Lab 6.5: Adding a Property Sheet to an Application

Objectives

After completing this lab, you will be able to:

- Use the Component Gallery to create a property sheet.
- Use the dialog editor to build pages for the property sheet.
- Link pages together using CPropertyPage and CPropertySheet
- Update an application and its registry from a property sheet.

Prerequisites

This lab assumes competence in the basic use of the dialog resource editor.

Lab Setup

To see the solution to this lab, click this icon.

Estimated time to complete this lab: **30 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Creating a Property Sheet Template

In this exercise, you will use the property sheet component, a tool in the Component Gallery, to create a property sheet.

Exercise 2: Implementing the Property Sheet Classes

In this exercise, you will use DDV and DDX to set the member variables for the property sheet class for later retrieval in the calling function.

Exercise 3: Integrating the Property Sheet with an Application and the Registry

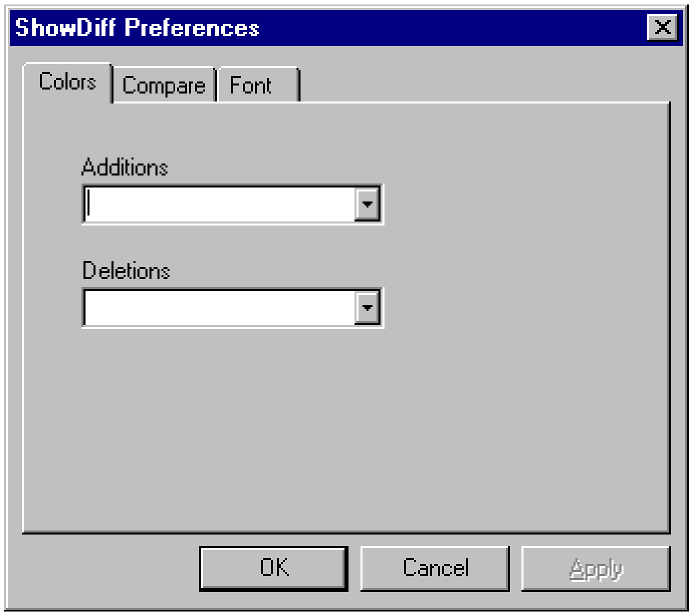
In this exercise, you will integrate the property sheet into an application, read and store data in the Registry, and use the property sheet to display and modify that data.

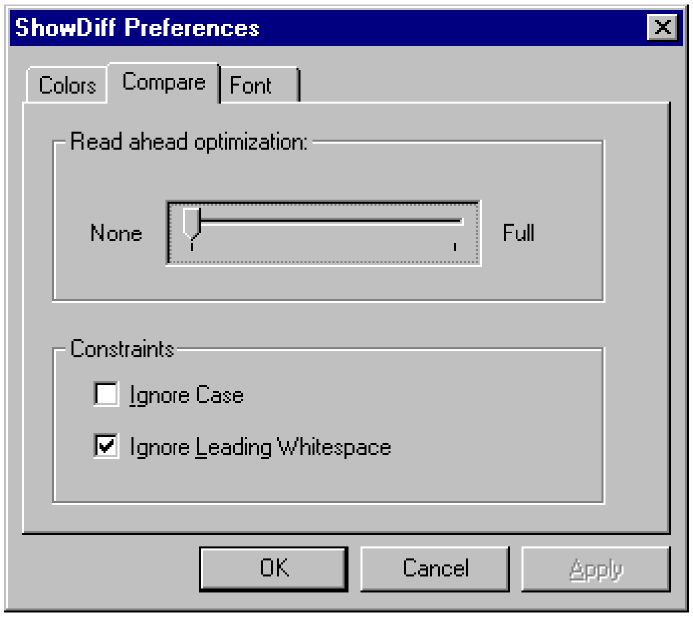
For this lab, you will need to use the project that you created in Lab 2: Adding a Modeless Dialog Box. You can copy the appropriate code from \Labs\C06\Lab05\Baseline. The completed code for the following exercises is in \Labs\C06\Lab05\Xxx, where Xxx is the exercise number.

Exercise 1: Creating the Property Sheet Template

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab05\Baseline.

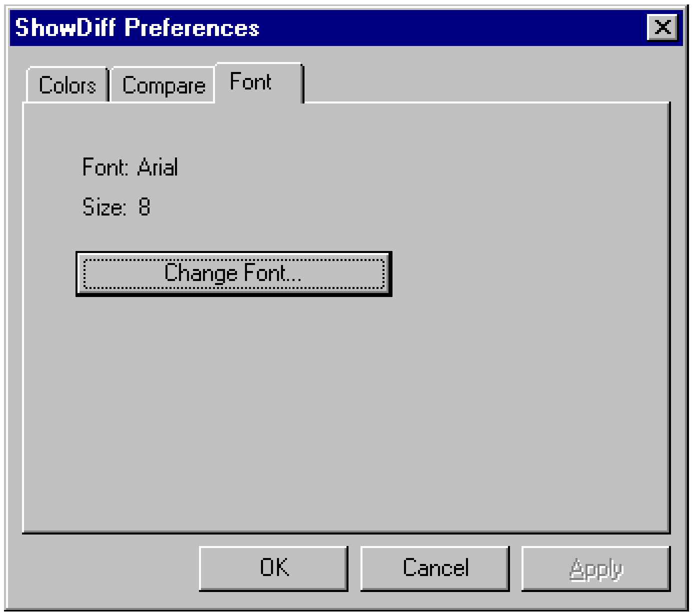
The Microsoft Foundation Class Library (MFC) provides property sheets and their pages with classes, **CPropertySheet** and **CPropertyPage**. The Developer Studio Gallery provides a wizard component that automates much of the effort of creating property sheets. This exercise will use this tool from the Developer Studio Gallery to create the property sheet for the ShowDiff application.





☐ Ignore Case

☑ Ignore Leading Whitespace

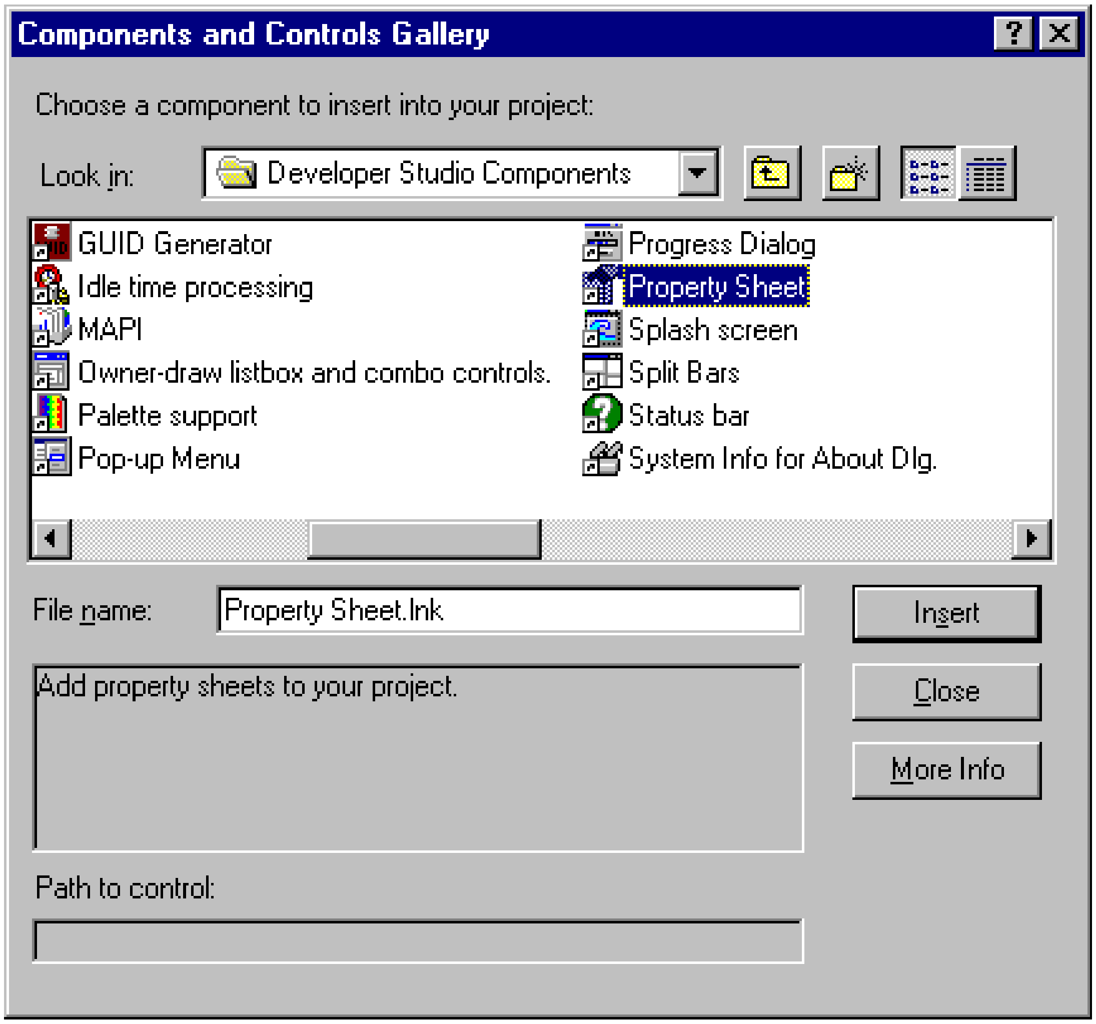


Font: Arial

Size: 8

➢ **Create the framework for the property sheet**

1. From the Project menu, choose Add to Project, and then choose Components and Controls. In the Gallery dialog box, choose Developer Studio Components.


Components and Controls Gallery
Choose a component to insert into your project:
Look in:
Developer Studio Components
GUID Generator
Idle time processing
MAPI
Owner-draw listbox and combo controls.
Palette support
Pop-up Menu
Progress Dialog
Property Sheet
Splash screen
Split Bars
Status bar
System Info for About Dlg.
File name:
Property Sheet.lnk
Insert
Close
More Info
Add property sheets to your project.
Path to control:

2. Select the Property Sheet component, choose Insert and click OK. You will then be asked whether you want to create a property sheet or a wizard. Choose property sheet and click Next.
3. In the next page, do not support previewing, and select No to leave the property sheet as modal.
4. You must then choose the class that will access this property sheet. Choose **CMainFrame**, which is where the main menu resides.
5. Choose three pages.
6. In the next page, you will name the classes for the property sheet and its pages. Use the names in this table.

Object	Class	Filenames
Property Sheet	**CPreferences**	Prefer.H and Prefer.Cpp
Page 1	**CColorPage**	Pages.H and Pages.Cpp
Page 2	**CComparePage**	Pages.H and Pages.Cpp
Page 3	**CFontPage**	Pages.H and Pages.Cpp

7. Click the Finish button. The Property Sheet wizard will create Prefer.H, Prefer.Cpp, Pages.H., and Pages.Cpp. It also will create three placeholder dialog resources for the three pages of your property sheet.

➢ **Modify the dialog box templates**

If you need to review the dialog editor, see Lab 6.1: Modifying Resources and Adding Dialog Boxes.

1. From the Dialog Editor, select the IDD_PROPPAGE1 dialog. Delete the static text provided. Change the ID of the dialog, and create controls according to this table.

Control Type	ID	Caption
Dialog	IDD_PAGE_COLORS	Colors
Static text	IDC_STATIC	Foreground
Static text	IDC_STATIC	Background
Combo Box	IDC_COMBO_COLOR_FORE	
Combo Box	IDC_COMBO_COLOR_BACK	

2. Select the IDD_PROPPAGE2 dialog. Delete the static text provided. Change the ID of the dialog, and create controls according to this table.

Control Type	ID	Caption
Dialog	IDD_PAGE_COMPARE	Compare
Slider	IDC_TRACKBAR1	Trackbar
		Set: Tickmarks, Autoticks, Border
Group Box	IDC_STATIC	Read ahead optimization:
Static text	IDC_STATIC	None
Static text	IDC_STATIC	Full
Group Box	IDC_STATIC	Constraints
Check Box	IDC_CHECK_IGNORE_CASE	&Ignore Case
Check Box	IDC_CHECK_IGNORE_WHITE	Ignore &Leading Whitespace

3. Select the IDD_PROPPAGE3 dialog. Delete the static text provided. Change the ID of the dialog, and create controls according to this table.

Control Type	ID	Caption
Dialog	IDD_PAGE_FONT	Font
Static text	IDC_STATIC	Font:
Static text	IDC_STATIC	Size:
Pushbutton	IDC_BUTTON_FONT	Change Font...
Static text	IDC_STATIC_FONT	Font:
Static text	IDC_STATIC_SIZE	Size:

4. Save Diff.Rc.

➢ **Change the resource identifiers for the property pages**

1. Open Pages.H.
2. In each of the class definitions is a dialog data section. For **CColorPage**, this section is this code:

```
// Dialog Data
    //{{AFX_DATA(CColorPage)
    enum { IDD = IDD_PROPPAGE1 };
        // NOTE - ClassWizard will add data members here.
        // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_DATA
```

Ignore the warning and change IDD_PROPPAGE1 to IDD_PAGE_COLORS, the new identifier for this dialog box.

3. Repeat for the other two classes, changing the temporary identifiers to the new identifiers, IDD_PAGE_COMPARE and IDD_PAGE_FONT.
4. Save Pages.H.

➤ **Change the caption of the property sheet**

1. Open the string table resource.
2. IDS_PROPSHT_CAPTION is the caption string for the property sheet. Change its caption to ShowDiff Preferences.
3. Save Diff.Rc.

➤ **Add a menu item for the property sheet**

1. Open the IDR_MAINFRAME menu resource.
2. Add a separator at the end of the View Menu.
3. Add a new menu item, ID_VIEW_PREFERENCES with a caption &Preferences, to the end of the View menu.
4. Save Diff.Rc.

➤ **Activate the property sheet**

1. From the View menu, start ClassWizard.
2. In the **CMainFrame** class, create a command handler for ID_VIEW_PREFERENCES, OnViewPreferences.
3. Edit the code for OnViewPreferences.
4. Call the **CMainFrame::OnProperties** function that is provided by the Property Sheet wizard.
5. Save all files. The complete function follows.

```
void CMainFrame::OnViewPreferences()
{
    OnProperties();
}
```

The completed code for this exercise is in \Labs\C06\Lab05\Ex01.

Exercise 2: Implementing the Property Sheet Classes

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab05\Ex01.

You can think of each page of a property sheet as an individual dialog box. As with all dialog boxes, you can use dialog data validation (DDV) and dialog data exchange (DDX) to set member variables for later retrieval in the calling function.

In this lab, you will implement the **CColorPage** class; you may also choose to implement an owner-draw combo box class to display colors for this page.

➤ **Rename the member variables for pages in CPreferences**

1. Open Prefer.h.
2. Change the names of the member variables to more informative names.

```
CColorPage      m_ColorPage;
CComparePage    m_ComparePage;
CFontPage       m_FontPage;
```

3. Save Prefer.h. Open Prefer.cpp.

4. Change the calls to **CPropertySheet::AddPage** to use the new names, as shown in this code:
```
AddPage(&m_ColorPage);
AddPage(&m_ComparePage);
AddPage(&m_FontPage);
```
5. Save Prefer.cpp.

➤ **Implement CComparePage**

1. Using ClassWizard, add member variables for each of the user controls in this template.

Control ID	Type	Variable name
IDC_CHECK_IGNORE_CASE	BOOL	m_bIgnoreCase
IDC_CHECK_IGNORE_WHITE	BOOL	m_bIgnoreWhiteSpace
IDC_TRACKBAR1	CSliderCtrl	m_Slider

2. Add a protected member for the trackbar's position.
```
UINTm_uReadAheadOptLevel;
```
3. Add member functions for setting and getting member variables.
```
// Attributes
public:
    void SetIgnoreCase(BOOL bIgnore){ m_bIgnoreCase = bIgnore; }
    void SetIgnoreWhiteSpace(BOOL bIgnore)
                            { m_bIgnoreWhiteSpace = bIgnore; }
    void SetReadAheadOptLevel(UINT nLevel)
                            { m_uReadAheadOptLevel = nLevel; }

    BOOL GetIgnoreCase() const { return m_bIgnoreCase; }
    BOOL GetIgnoreWhiteSpace() const { return m_bIgnoreWhiteSpace; }
    UINT GetReadAheadOptLevel() const { return m_uReadAheadOptLevel; }
```
4. Using ClassWizard, add a handler for WM_INITDIALOG. In this handler, set the range of the slider from 0 to 4.
```
BOOL CComparePage::OnInitDialog()
{
    CPropertyPage::OnInitDialog();
    m_Slider.SetRange(0, 4);
    return TRUE;
}
```
5. In **CComparePage::DoDataExchange**, add a routine to set or get the position of the slider. **CDataExchange.m_bSaveAndValidate** is TRUE if data is being read from the controls in the dialog box, and FALSE if data is being written to the controls.
```
if (pDX->m_bSaveAndValidate)
{
    m_uReadAheadOptLevel = m_Slider.GetPos();
}
else
{
    m_Slider.SetPos(m_uReadAheadOptLevel);
}
```
6. Save Pages.h and Pages.cpp.

➤ **Implement CFontPage**

1. Using ClassWizard, add member variables for two of the static controls in this template.

Control ID	Type	Variable name
IDC_STATIC_FONT	**CString**	m_FontName
IDC_STATIC_SIZE	**CString**	m_FontSize

2. Declare a variable to hold the character format.

```
CHARFORMAT  m_CharacterFormat;
```

3. Declare functions to get and set the character format.

```
public:
    void SetCharacterFormat(CHARFORMAT& CharFormat);
    void GetCharacterFormat(CHARFORMAT& CharFormat) const;
```

4. Add the implementation for these functions in the file Pages.cpp.

```
void CFontPage::SetCharacterFormat(CHARFORMAT& CharFormat)
{
    m_CharacterFormat = CharFormat;
}

void CFontPage::GetCharacterFormat(CHARFORMAT& CharFormat) const
{
    CharFormat = m_CharacterFormat;
}
```

5. Using ClassWizard, add a handler for BN_CLICKED on IDC_BUTTON_FONT. Go to the code created.
6. Create a Font common dialog box.

```
CFontDialog dlg(m_CharacterFormat);
```

7. Show the dialog box.

```
if(dlg.DoModal() == IDOK)
```

8. Retrieve the character format into m_CharacterFormat.

```
dlg.GetCharFormat(m_CharacterFormat);
```

9. Update the display.

```
UpdateData(FALSE);
```

10. The complete function follows.

```
void CFontPage::OnButtonFont()
{
    CFontDialog dlg(m_CharacterFormat);

    if(dlg.DoModal() == IDOK)
    {
        dlg.GetCharFormat(m_CharacterFormat);
        UpdateData(FALSE);
    }
}
```

11. Add code to the beginning of **CFontPage::DoDataExchange** to convert m_CharacterFormat to the member variables, as shown in the following code.

```
    if (!pDX->m_bSaveAndValidate) //updating the controls
    {
        //  Update FaceName and Size CString variables that
        //  are assocated with statics on the page.
        m_FontName = m_CharacterFormat.szFaceName;
        wsprintf(m_FontSize.GetBufferSetLength(32), "%ld Points",
m_CharacterFormat.yHeight/20);
        m_FontSize.ReleaseBuffer();
    }
```

12. Save Pages.cpp and Pages.h.
13. Build and run Diff.exe.

The code for this completed exercise is in \Labs\C06\Lab05\Ex02.

Exercise 3: Integrating the Property Sheet with the Application and the Registry

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab05\Ex02.

Up to now, you have been able to display the property sheet and manipulate its controls, but it has no association to the ShowDiff application. In this exercise, you will read and store data in the Registry and use the property sheet to display and modify that data.

➤ **Declare and initialize new member variables and functions**

1. Open Mainfrm.h. Declare protected attribute variables for ignore case, ignore whitespace, and look ahead a level. This code illustrates this process.

```
protected:
    //  Preferences
    BOOL    m_bIgnoreCase;          // Ignore case
    BOOL    m_bIgnoreWhiteSpace;// Ignore leading white space
    UINT    m_uReadAheadOptLevel;
```

2. Add public access member functions for these variables.

```
void SetIgnoreCase(BOOL bIgnore){ m_bIgnoreCase = bIgnore; }
void SetIgnoreWhiteSpace(BOOL bIgnore)
                        { m_bIgnoreWhiteSpace = bIgnore; }
void SetReadAheadOptLevel(UINT uLevel)
                        { m_uReadAheadOptLevel = uLevel; }

BOOL GetIgnoreCase() const { return m_bIgnoreCase; }
BOOL GetIgnoreWhiteSpace() const
                        { return m_bIgnoreWhiteSpace; }
UINT  GetReadAheadOptLevel() const
                        { return m_uReadAheadOptLevel; }
```

3. Save Mainfrm.h. and open Mainfrm.cpp to the **CMainFrame** constructor. Initialize the new variables.

```
m_bIgnoreCase = FALSE;
m_bIgnoreWhiteSpace = FALSE;
m_uReadAheadOptLevel = 2;
```

4. Save Mainfrm.cpp.

➤ **Create helper members in CDiffView**

1. To implement the font preferences functionality, you will need to be able to get and set the current fonts in both of the panes of **CDiffView**. Define public member functions in Diffview.h to get and set the current font, using the CHARFORMAT structure, as shown in this code.

   ```
   void GetCharacterFormat(CHARFORMAT& CharFmt);
   void SetCharacterFormat(CHARFORMAT& CharFmt);
   ```

2. Use **CRichEditCtrl::GetDefaultCharFormat** and **CRichEditCtrl::SetDefaultCharFormat** to get and set the character format in Diffview.cpp.

   ```
   void CDiffView::GetCharacterFormat(CHARFORMAT& CharFmt)
   {
       GetRichEditCtrl().GetDefaultCharFormat(CharFmt);
   }

   void CDiffView::SetCharacterFormat(CHARFORMAT& CharFmt)
   {
       GetRichEditCtrl().SetDefaultCharFormat(CharFmt);
       m_CharacterFormat = CharFmt;
   }
   ```

3. Save Diffview.h and Diffview.cpp.

➢ **Implement the OnViewPreferences handler**

1. Move to **CMainFrame::OnViewPreferences** in Mainfrm.cpp. Comment out the call to the function **OnProperties**. Construct a **CPreferences** object. Note that as part of the constructor for **CPreferences** (in Prefs.cpp), the pages are constructed and added into the property sheet.

   ```
   CPreferences Preferences;
   ```

2. Initialize the member variables of **CComparePage**.

   ```
   Preferences.m_ComparePage.SetIgnoreCase(GetIgnoreCase());
   Preferences.m_ComparePage.SetIgnoreWhiteSpace
                                   (GetIgnoreWhiteSpace());
   Preferences.m_ComparePage.SetReadAheadOptLevel
                                   (GetReadAheadOptLevel());
   ```

3. To initialize the font page, find the character format of the left pane and use it to set the format for the font page, as shown in this code.

   ```
   CDiffView* pViewLeft = GetView (0);
   CHARFORMAT CharFmt;
   memset(&CharFmt, 0, sizeof(CharFmt));
   CharFmt.cbSize = sizeof(CharFmt);
   pViewLeft->GetCharacterFormat(CharFmt);
   Preferences.m_FontPage.SetCharacterFormat (CharFmt);
   ```

4. Display the property sheet as a modal dialog box.

   ```
   if (Preferences.DoModal() == IDOK)
   ```

5. If the user presses the OK button, update the member variables from the **CComparePage**, as shown in this code.

   ```
   SetIgnoreCase(Preferences.m_ComparePage.GetIgnoreCase());
   SetIgnoreWhiteSpace
       (Preferences.m_ComparePage.GetIgnoreWhiteSpace());
   SetReadAheadOptLevel
       (Preferences.m_ComparePage.GetReadAheadOptLevel());
   ```

6. Get a pointer to the right view and use the two view pointers to update their character formats.

```
CDiffView* pViewRight = GetView (1);
Preferences.m_FontPage.GetCharacterFormat (CharFmt);
pViewLeft->SetCharacterFormat(CharFmt);
pViewRight->SetCharacterFormat(CharFmt);
```

7. Call a function to write the values into the Registry.

```
WriteRegistryPreferences();
```

8. Save Mainfrm.cpp. The complete function follows.

```
void CMainFrame::OnViewPreferences()
{
    CPreferences Preferences;


    //Initialize pages
    //compare
    Preferences.m_ComparePage.SetIgnoreCase(GetIgnoreCase());
    Preferences.m_ComparePage.SetIgnoreWhiteSpace
                                (GetIgnoreWhiteSpace());
    Preferences.m_ComparePage.SetReadAheadOptLevel
                                (GetReadAheadOptLevel());

    //font page
    // Get the left view as a data source
    CDiffView* pViewLeft = GetView (0);
    CHARFORMAT CharFmt;
    memset(&CharFmt, 0, sizeof(CharFmt));
    CharFmt.cbSize = sizeof(CharFmt);
    pViewLeft->GetCharacterFormat(CharFmt);
    Preferences.m_FontPage.SetCharacterFormat (CharFmt);

    if (Preferences.DoModal() == IDOK)
    {

        //update Compare options
        SetIgnoreCase(Preferences.m_ComparePage.GetIgnoreCase());
        SetIgnoreWhiteSpace
            (Preferences.m_ComparePage.GetIgnoreWhiteSpace());
        SetReadAheadOptLevel
            (Preferences.m_ComparePage.GetReadAheadOptLevel());

        //update Fonts
        CDiffView* pViewRight = GetView (1);
        Preferences.m_FontPage.GetCharacterFormat (CharFmt);
        pViewLeft->SetCharacterFormat(CharFmt);
        pViewRight->SetCharacterFormat(CharFmt);

        WriteRegistryPreferences();
    }
}
```

➢ **Implement a function to write Registry variables**

1. Open the String Table resource and add the strings listed in this table.

ID	**String**

IDS_REG_SECTION	Settings
IDS_REG_IGNORECASE	IgnoreCase
IDS_REG_IGNOREWHITESPACE	IgnoreWhiteSpace
IDS_REG_READAHEAD	ReadAhead

2. **CWinApp** provides member functions that write to a private profile file or to the Registry. The first step in writing to the Registry is to establish the key using **CWinApp::SetRegistryKey**. Because **SetRegistryKey** is protected, add this code to **CDiffApp::InitInstance** in Diff.cpp:

```
CString Key;
Key.LoadString(AFX_IDS_APP_TITLE);
SetRegistryKey (Key);
```

3. In Mainfrm.h, declare a public member function to write to the Registry.

```
void WriteRegistryPreferences();
```

4. In Mainfrm.cpp, define the function.

```
void CMainFrame::WriteRegistryPreferences()
```

5. Get a pointer to the application so you can call the **CWinApp** registry functions.

```
CWinApp* pApp = AfxGetApp();
```

6. Set up a **CString** for the section key.

```
CString Section;
Section.LoadString(IDS_REG_SECTION);
```

7. Set up a buffer for the entry key.

```
CString Entry;
Entry.LoadString(IDS_REG_IGNORECASE);
```

8. The three values that you will write to the Registry are all integers. Use **CWinApp:WriteProfileInt**, as shown in this code.

```
pApp->WriteProfileInt (Section, Entry, GetIgnoreCase());

Entry.LoadString(IDS_REG_IGNOREWHITESPACE);
    pApp->WriteProfileInt (Section, Entry, GetIgnoreWhiteSpace());

Entry.LoadString(IDS_REG_READAHEAD);
pApp->WriteProfileInt (Section, Entry, GetReadAheadOptLevel());
```

9. As an exercise, you can explore the writing of the character format with **CWinApp::WriteProfileBinary**. Save all your files. The complete WriteRegistryPreferences function follows.

```
void CMainFrame::WriteRegistryPreferences()
{
    CWinApp* pApp = AfxGetApp();

    CString Section;
    Section.LoadString(IDS_REG_SECTION);

    CString Entry;
    Entry.LoadString(IDS_REG_IGNORECASE);
    pApp->WriteProfileInt (Section, Entry, GetIgnoreCase());

    Entry.LoadString(IDS_REG_IGNOREWHITESPACE);
    pApp->WriteProfileInt (Section, Entry, GetIgnoreWhiteSpace());

    Entry.LoadString(IDS_REG_READAHEAD);
    pApp->WriteProfileInt (Section, Entry, GetReadAheadOptLevel());
}
```

➢ **Implement a function to read Registry variables**

1. **CMainFrame::ReadRegistryPreferences** is a simple reverse of **CMainFrame::WriteRegistryPreferences**. The complete function follows.

   ```
   void CMainFrame::ReadRegistryPreferences()
   {
       CWinApp* pApp = AfxGetApp();

       CString Section;
       Section.LoadString(IDS_REG_SECTION);


       CString Entry;
       Entry.LoadString(IDS_REG_IGNORECASE);
       SetIgnoreCase(pApp->GetProfileInt (Section, Entry,0));

       Entry.LoadString(IDS_REG_IGNOREWHITESPACE);
       SetIgnoreWhiteSpace(pApp->GetProfileInt (Section, Entry, 0));

       Entry.LoadString(IDS_REG_READAHEAD);
       SetReadAheadOptLevel(pApp->GetProfileInt (Section, Entry, 0));
   }
   ```

2. As with **WriteRegistryPreferences**, you can implement reading character preferences with **CWinApp::GetProfileBinary**.
3. You should read in the preferences when you initialize the application. Add a call to the function **CMainFrame::ReadRegistryPreferences** at the end **CDiffApp::InitInstance**, but before the **return** statement.
4. Remove references to **CMainFrame::OnProperties**.
5. Build and run Diff.exe.

The code for this completed exercise is in \Labs\C06\Lab05\Ex03.