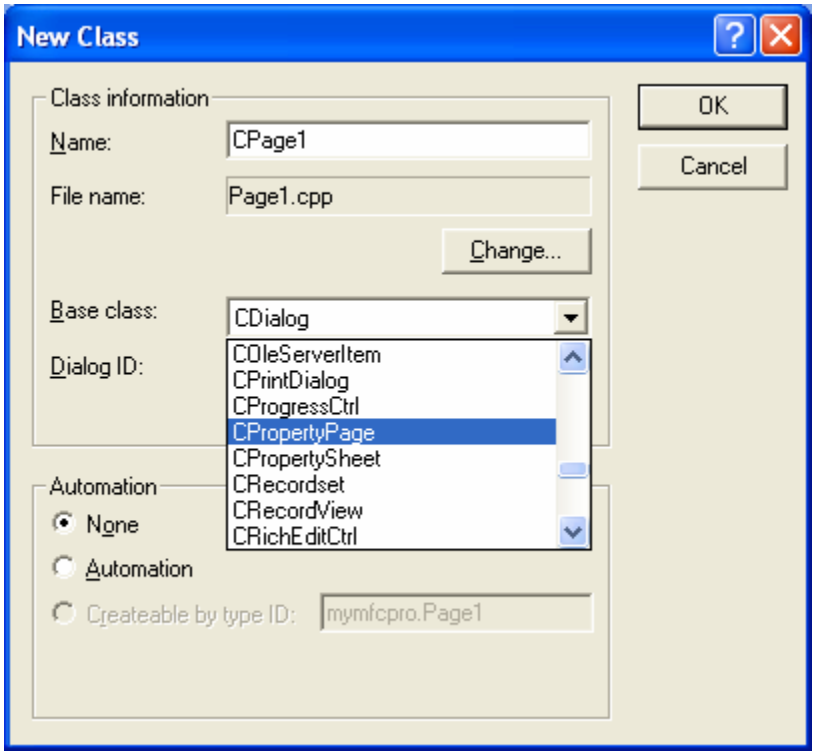


Figure 9: Creating the `CPage1` class.

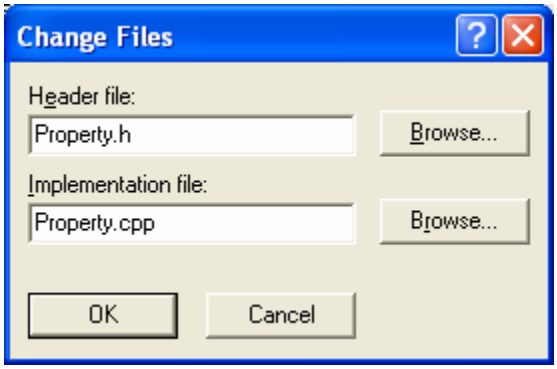


Figure 10: Changing the class header and implementation file names.

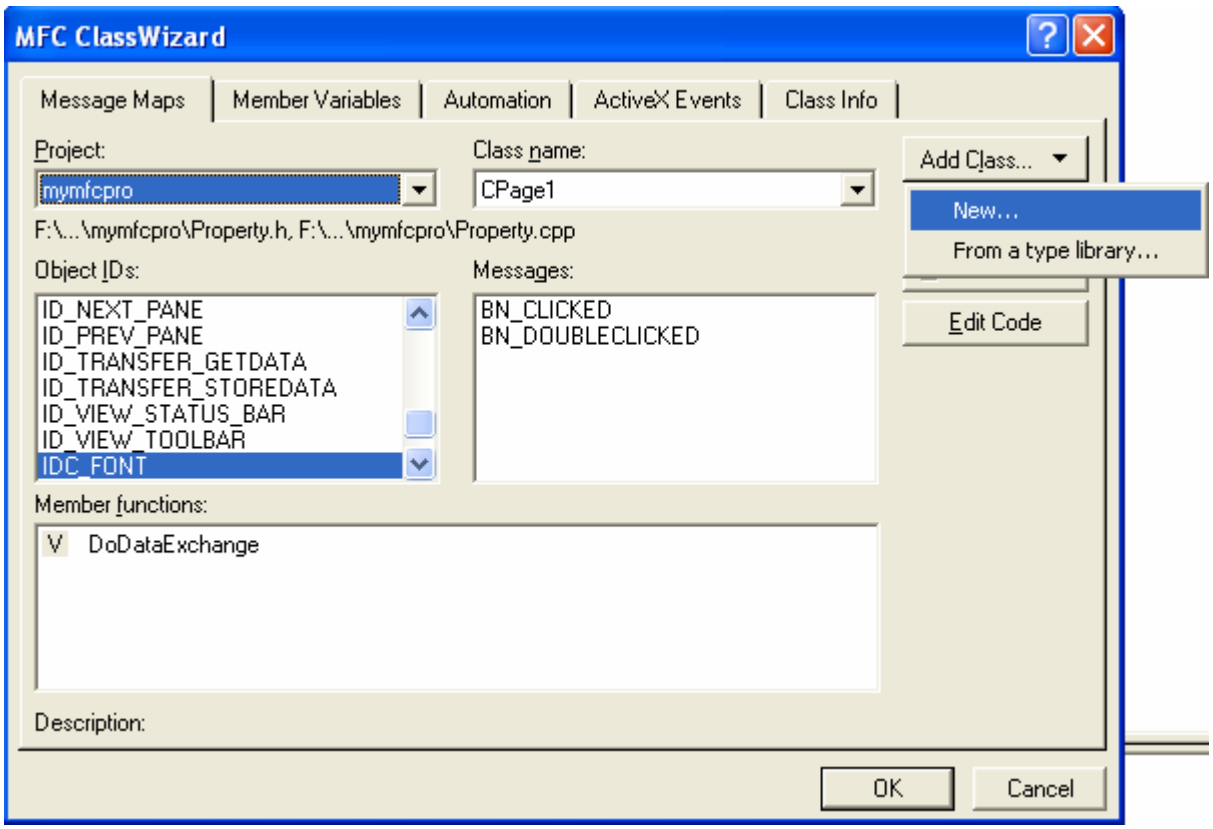


Figure 11: Using ClassWizard to create the new `CPage2`, `CPage3`, and `CPage4` classes.

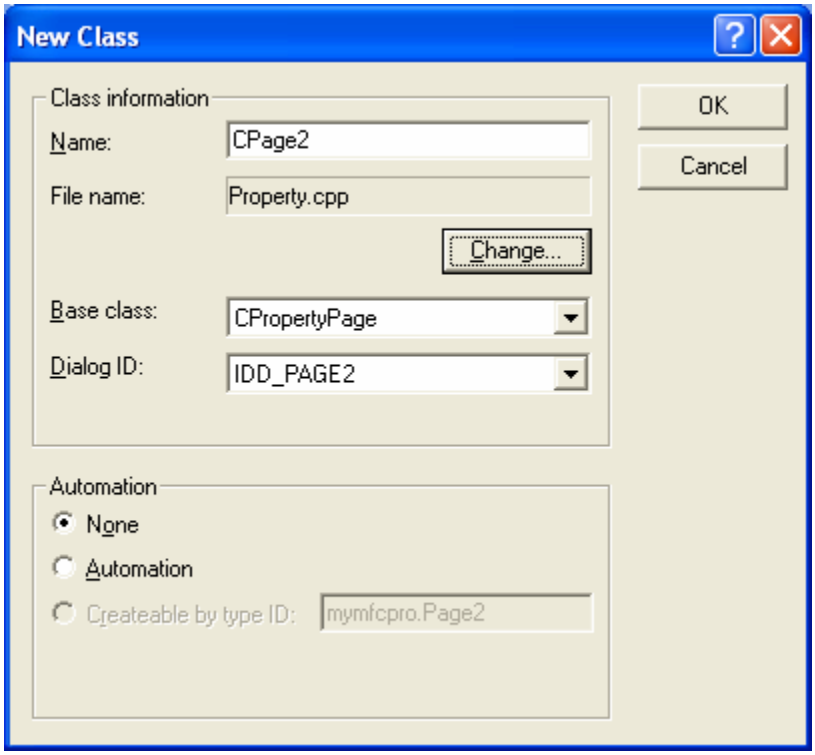


Figure 12: `CPage2` new class information.

Continue for `CPage3` and `CPage4` classes…Then continue for the data member.

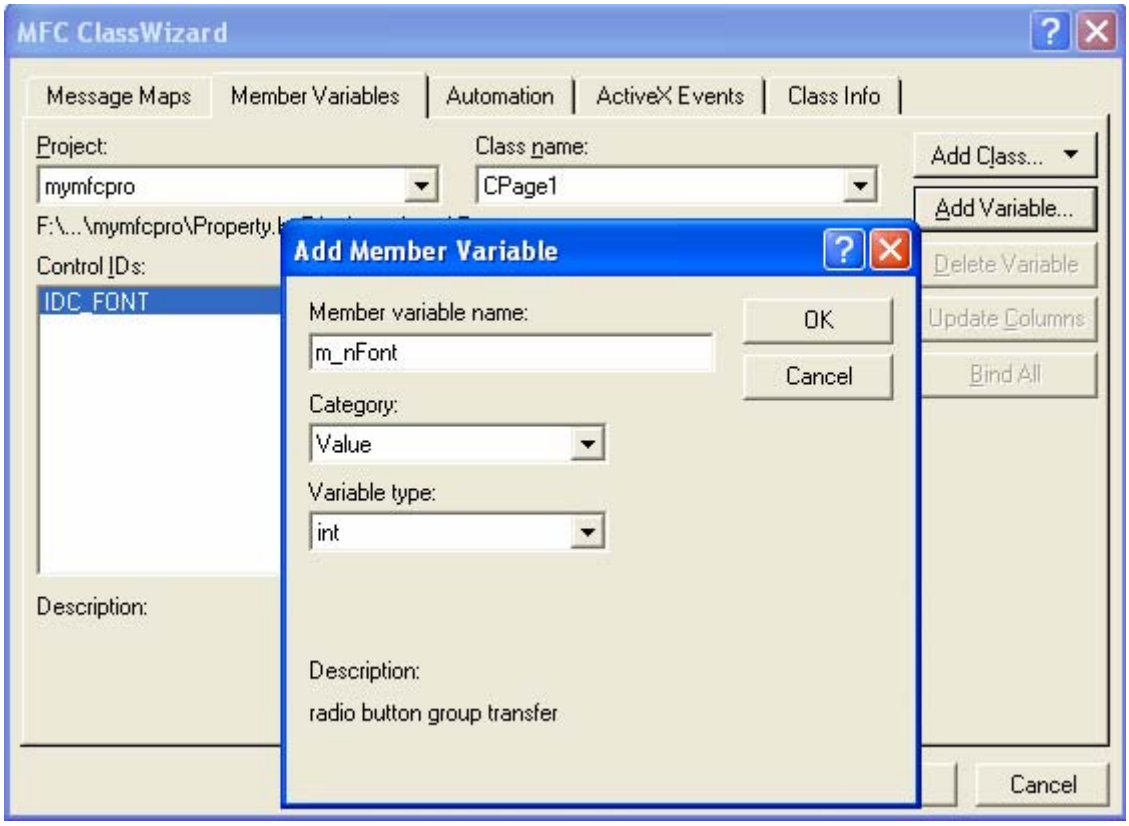


Figure 13: Adding data member variable.

Continue for other classes and for `CPage4`, set the minimum value of `IDC_FONTSIZE` to 8 and its maximum value to 24.

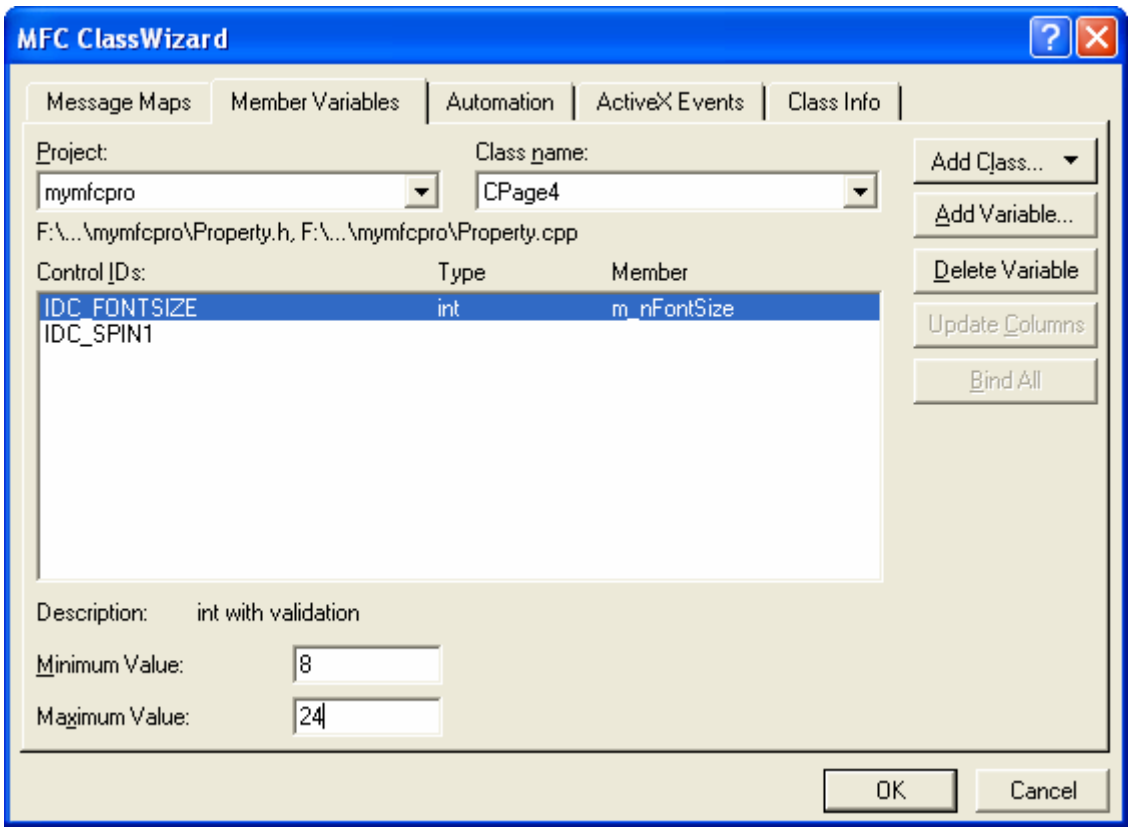


Figure 14: Setting the minimum value of `IDC_FONTSIZE` to 8 and its maximum value to 24.

Finally, use ClassWizard to add an `OnInitDialog()` message handler function for `CPage4`.

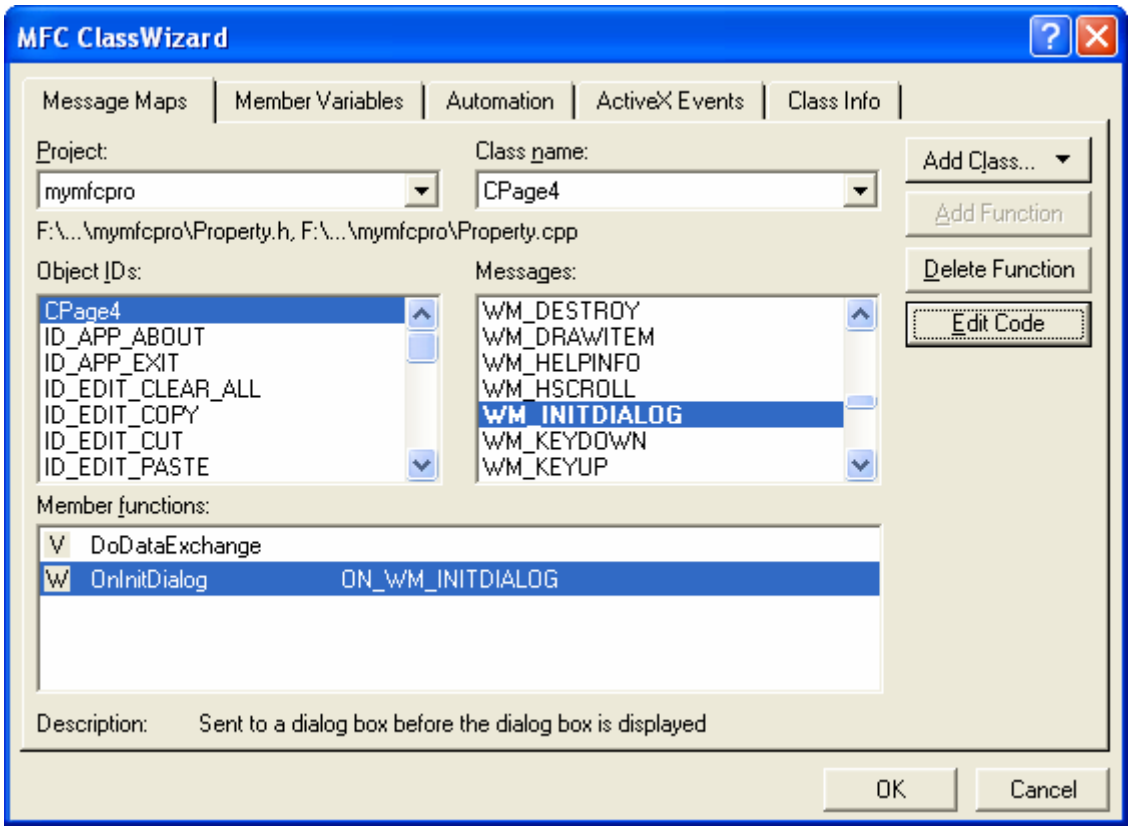

Figure 15: Adding an `OnInitDialog()` message handler function for `CPage4`.

Use ClassWizard to create a class derived from `CPropertySheet`. Choose the name `CFontSheet`. Generate the code in the files **Property.h** and **Property.cpp**, the same files you used for the property page classes. Listing 1 shows these files with the added code in orange color.

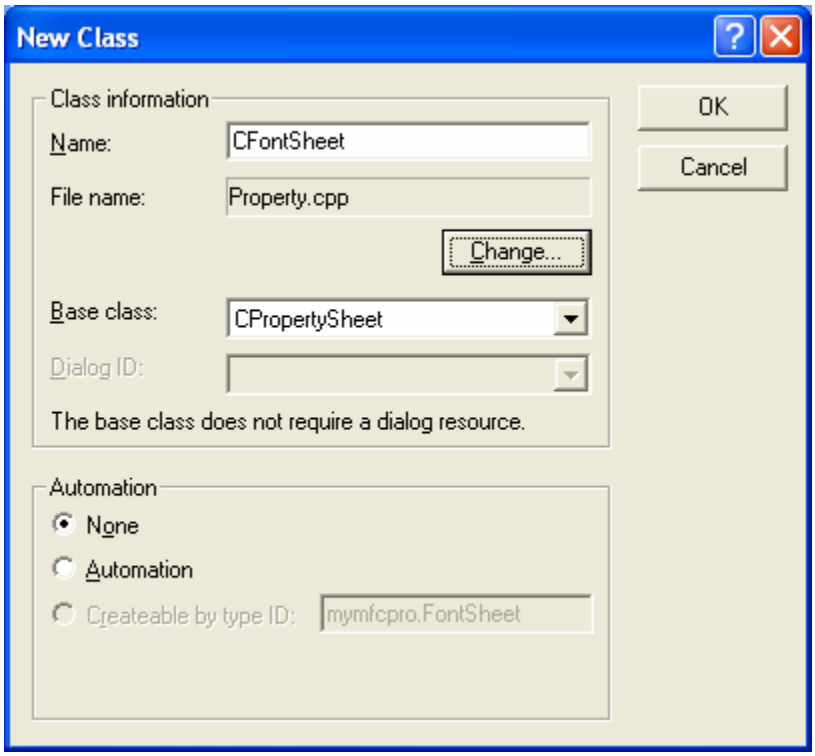


Figure 16: Using ClassWizard to create a `CFontSheet` class that derived from `CPropertySheet`.

**PROPERTY.H**

```
#if !defined(AFX_PROPERTY_H__CD702F99_7495_11D0_8FDC_00C04FC2A0C2__INCLUDED_)
#define AFX_PROPERTY_H_ _CD702F99_7495_11D0_8FDC_00C04FC2A0C2__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Property.h : header file
//

#define WM_USERAPPLY WM_USER + 5
extern CView* g_pView;

/////////////////////////////////////////////////////////////////////////////
// CPage1 dialog

class CPage1 : public CPropertyPage
{
    DECLARE_DYNCREATE(CPage1)

// Construction
public:
    CPage1();
    ~CPage1();

// Dialog Data
    //{{AFX_DATA(CPage1)
```

```
    enum { IDD = IDD_PAGE1 };
    int     m_nFont;
    //}}AFX_DATA


// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPage1)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
                                                     // support
    //}}AFX_VIRTUAL
    virtual BOOL OnApply();
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPage1)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////
// CPage2 dialog

class CPage2 : public CPropertyPage
{
    DECLARE_DYNCREATE(CPage2)

// Construction
public:
    CPage2();
    ~CPage2();

// Dialog Data
    //{{AFX_DATA(CPage2)
    enum { IDD = IDD_PAGE2 };
    BOOL    m_bBold;
    BOOL    m_bItalic;
    BOOL    m_bUnderline;
    //}}AFX_DATA


// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPage2)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
                                                     // support
    //}}AFX_VIRTUAL
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPage2)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////
// CPage3 dialog
```

```
class CPage3 : public CPropertyPage
{
    DECLARE_DYNCREATE(CPage3)

// Construction
public:
    CPage3();
    ~CPage3();

// Dialog Data
    //{{AFX_DATA(CPage3)
    enum { IDD = IDD_PAGE3 };
    int     m_nColor;
    //}}AFX_DATA


// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPage3)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
                                                     // support
    //}}AFX_VIRTUAL
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPage3)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////
// CPage4 dialog

class CPage4 : public CPropertyPage
{
    DECLARE_DYNCREATE(CPage4)

// Construction
public:
    CPage4();
    ~CPage4();

// Dialog Data
    //{{AFX_DATA(CPage4)
    enum { IDD = IDD_PAGE4 };
    int     m_nFontSize;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPage4)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
                                                     // support
    //}}AFX_VIRTUAL
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPage4)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////
// CFontSheet

class CFontSheet : public CPropertySheet
{
    DECLARE_DYNAMIC(CFontSheet)

public:
    CPage1 m_page1;
    CPage2 m_page2;
    CPage3 m_page3;
    CPage4 m_page4;

// Construction
public:
    CFontSheet(UINT nIDCaption, CWnd* pParentWnd = NULL, UINT iSelectPage = 0);
    CFontSheet(LPCTSTR pszCaption, CWnd* pParentWnd = NULL, UINT iSelectPage = 0);

// Attributes
public:

// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFontSheet)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CFontSheet();

    // Generated message map functions
protected:
    //{{AFX_MSG(CFontSheet)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_PROPERTY_H_ _CD702F99_7495_11D0_8FDC_00C04FC2A0C2__INCLUDED_)
```

**PROPERTY.CPP**

```
// Property.cpp : implementation file
//

#include "stdafx.h"
#include "mymfcpro.h"
#include "Property.h"

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

```
CView* g_pView;
/////////////////////////////////////////////////////////////////////
// CPage1 property page

IMPLEMENT_DYNCREATE(CPage1, CPropertyPage)

CPage1::CPage1() : CPropertyPage(CPage1::IDD)
{
    //{{AFX_DATA_INIT(CPage1)
    m_nFont = -1;
    //}}AFX_DATA_INIT
}

CPage1::~CPage1()
{
}

BOOL CPage1::OnApply()
{
    TRACE("CPage1::OnApply\n");
    g_pView->SendMessage(WM_USERAPPLY);
    return TRUE;
}

BOOL CPage1::OnCommand(WPARAM wParam, LPARAM lParam)
{
    SetModified(TRUE);
    return CPropertyPage::OnCommand(wParam, lParam);
}

void CPage1::DoDataExchange(CDataExchange* pDX)
{
    TRACE("Entering CPage1::DoDataExchange -- %d\n", pDX->m_bSaveAndValidate);
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPage1)
    DDX_Radio(pDX, IDC_FONT, m_nFont);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPage1, CPropertyPage)
    //{{AFX_MSG_MAP(CPage1)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////
// CPage1 message handlers

/////////////////////////////////////////////////////////////////////
// CPage2 property page

IMPLEMENT_DYNCREATE(CPage2, CPropertyPage)

CPage2::CPage2() : CPropertyPage(CPage2::IDD)
{
    //{{AFX_DATA_INIT(CPage2)
    m_bBold = FALSE;
    m_bItalic = FALSE;
    m_bUnderline = FALSE;
    //}}AFX_DATA_INIT
}

CPage2::~CPage2()
{
}
```

```
BOOL CPage2::OnCommand(WPARAM wParam, LPARAM lParam)
{
    SetModified(TRUE);
    return CPropertyPage::OnCommand(wParam, lParam);
}

void CPage2::DoDataExchange(CDataExchange* pDX)
{
    TRACE("Entering CPage2::DoDataExchange -- %d\n", pDX->m_bSaveAndValidate);
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPage2)
    DDX_Check(pDX, IDC_BOLD, m_bBold);
    DDX_Check(pDX, IDC_ITALIC, m_bItalic);
    DDX_Check(pDX, IDC_UNDERLINE, m_bUnderline);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPage2, CPropertyPage)
    //{{AFX_MSG_MAP(CPage2)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPage2 message handlers

/////////////////////////////////////////////////////////////////////////////
// CPage3 property page

IMPLEMENT_DYNCREATE(CPage3, CPropertyPage)

CPage3::CPage3() : CPropertyPage(CPage3::IDD)
{
    //{{AFX_DATA_INIT(CPage3)
    m_nColor = -1;
    //}}AFX_DATA_INIT
}

CPage3::~CPage3()
{
}

BOOL CPage3::OnCommand(WPARAM wParam, LPARAM lParam)
{
    SetModified(TRUE);
    return CPropertyPage::OnCommand(wParam, lParam);
}

void CPage3::DoDataExchange(CDataExchange* pDX)
{
   TRACE("Entering CPage3::DoDataExchange -- %d\n", pDX->m_bSaveAndValidate);
   CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPage3)
    DDX_Radio(pDX, IDC_COLOR, m_nColor);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPage3, CPropertyPage)
    //{{AFX_MSG_MAP(CPage3)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////
// CPage3 message handlers

/////////////////////////////////////////////////////////////////////////
// CPage4 property page

IMPLEMENT_DYNCREATE(CPage4, CPropertyPage)

CPage4::CPage4() : CPropertyPage(CPage4::IDD)
{
    //{{AFX_DATA_INIT(CPage4)
    m_nFontSize = 0;
    //}}AFX_DATA_INIT
}

CPage4::~CPage4()
{
}

BOOL CPage4::OnCommand(WPARAM wParam, LPARAM lParam)
{
    SetModified(TRUE);
    return CPropertyPage::OnCommand(wParam, lParam);
}

void CPage4::DoDataExchange(CDataExchange* pDX)
{
    TRACE("Entering CPage4::DoDataExchange -- %d\n", pDX->m_bSaveAndValidate);
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPage4)
    DDX_Text(pDX, IDC_FONTSIZE, m_nFontSize);
    DDV_MinMaxInt(pDX, m_nFontSize, 8, 24);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPage4, CPropertyPage)
    //{{AFX_MSG_MAP(CPage4)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CPage4 message handlers

BOOL CPage4::OnInitDialog()
{
    CPropertyPage::OnInitDialog();
    ((CSpinButtonCtrl*) GetDlgItem(IDC_SPIN1))->SetRange(8, 24);
    return TRUE;  // return TRUE unless you set the focus to a control
                  //  EXCEPTION: OCX Property Pages should return FALSE
}
/////////////////////////////////////////////////////////////////////////
// CFontSheet

IMPLEMENT_DYNAMIC(CFontSheet, CPropertySheet)

CFontSheet::CFontSheet(UINT nIDCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
{
}

CFontSheet::CFontSheet(LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{
    AddPage(&m_page1);
    AddPage(&m_page2);
```

```
    AddPage(&m_page3);
    AddPage(&m_page4);
}

CFontSheet::~CFontSheet()
{
}

BEGIN_MESSAGE_MAP(CFontSheet, CPropertySheet)
    //{{AFX_MSG_MAP(CFontSheet)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFontSheet message handlers
```

Listing 1: The MYMFCPRO header and implementation file listings for the property page and property sheet classes.

Add two data members and two prototypes to the `CMymfcproView` class.

```
private:
    CFontSheet m_sh;
    BOOL m_bDefault; // TRUE default format, FALSE selection
```

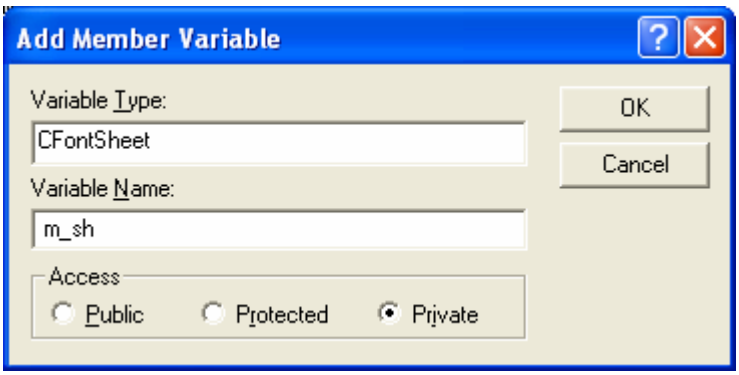


Figure 17: Adding `m_sh` member variable.

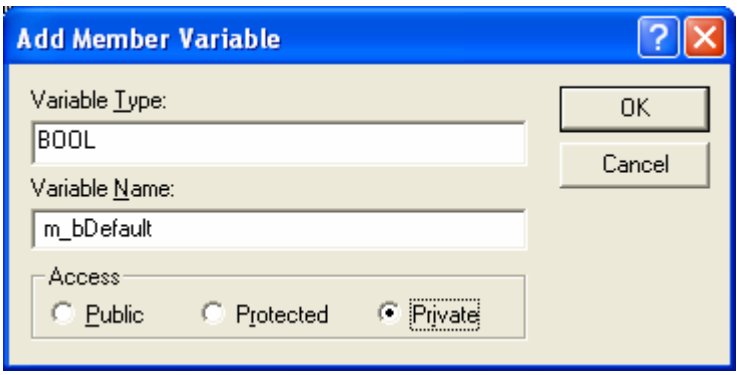


Figure 18: Adding `m_bDefault` member variables.

If you use ClassView for the data members, the `#include` for **Property.h** will be added automatically as shown below.

```
// mymfcproView.h : interface of the CMymfcpro
//
/////////////////////////////////////////////////

#if !defined(AFX_MYMFCPROVIEW_H__BE15154D_0538
#define AFX_MYMFCPROVIEW_H__BE15154D_0538_4AFD

#include "Property.h"    // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

Listing 2.

Now add the prototype for the private function **Format**:

```
private:
      void Format(CHARFORMAT &cf);
```

```
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL m_bDefault;
    CFontSheet m_sh;

private:
    void Format(CHARFORMAT &cf);
};
```

Listing 3.

Insert the prototype for the protected function `OnUserApply()` before the `DECLARE_MESSAGE_MAP` macro.

```
      afx_msg LRESULT OnUserApply(WPARAM wParam, LPARAM lParam);
```

```
    afx_msg void OnFormatDefault();
    afx_msg void OnFormatSelection();
    afx_msg void OnUpdateFormatSelection(CCmdUI* pCmdUI);
    //}}AFX_MSG
    afx_msg LRESULT OnUserApply(WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
private:
    BOOL m_bDefault;
    CFontSheet m_sh;
```

Listing 4.

Edit and add code in the file **mymfcproView.cpp**. Map the user-defined `WM_USERAPPLY` message, as shown here:

```
      ON_MESSAGE(WM_USERAPPLY, OnUserApply)
```

```
BEGIN_MESSAGE_MAP(CMymfcproView, CView)
    //{{AFX_MSG_MAP(CMymfcproView)
    ON_COMMAND(ID_TRANSFER_GETDATA, OnTransferGetdata)
    ON_COMMAND(ID_TRANSFER_STOREDATA, OnTransferStoredata)
    ON_UPDATE_COMMAND_UI(ID_TRANSFER_STOREDATA, OnUpdateTra
    ON_WM_CREATE()
    ON_WM_SIZE()
    ON_COMMAND(ID_FORMAT_DEFAULT, OnFormatDefault)
    ON_COMMAND(ID_FORMAT_SELECTION, OnFormatSelection)
    ON_UPDATE_COMMAND_UI(ID_FORMAT_SELECTION, OnUpdateForma
    ON_MESSAGE(WM_USERAPPLY, OnUserApply)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
```

Listing 5.

Add the following lines to the `OnCreate()` function, just before the `return 0` statement:

```
CHARFORMAT cf;
Format(cf);
m_rich.SetDefaultCharFormat(cf);
```

```
int CMymfcproView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    CRect rect(0, 0, 0, 0);
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;
    m_rich.Create(ES_AUTOVSCROLL | ES_MULTILINE | ES_WANTRETURN |
                  WS_CHILD | WS_VISIBLE | WS_VSCROLL, rect, this, 1);
    CHARFORMAT cf;
    Format(cf);
    m_rich.SetDefaultCharFormat(cf);
    return 0;
}
```

Listing 6.

Edit the view constructor to set default values for the property sheet data members, as follows:

```
CMymfcproView::CMymfcproView() : m_sh("")
{
    m_sh.m_page1.m_nFont = 0;
    m_sh.m_page2.m_bBold = FALSE;
    m_sh.m_page2.m_bItalic = FALSE;
    m_sh.m_page2.m_bUnderline = FALSE;
    m_sh.m_page3.m_nColor = 0;
    m_sh.m_page4.m_nFontSize = 12;
    g_pView = this;
    m_bDefault = TRUE;
}
```

```
CMymfcproView::CMymfcproView() : m_sh("")
{
    // TODO: add construction code here
    m_sh.m_page1.m_nFont = 0;
    m_sh.m_page2.m_bBold = FALSE;
    m_sh.m_page2.m_bItalic = FALSE;
    m_sh.m_page2.m_bUnderline = FALSE;
    m_sh.m_page3.m_nColor = 0;
    m_sh.m_page4.m_nFontSize = 12;
    g_pView = this;
    m_bDefault = TRUE;
}
```

Listing 7.

Edit the format command handlers, as shown here:

```
void CMymfcproView::OnFormatDefault()
{
    m_sh.SetTitle("Default Format");
    m_bDefault = TRUE;
    m_sh.DoModal();
}

void CMymfcproView::OnFormatSelection()
{
    m_sh.SetTitle("Selection Format");
    m_bDefault = FALSE;
    m_sh.DoModal();
}

void CMymfcproView::OnUpdateFormatSelection(CCmdUI* pCmdUI)
{
    long nStart, nEnd;
    m_rich.GetSel(nStart, nEnd);
    pCmdUI->Enable(nStart != nEnd);
}
```

```
void CMymfcproView::OnFormatDefault()
{
    // TODO: Add your command handler code here
    m_sh.SetTitle("Default Format");
    m_bDefault = TRUE;
    m_sh.DoModal();
}

void CMymfcproView::OnFormatSelection()
{
    // TODO: Add your command handler code here
    m_sh.SetTitle("Selection Format");
    m_bDefault = FALSE;
    m_sh.DoModal();
}

void CMymfcproView::OnUpdateFormatSelection(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    long nStart, nEnd;
    m_rich.GetSel(nStart, nEnd);
    pCmdUI->Enable(nStart != nEnd);
}
```

Listing 8.

Add the following handler for the user-defined `WM_USERAPPLY` message:

```
LRESULT CMymfcproView::OnUserApply(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMymfcproView::OnUserApply -- wParam = %x\n", wParam);
    CHARFORMAT cf;
    Format(cf);
    if (m_bDefault)
    {
        m_rich.SetDefaultCharFormat(cf);
    }
    else
    {
```

```
            m_rich.SetSelectionCharFormat(cf);
        }
        return 0;
    }
```

```
    m_rich.GetSel(nStart, nEnd);
    pCmdUI->Enable(nStart != nEnd);
}

LRESULT CMymfcproView::OnUserApply(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMymfcproView::OnUserApply -- wParam = %x\n", wParam);
    CHARFORMAT cf;
    Format(cf);
    if (m_bDefault)
    {
        m_rich.SetDefaultCharFormat(cf);
    }
    else
    {
        m_rich.SetSelectionCharFormat(cf);
    }
    return 0;
}
```

Listing 9.

Add the `Format()` helper function, as shown below, to set a `CHARFORMAT` structure based on the values of the property sheet data members.

```
void CMymfcproView::Format(CHARFORMAT& cf)
{
    cf.cbSize = sizeof(CHARFORMAT);
    cf.dwMask = CFM_BOLD | CFM_COLOR | CFM_FACE |
                CFM_ITALIC | CFM_SIZE | CFM_UNDERLINE;
    cf.dwEffects = (m_sh.m_page2.m_bBold ? CFE_BOLD : 0) |
                   (m_sh.m_page2.m_bItalic ? CFE_ITALIC : 0) |
                   (m_sh.m_page2.m_bUnderline ? CFE_UNDERLINE : 0);
    cf.yHeight = m_sh.m_page4.m_nFontSize * 20;
    switch(m_sh.m_page3.m_nColor) {
    case -1:
    case 0:
        cf.crTextColor = RGB(0, 0, 0);
        break;
    case 1:
        cf.crTextColor = RGB(255, 0, 0);
        break;
    case 2:
        cf.crTextColor = RGB(0, 255, 0);
        break;
    }
    switch(m_sh.m_page1.m_nFont) {
    case -1:
    case 0:
        strcpy(cf.szFaceName, "Times New Roman");
        break;
    case 1:
        strcpy(cf.szFaceName, "Arial");
        break;
    case 2:
        strcpy(cf.szFaceName, "Courier New");
        break;
    }
    cf.bCharSet = 0;
    cf.bPitchAndFamily = 0;
}
```

```
void CMymfcproView::Format(CHARFORMAT& cf)
{
    cf.cbSize = sizeof(CHARFORMAT);
    cf.dwMask = CFM_BOLD | CFM_COLOR | CFM_FACE |
                CFM_ITALIC | CFM_SIZE | CFM_UNDERLINE;
    cf.dwEffects = (m_sh.m_page2.m_bBold ? CFE_BOLD : 0) |
                   (m_sh.m_page2.m_bItalic ? CFE_ITALIC : 0) |
                   (m_sh.m_page2.m_bUnderline ? CFE_UNDERLINE : 0);
    cf.yHeight = m_sh.m_page4.m_nFontSize * 20;
    switch(m_sh.m_page3.m_nColor) {
    case -1:
    case 0:
        cf.crTextColor = RGB(0, 0, 0);
        break;
    case 1:
        cf.crTextColor = RGB(255, 0, 0);
        break;
    case 2:
        cf.crTextColor = RGB(0, 255, 0);
        break;
    }
    switch(m_sh.m_page1.m_nFont) {
    case -1:
    case 0:
        strcpy(cf.szFaceName, "Times New Roman");
        break;
    case 1:
        strcpy(cf.szFaceName, "Arial");
        break;
    case 2:
        strcpy(cf.szFaceName, "Courier New");
        break;
    }
    cf.bCharSet = 0;
    cf.bPitchAndFamily = 0;
}
```

Listing 10.

Build and test the enhanced MYMFCPRO application. Type some text, and then choose **Default** from the **Format** menu.

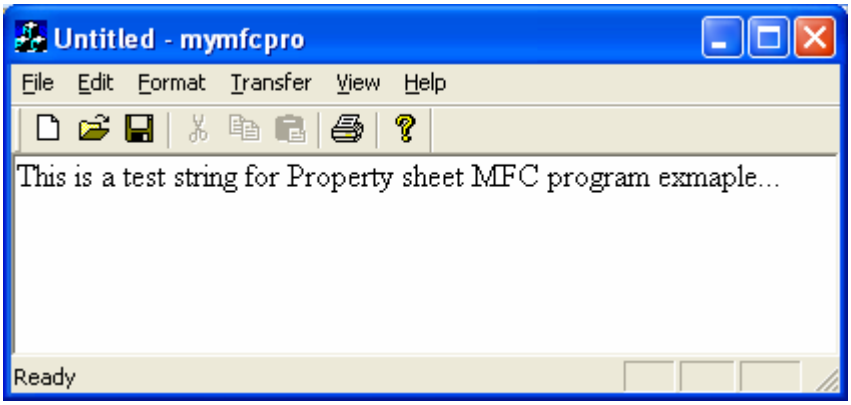


Figure 19: The new MYMFCPRO program output with property sheet.

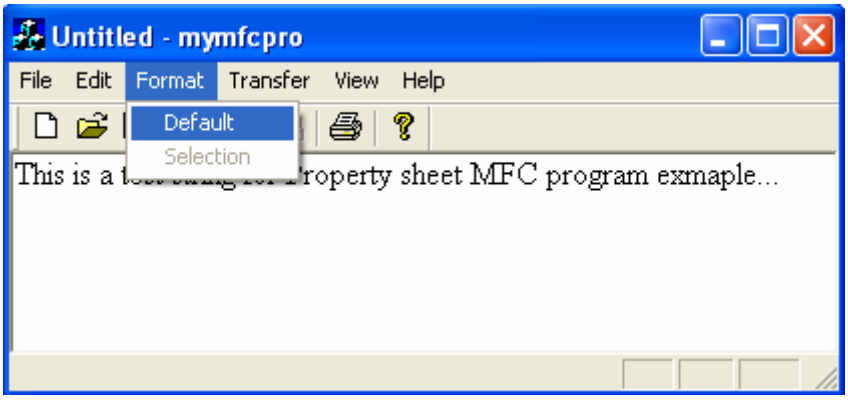


Figure 20: MYMFCPRO program output with **Default** and **Selection** menus.

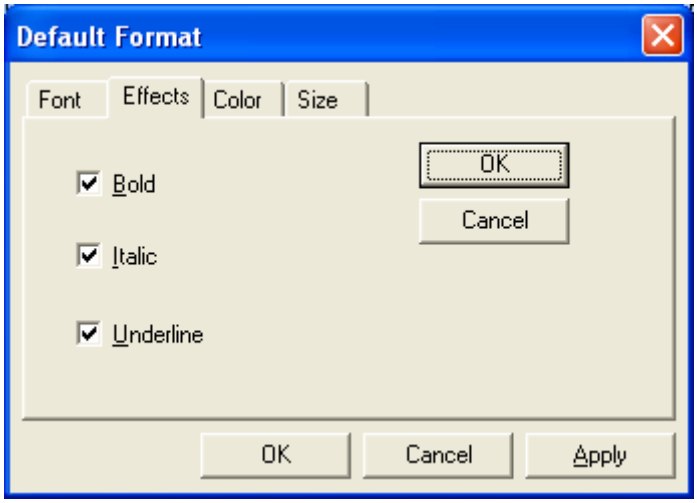


Figure 21: Setting the text's font, effects, color and size through property sheets.

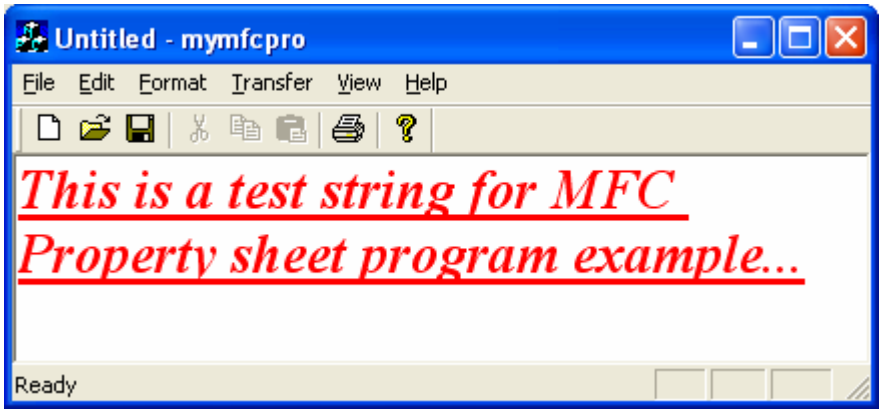


Figure 22: MYMFCPRO program output with property sheet in action.

Observe the TRACE messages in the **Debug** window as you click on property sheet tabs and click the **Apply** button. Try highlighting some text and then formatting the selection.

```
Loaded 'C:\WINDOWS\system32\mslbui.dll', no matching symbol
Warning: no message line prompt for ID 0x8005.
Entering CPage1::DoDataExchange -- 0
Entering CPage1::DoDataExchange -- 1
Entering CPage1::DoDataExchange -- 0
Entering CPage1::DoDataExchange -- 1
Entering CPage3::DoDataExchange -- 0
Entering CPage3::DoDataExchange -- 1
Entering CPage1::DoDataExchange -- 0
Entering CPage1::DoDataExchange -- 1
Entering CPage3::DoDataExchange -- 0
Entering CPage3::DoDataExchange -- 1
Entering CPage4::DoDataExchange -- 0
Entering CPage4::DoDataExchange -- 1
Entering CPage3::DoDataExchange -- 0
Entering CPage3::DoDataExchange -- 1
Entering CPage1::DoDataExchange -- 0
Entering CPage1::DoDataExchange -- 1
CPage1::OnApply
CMymfcproView::OnUserApply -- wParam = 0
The thread 0x218 has exited with code 0 (0x0).
The program 'F:\mfcproject\mymfcpro\Debug\mymfcpro.exe' has
```

Figure 23: Viewing MYMFCPRO debug information (TRACE) through the debug window.

### Apply Button Processing

You might be curious about the way the property sheet classes process the **Apply** button. In all the page classes, the overridden `OnCommand()` functions enable the Apply button whenever a control sends a message to the page. This works fine for pages 1 through 3 in MYMFCPRO, but for page 4, `OnCommand()` is called during the initial conversation between the spin button control and its buddy. The `OnApply()` virtual override in the `CPage1` class sends a user-defined message to the view. The function finds the view in an expedient way, by using a global variable set by the view class. A better approach would be to pass the view pointer to the sheet constructor and then to the page constructor.

The view class calls the property sheet's `DoModal()` function for both default formatting and selection formatting. It sets the `m_bDefault` flag to indicate the mode. We don't need to check the return from `DoModal()` because the user-defined message is sent for both the **OK** button and the **Apply** button. If the user clicks **Cancel**, no message is sent.

### The `CMenu` Class

Up to this point, the application framework and the menu editor have shielded you from the menu class, `CMenu`. A `CMenu` object can represent each Windows menu, including the top-level menu items and associated pop-up menus. Most of the time, the menu's resource is directly attached to a frame window when the window's `Create()` or `LoadFrame()` function is called, and a `CMenu` object is never explicitly constructed. The `CWnd` member function `GetMenu()` returns a temporary `CMenu` pointer. Once you have this pointer, you can freely access and update the menu object.

Suppose you want to switch menus after the application starts. `IDR_MAINFRAME` always identifies the initial menu in the resource script. If you want a second menu, you use the menu editor to create a menu resource with your own ID. Then, in your program, you construct a `CMenu` object, use the `CMenu::LoadMenu` function to load the menu from the resource, and call the `CWnd::SetMenu` function to attach the new menu to the frame window. Then you call the Detach member function to separate the object's `HMENU` handle so that the menu is not destroyed when the `CMenu` object goes out of scope.

You can use a resource to define a menu, and then your program can modify the menu items at runtime. If necessary, however, you can build the whole menu at runtime, without benefit of a resource. In either case, you can use `CMenu` member functions such as `ModifyMenu()`, `InsertMenu()`, and `DeleteMenu()`. Each of these functions operates on an individual menu item identified by ID or by a relative position index. A menu object is actually composed of a nested structure of submenus. You can use the `GetSubMenu()` member function to get a `CMenu` pointer to a pop-up menu contained in the main `CMenu` object. The `CMenu::GetMenuString` function returns the menu item string corresponding to either a zero-based index or a command ID. If you use the command ID option, the menu is searched, together with any submenus.

### Creating Floating Pop-Up Menus

For the floating pop-up menus the user **presses the right mouse button** and a floating menu offers choices that relate to the current selection. It's easy to create these menus using the resource editor and the MFC library `CMenu::TrackPopupMenu` function. Just follow these steps:

Use the menu editor to insert a new, empty menu in your project's resource file.

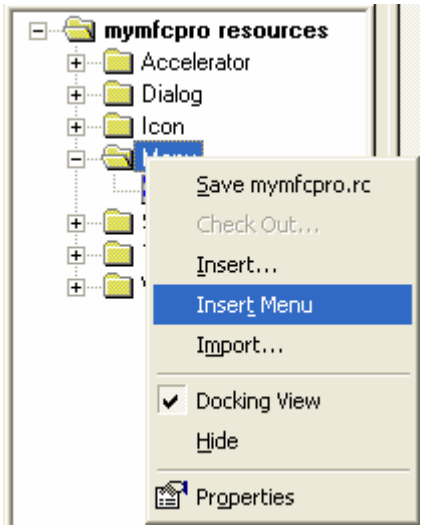


Figure 24: Inserting new menu for floating pop-up menus.

Type some characters in the left top-level item, and then add your menu items in the resulting pop-up menu.

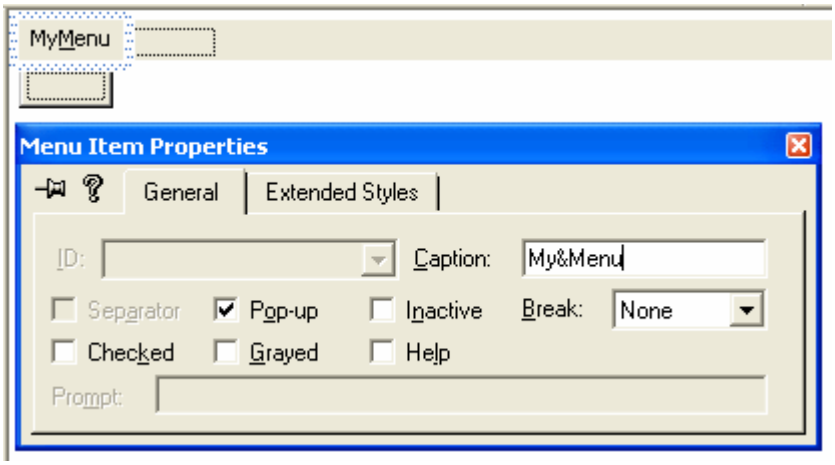


Figure 25: Adding and modifying new pop-up menu properties.

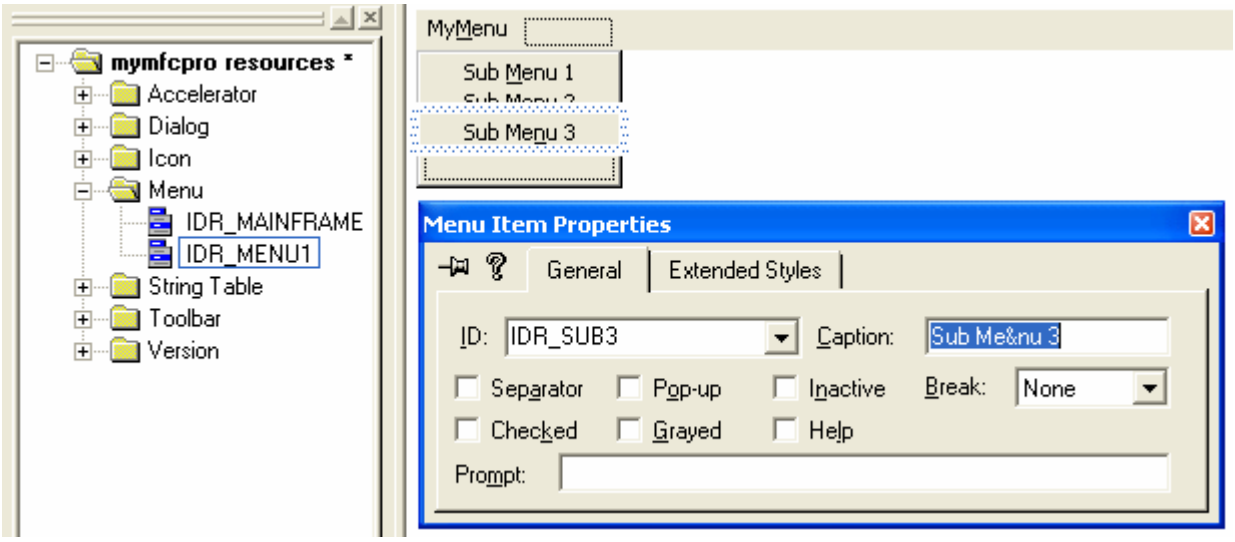


Figure 26: Adding and modifying floating pop-up menu items.

Use ClassWizard to add a `WM_CONTEXTMENU` message handler in your view class or in some other window class that receives mouse-click messages.

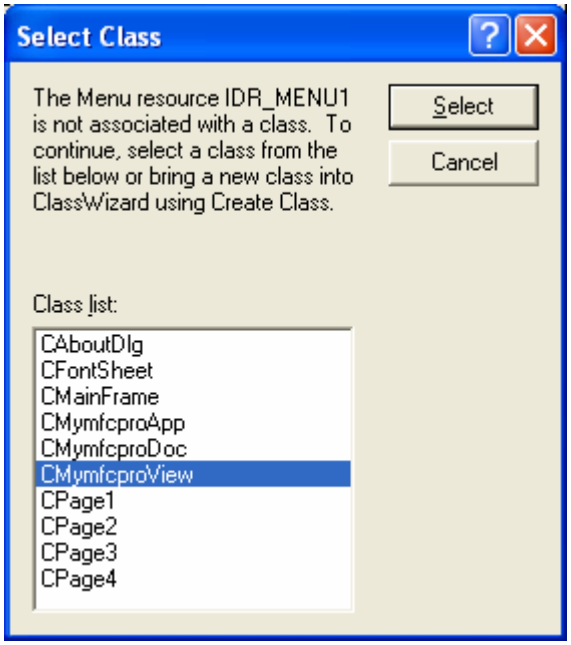


Figure 27: Adding a `WM_CONTEXTMENU` message handler in the view class.

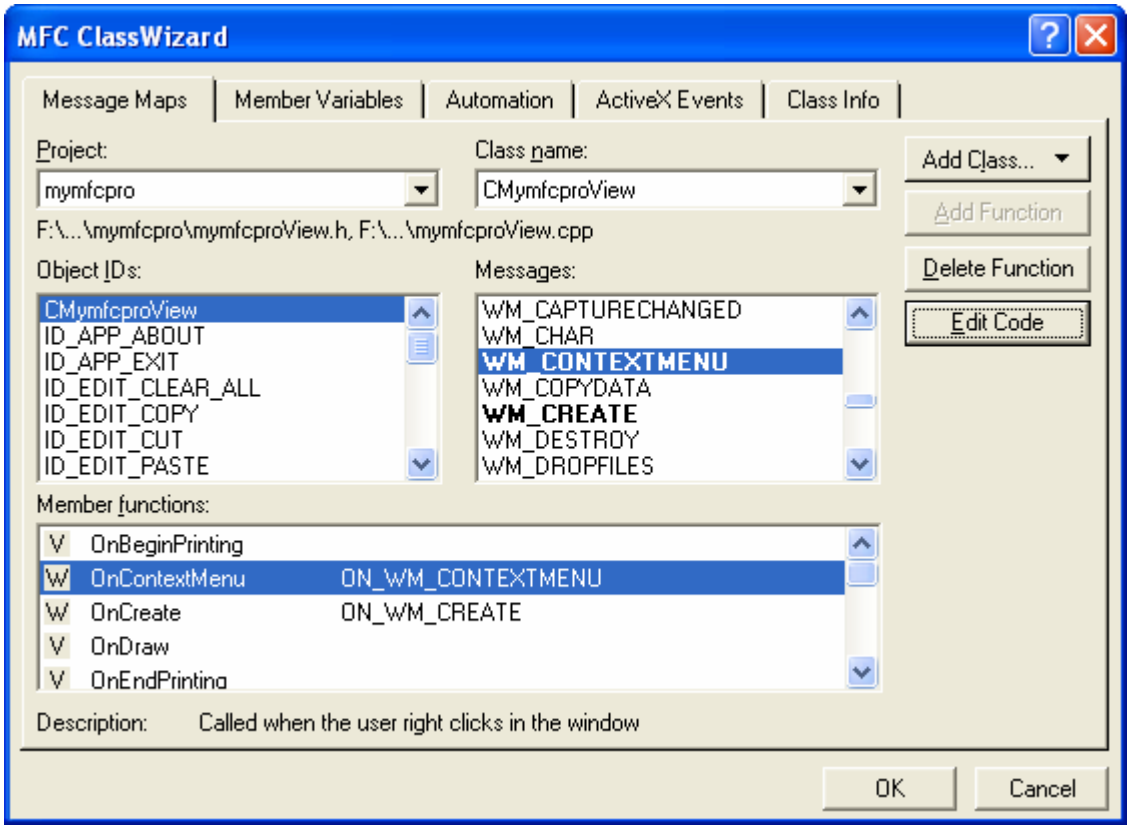


Figure 28: Mapping a message to the view class.

Code the handler as shown below.

```
void CMymfcproView::OnContextMenu(CWnd* pWnd, CPoint point)
{
	// TODO: Add your message handler code here
	CMenu menu;
	// menu.LoadMenu(IDR_MYFLOATINGMENU);
	menu.LoadMenu(IDR_MENU1);
	menu.GetSubMenu(0)->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
point.y, this);
}
```

```
void CMymfcproView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    // TODO: Add your message handler code here
    CMenu menu;
    // menu.LoadMenu(IDR_MYFLOATINGMENU);
    menu.LoadMenu(IDR_MENU1);
    menu.GetSubMenu(0)->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON,
        point.x, point.y, this);
}
```

Listing 11.

You can use ClassWizard to map the floating menu's command IDs the same way you would map the frame menu's command IDs. Build and run the MYMFCPRO.

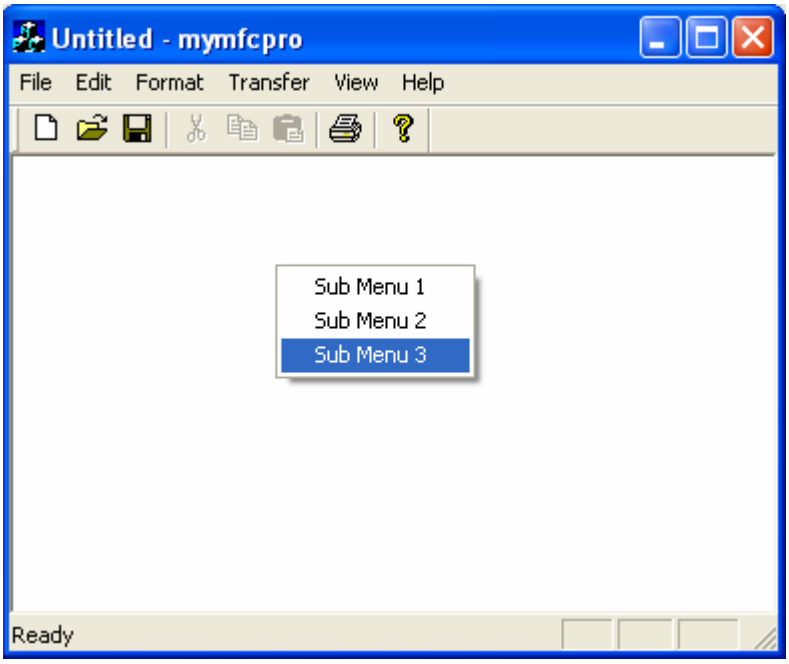


Figure 29: MYMFCPRO program output with floating pop-up menu.

### Extended Command Processing

In addition to the `ON_COMMAND` message map macro, the MFC library provides an extended variation, `ON_COMMAND_EX`. The extended command message map macro provides two features not supplied by the regular command message, a command ID function parameter and the ability to reject a command at runtime, sending it to the next object in the command route. If the extended command handler returns `TRUE`, the command goes no further; if it returns `FALSE`, the application framework looks for another command handler.
The command ID parameter is useful when you want one function to handle several related command messages. You might invent some of your own uses for the rejection feature. ClassWizard can't help you with extended command handlers, so you'll have to do the coding yourself, outside the `AFX_MSG_MAP` brackets. Assume that `IDM_ZOOM_1` and `IDM_ZOOM_2` are related command IDs defined in **resource.h**. Here's the class code you'll need to process both messages with one function, `OnZoom()`:

```
BEGIN_MESSAGE_MAP(CMyView, CView)
    ON_COMMAND_EX(IDM_ZOOM_1, OnZoom)
    ON_COMMAND_EX(IDM_ZOOM_2, OnZoom)
END_MESSAGE_MAP()

BOOL CMyView::OnZoom(UINT nID)
{
    if (nID == IDM_ZOOM_1) {
        // code specific to first zoom command
    }
    else {
        // code specific to second zoom command
    }
    // code common to both commands
    return TRUE; // Command goes no further
}
```

Here's the function prototype:

```
afx_msg BOOL OnZoom(UINT nID);
```

Other MFC message map macros are helpful for processing ranges of commands, as you might see in dynamic menu applications. These macros include:

```
ON_COMMAND_RANGE
ON_COMMAND_EX_RANGE
ON_UPDATE_COMMAND_UI_RANGE
```

If the values of `IDM_ZOOM_1` and `IDM_ZOOM_2` were consecutive, you could rewrite the `CMyView` message map as follows:

```
BEGIN_MESSAGE_MAP(CMyView, CView)
    ON_COMMAND_EX_RANGE(IDM_ZOOM_1, IDM_ZOOM_2, OnZoom)
END_MESSAGE_MAP()
```

Now `OnZoom()` is called for both menu choices, and the handler can determine the choice from the integer parameter.

### 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.