Lab 6.4: Customizing the Common Dialog Class

Objectives

After completing this lab, you will be able to:

- Create a template to expand the functionality of a common dialog box.
- Create a class that expands the functionality of a common dialog box.
- Respond to messages sent by a common dialog box.
- Explore some of the source code for MFC.
- Stream data into a RichEdit control.
- Incorporate an expanded common dialog box into an existing application.

Prerequisites

Familiarity with the topics covered in this chapter.

Lab Setup

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

To see a demonstration of the solution to this lab, click this icon.

Estimated time to complete this lab: **60 minutes**.

Exercises

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

Exercise 1: Creating a Customized File Open Template

In this exercise, you will create a child dialog box of a common (parent) dialog box; the child dialog will inherit all the controls of the parent dialog box.

Exercise 2: Creating and Customizing a Dialog Class

In this exercise, you will use the ClassWizard to create the dialog class and add custom controls and handlers for the newly created dialog class.

Exercise 3: Implementing the Handlers for a Dialog Class

In this exercise, you will implement the handlers created in Exercise 2 and their supporting functions in the dialog class.

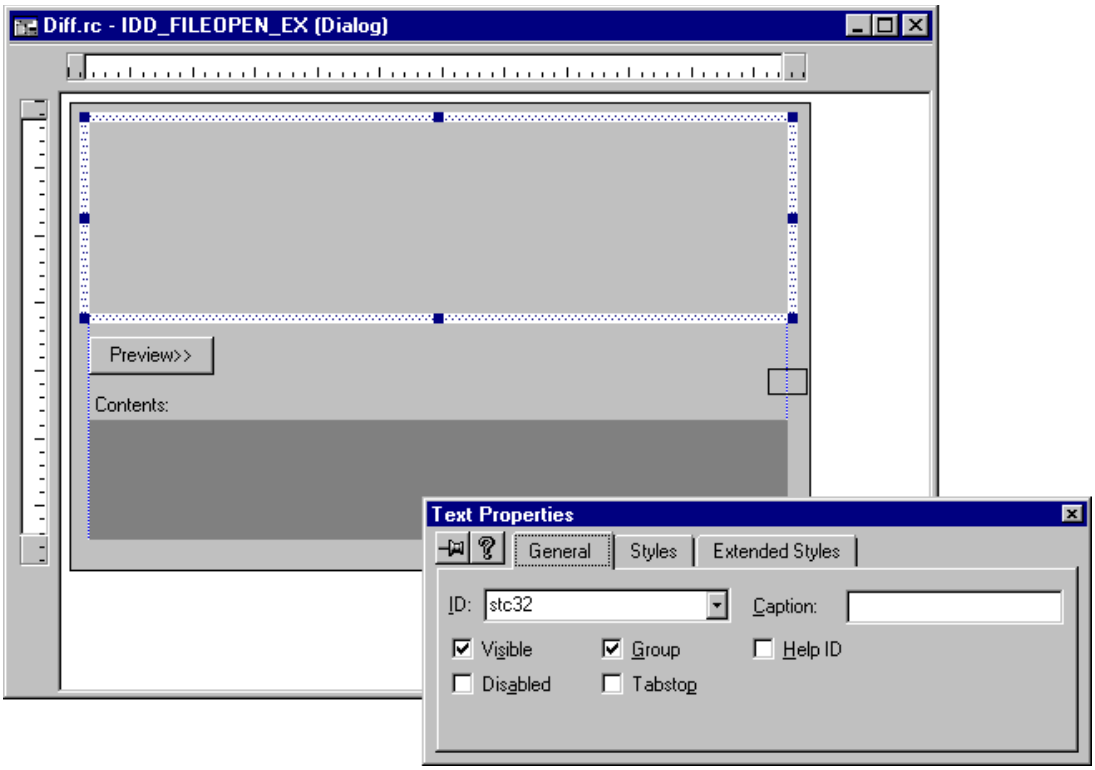
For this lab, you will need to use the project that you created in Chapter 5, Lab 4. You can copy this project from \Labs\C06\Lab04\Baseline. The completed code for these exercises is in \Labs\C06\Lab01\Xxx, where Xxx is the exercise number.

Exercise 1: Creating a Customized File Open 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\Lab04\Baseline.

In this exercise, you will create a child dialog box of a common dialog box, thus inheriting all the controls of the common dialog box. Even though your dialog box is a child of the common dialog box, you will lay out the child as if it were the parent, creating a special static control to “hold” the common dialog box. You also will need to tell Windows if you have controls that can be tabbed in your child dialog box.

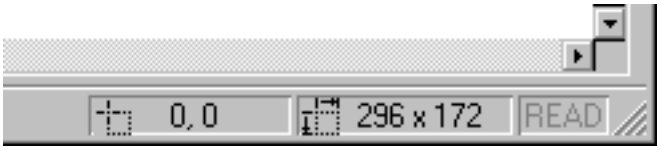
This graphic shows the completed template.



➢ **Create a new child dialog box of the common dialog box File Open**

1. From the Insert menu, choose Resource, or press CTRL+R.
2. From the Insert Resource dialog, choose Dialog and click OK.
3. Open the Dialog Properties dialog box, and set the ID to IDD_FILEOPEN_EX.
4. Choose the Styles tab; set the style to Child and the border to None. Clear the default Titlebar option if it is selected. Select Clip siblings and Clip children.
5. Choose the More Styles tab. Select Visible, 3D-look, and Control.

   Control (DS_CONTROL) allows the parent dialog (File Open) to tab into the child (IDD_FILEOPEN_EX).
6. Resize the dialog to 296 by 172. You can see the size in the right pane of the status bar, as this graphic shows.



7. Delete both the OK and Cancel buttons.
8. Draw a static control from the position 7,7 sized to 280 by 71. Give it the ID **stc32** so that Windows will draw the File dialog box within it. Clear its caption.
9. Save Diff.rc.
10. Open Resource.h and remove the definition of **stc32** added by the Dialog Editor.

    To open Resource.h or Diff.rc for direct editing, from the File menu, select open. Select the desired file, and from the Open As drop-down listbox, choose Text.
11. Save Resource. h.
12. The constant **stc32** is defined in Dlgs.h, so include Dlgs.h in Diff.rc.

➢ **Add an unfold push button and a label**

1. Open dialog IDD_FILEOPEN_EX.
2. Draw a push button from the position 7,86 sized to 50,14.
3. Open the properties dialog box. Set the button's ID to IDC_BUTTON_OPTIONS, and its caption to Preview>>.
4. Draw a static control from the position 9,107 sized to 31,8.
5. Set the caption of the static to Contents.

➢ **Save Diff.rc.**

1. Open dialog IDD_FILEOPEN_EX.
2. Choose the RichEdit control icon on the Control palette.
3. Draw a RichEdit control from the position 7,117 sized to 280,44.
4. Open the properties dialog. Set the ID of the RichEdit control to IDC_EDIT_CONTENTS.
5. Clear the Tabstop option.
6. Save Diff.rc.
7. Set the style of the RichEdit control by directly editing Diff.rc.

Since you want the user to scroll through the file, set the style to WS_VISIBLE | WS_BORDER | WS_VSCROLL | WS_HSCROLL.

➢ **Add size marking controls**

1. Open dialog IDD_FILEOPEN_EX.
2. Draw a picture control from the position 279,98 sized to 16,10.
3. Open the properties dialog box. Set the ID to IDC_FOLDSIZE and clear the Visible option.
4. Draw another picture control from the position 279,161, also sized 16,10.
5. Open the properties dialog box. Set the ID to IDC_UNFOLDSIZE and clear the Visible option.
6. Save Diff.rc.

Normally at this point, you would set the tabbing order and test the dialog box. Because only one control has WS_TABSTOP set, and because **FileOpenEx** is a child dialog box with a custom control, there is essentially nothing to test.

The code for this completed exercise is in \Labs\C06\Lab04\Ex01.

Exercise 2: Creating and Customizing a Dialog Class

Continue with the files you created in Exercise 1 or, if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab04\Ex01.

➢ **Use ClassWizard to Create the Dialog Class**

In this part of the exercise, you will invoke the ClassWizard to create the dialog class.

To have ClassWizard automatically present the Adding a Class dialog box, open ClassWizard from the dialog editor session where you just created the new dialog resource.

---

1. Run the dialog editor on the IDD_FILEOPEN_EX dialog resource.
2. Create a dialog class using ClassWizard.

   The new dialog class is created in a new pair of .Cpp and .H files, whose names are derived (by default) from the dialog class name.
3. In the dialog editor, be sure that your dialog box template window is the active child window.
4. Invoke ClassWizard. It should automatically display the Adding A Class dialog box. (If for some reason this dialog box does not appear, click the Add Class button.)
5. Select the Create a new class option, and click OK. The Create New Class dialog box appears.

6. In the Name box, type **CFileOpenEx** for the name of the associated C++ dialog class.
7. ClassWizard provides different styles of automation for dialog boxes derived from **CCommonDialog** and **CDialog**. To use ClassWizard to best effect, begin deriving from **CDialog,** and later derive from **CFileDialog**. Do this in ClassWizard and modify the class in the editor. In the Base Class box, select **CDialog**. The Dialog ID should already be set to IDD_FILEOPEN_EX.
8. Choose OK.
9. Briefly examine the source files for the CFileOpenEx class

➢ **Add Member Variables to the Dialog Class**

In this part of the exercise, you will create C++ member variables for the new **CFileOpenEx** class.

1. Open the newly created FileOpenEx.H.

   Because the RichEdit control is treated as a custom control, you will have to declare it manually rather than through ClassWizard.
2. Right-click and select CFileOpenEx in the class view window. Add the following protected member variable:

```
CRichEditCtrl   m_RichEdit
```

3. Add state variables to indicate whether the dialog box has been unfolded and whether it is currently unfolded. Add these variables to the protected implementation of the class:

```
BOOL    m_bFirstTime
BOOL    m_bUnfolded

Save FILEOPENEX.H
```

➢ **Create Handlers for the Dialog Class**

Using ClassWizard, add handlers for **DoModal**, **OnSize**, and **OnButtonOptions**. You will need to add the handler for **OnFileNameChanged** manually.

1. Invoke ClassWizard for the Diff project.
2. Move to the Message Maps tab in ClassWizard.
3. Create a handler for a dialog message.
4. In Class Wizard, go to the Message Maps property page. In the Class Name combo box, select the dialog class (**CFileOpenEx**).
5. Select the class name as the Object ID.
6. Select the message you want to handle in the Messages list box.
7. Choose the Add Function button.
8. The Add Member Function dialog box will be displayed. You can accept (the usual case) or change the Member Function name. Click OK to accept.
9. The function name will be displayed in the Member Functions list box.

➢ **Create a handler for a dialog control**

1. In Class Wizard, go to the Message Maps property page. In the Class Name combo box, select the dialog class (**CFileOpenEx**).
2. Select the control you want to handle in the ObjectIDs list box.
3. Select the message you want to handle in the Messages list box.
4. Choose the Add Function button.
5. The Add Member Function dialog box will be displayed. You may accept (the usual case) or change the Member Function name. Click OK to accept.
6. The function name will be displayed in the Member Functions list box.

➢ **Create a handler for a function not listed in ClassWizard**

When the selected file is changed in the Explorer-style list in the File dialog box, a **CDN_SELCHANGE** message is sent to the dialog itself. (For more information about this message, see “Open and Save As Dialog Boxes” in Win32SDK.) You will need to examine the source code for the **CFileDialog** class to determine how this message is handled.

1. In DlgFile.Cpp in the MFC source code, examine the **CFileDialog::OnNotify** handler.

```
BOOL CFileDialog::OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult)
{
    ...
    switch(pNotify->hdr.code)
    {
    ...
    case CDN_SELCHANGE:
        OnFileNameChange();
        return TRUE;
    ...
}
```

2. You can override **OnFileNameChange** in your derived class to process this message.
3. Add the **OnFileNameChange** to this dialog class as a protected function.

```
virtual void OnFileNameChange()
```

The completed code for this exercise is in \Labs\C06\Lab04\Ex02.

Exercise 3: Implementing the Handlers for a Dialog Class

Begin your implementation of **CDialogFileEx** by continuing Exercise 2 or, if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab04\Ex02.

CDialogFileEx has four primary entry points:

1. DoModal
2. OnButtonOptions
3. OnSize
4. OnFileNameChange

In this exercise, you will implement each of these handlers and their supporting functions.

➢ **Change the derivation of CFileOpenEx to CFileDialog**

1. Open FileOpenEx.h.
2. Change the parent class to **CFileDialog**.

```
class CFileIOpenEx : public CFileDialog
```

3. Save FileOpenEx.h.
4. Open FileOpenEx.cpp and scroll to the **CFileOpenEx** constructor. Convert the default constructor code for **CFileDialog** from:

```
CFileOpenEx::CFileOpenEx(CWnd* pParent /*=NULL*/)
    : CDialog(CFileOpenEx::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFileOpenEx)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}
```

To a constructor that looks like:

```
CFileOpenEx::CFileOpenEx(   LPCTSTR lpszDefExt /*= NULL*/,
                            LPCTSTR lpszFileName /*= NULL*/,
                            DWORD dwFlags /*= OFN_HIDEREADONLY*/,
                            LPCTSTR lpszFilter /*= NULL*/,
                            CWnd* pParentWnd /*= NULL*/)
            : CFileDialog(TRUE, lpszDefExt, lpszFileName, dwFlags,
lpszFilter, pParentWnd)
{

    //{{AFX_DATA_INIT(CFileOpenEx)
        // NOTE: ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}
```

6. While you are editing FileOpenEx.cpp, convert all occurrences of CDialog to CFileDialog.
7. Save FileOpenEx.cpp.
8. Open FileOpenEx.h.
9. Change the default constructor declaration from:

```
CFileOpenEx(CWnd* pParent = NULL);   // standard constructor
```

   To reflect the new constructor declaration:

```
CFileOpenEx(LPCTSTR lpszDefExt = NULL,
            LPCTSTR lpszFileName = NULL,
            DWORD dwFlags = OFN_HIDEREADONLY,
            LPCTSTR lpszFilter = NULL,
            CWnd* pParentWnd = NULL);
```

10. Save FileOpenEx.h.

➢ **Initialize preview panel members in the constructor**

The difference between **CFileDialog** and **CFileOpenEx** rests in the display of the preview panel. Because the preview panel does not exist until **DoModal** is called, the constructor needs only to mark that the dialog box is not showing the preview. You will need to fold the dialog box at the first display.

1. The flags for the preview panel are m_bUnfolded and m_bFirstTime. Set these flags in the constructor.

```
m_bUnfolded = TRUE;
m_bFirstTime = TRUE;
```

2. Save DlgFlex.cpp.

The complete constructor is shown in this code sample.

```
CFileOpenEx::CFileOpenEx(   LPCTSTR lpszDefExt /*= NULL*/,
                            LPCTSTR lpszFileName /*= NULL*/,
                            DWORD dwFlags /*= OFN_HIDEREADONLY*/,
                            LPCTSTR lpszFilter /*= NULL*/,
                            CWnd* pParentWnd /*= NULL*/)
            : CFileDialog(TRUE, lpszDefExt, lpszFileName, dwFlags,
lpszFilter, pParentWnd)
{
        //
        //  Always start out folded
        //
    m_bUnfolded = TRUE;
    m_bFirstTime = TRUE;
```

```
    //{{AFX_DATA_INIT(CFileOpenEx)
        // NOTE: ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}
```

➢ **Set up DDX**

Since a RichEdit control is a custom control from the Resource editors, you will need to set up Data Exchange (DDX) yourself.

1. Add a call to **DDX_Control** to **DoDataExchange**.

```
DDX_Control(pDX, IDC_EDIT_CONTENTS, m_RichEdit);
```

2. Save DlgFlex.cpp. The complete function is shown in this code sample.

```
void CFileOpenEx::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFileIOpenEx)
    //}}AFX_DATA_MAP
        DDX_Control(pDX, IDC_EDIT_CONTENTS, m_RichEdit);
}
```

➢ **Link the dialog box to the template in DoModal**

DoModal associates the template you have created with the **CFileDialog** common dialog box and then calls **CFileDialog::DoModal**. The **CFileDialog** class has an OPENFILENAME structure as one of its members (m_ofn).

1. You can associate IDD_FILEOPEN_EX with an instance of **CFileOpenEx** by directly setting the lpTemplateName member:

```
m_ofn.lpTemplateName = MAKEINTRESOURCE(IDD_FILEOPEN_EX);
```

However, If you investigate the source of MFC, you will find a member function, **SetTemplate**, that abstracts this functionality.

2. ClassWizard provides an **enum** IDD that isolates your code from changes you may make in your dialog resource. Use this variable when you call **SetTemplate**:

```
SetTemplate(NULL, MAKEINTRESOURCE(IDD));
```

3. Save FileOpenEx.cpp. The complete function is shown in this code.

```
int CFileOpenEx::DoModal()
{
    SetTemplate(NULL, MAKEINTRESOURCE(IDD));

    return CFileDialog::DoModal();

}
```

➢ **Link the Preview button to the Expand function**

The Preview button folds or unfolds the preview. Because all the work is done in **Expand**, all you need to do is call **Expand** with the logical inversion of the current state of the preview.

1. In **OnButtonOptions**, call **Expand**.

```
Expand(!m_bUnfolded);
```

2. Save FileOpenEx.cpp. The complete function is shown in this code.

```
void CFileOpenEx::OnButtonOptions()
{
```

```
    Expand(!m_bUnfolded);
}
```

➢ **Fold the dialog the first time using OnSize**

OnSize is called right after the File dialog window is created, but before the dialog is made visible. This is an ideal time to size the dialog box to hide the preview. **OnSize** also is called after each time you resize the dialog, and you will want to forego additional processing at this point. Check m_bFirstTime and collapse the dialog if this is the first time through.

1. After calling the default function, check to see whether this is the first time **OnSize** has been called.
```
if(m_bFirstTime)
```

2. If it is, set m_bFirstTime to FALSE.
```
m_bFirstTime = FALSE;
```

3. Set the dialog to not expanded.
```
Expand(FALSE);
```

4. Save FileOpenEx.cpp. The complete function is shown in this code.
```
void CFileOpenEx::OnSize(UINT nType, int cx, int cy)
{
    CFileDialog::OnSize(nType, cx, cy);

    if(m_bFirstTime)
    {
        m_bFirstTime = FALSE;
        Expand(FALSE);
    }
}
```

➢ **Implement OnFileNameChange**

When the preview is unfolded, you will want to display a file when the user selects it in the pane. You will need to ensure that the user has selected a filename, as the corresponding message also is sent when the user deselects all files. You also need to validate that the filename is valid, as the filename is combined of the currently selected directory and the file in the filename edit control.

1. Check to see whether the dialog box is unfolded to show the preview.
```
if(m_bUnfolded)
{
```

2. Get the path name from the File dialog box, and check to see that it is not blank.
```
CString FileName = GetPathName();
if(FileName.GetLength())
{
```

3. Check to see whether the file exists.
```
OFSTRUCTof;
if(OpenFile(FileName, &of, OF_EXIST) != HFILE_ERROR)
```

4. You will implement **PreviewContents** to show the file in the RichEdit control later in this exercise. At this point, simply call **PreviewContents**.
```
PreviewContents(FileName);
```

5. Save FileOpenEx.cpp. The complete function is shown in this code sample.

```
void CFileOpenEx::OnFileNameChange()
{
    if(m_bUnfolded)
    {
        CString FileName = GetPathName();
        if(FileName.GetLength())
        {
            OFSTRUCTof;
            if(OpenFile(FileName, &of, OF_EXIST) != HFILE_ERROR)
            {
                PreviewContents(FileName);
            }
        }
    }
}
```

➢ Implement the Expand function

1. Build the Expand function to show or hide the preview

   Expand resizes the dialog box to show or hide the preview RichEdit control. Determine the size of the dialog box based on the parameter passed, and the positions of IDC_FOLDSIZE and IDC_UNFOLDSIZE. Resize the dialog box. Lastly, call **EnableOptionalChildren** to enable the RichEdit control.

2. Add the Expand function to the CFileOpenEx class as a protected member function

```
void    Expand(const BOOL bExpand = TRUE)
```

➢ Size the dialog box

1. Validate that it is necessary to resize the dialog box. Check that the input parameter is different from current state.

```
if( (bExpand && !m_bUnfolded) ||
    (!bExpand && m_bUnfolded) )
```

2. Find the new size. Before you change the dialog size, get the rectangle of the dialog box as it is, because you may need it later. Also declare additional CRects for the function's use.

```
CRect   rectInitDlg, rectNewDlg, rectBasic;
GetParent()->GetWindowRect(&rectInitDlg);
```

3. Determine which control marks the extent of the dialog box.

```
WORD wSizeCtrl = IDC_FOLDSIZE;
if(bExpand)
{
    wSizeCtrl = IDC_UNFOLDSIZE;
}
```

4. Get the control so that you can get its properties.

```
CWnd* pSizeMarker = GetDlgItem(wSizeCtrl);
```

5. Get the rectangle of the control and convert it to client-relative coordinates.

```
CRect rectMarker;
pSizeMarker->GetWindowRect(&rectMarker);
ScreenToClient(&rectMarker);
```

6. Set the new rectangle.

```
rectNewDlg.SetRect(0, 0, rectMarker.right, rectMarker.bottom);
```

7. Convert it back to screen coordinates.

```
::AdjustWindowRectEx(    &rectNewDlg,
                         GetParent()->GetStyle(),
                         FALSE,
                         GetParent()->GetExStyle());
```

➢ **Set the new size**

- Set the position of the parent dialog to the calculated coordinates.

```
GetParent()->SetWindowPos(  NULL,
                         0, 0,
                         rectNewDlg.Width(), rectNewDlg.Height(),
                         SWP_NOACTIVATE | SWP_NOMOVE | SWP_NOZORDER);
```

➢ **Enable or disable controls in the child window**

1. Determine the folded rectangle.

```
rectBasic = bExpand?rectInitDlg:rectNewDlg;
ClientToScreen(&rectBasic);
```

2. Call **EnableOptionalChildren** to enable or disable children of the dialog that are outside the folded rectangle.

```
EnableOptionalChildren(rectBasic, bExpand);
```

3. Set the folded flag to the current display state.

```
m_bUnfolded = bExpand;
```

4. Save FileOpenEx.cpp. The complete function is shown in this code sample.

```
void CFileOpenEx::Expand(const BOOL bExpand)
{
    if( (bExpand && !m_bUnfolded) ||
        (!bExpand && m_bUnfolded) )
    {
        CRect   rectInitDlg, rectNewDlg, rectBasic;
        GetParent()->GetWindowRect(&rectInitDlg);

        WORD wSizeCtrl = IDC_FOLDSIZE;
        if(bExpand)
        {
            wSizeCtrl = IDC_UNFOLDSIZE;
        }

        CWnd* pSizeMarker = GetDlgItem(wSizeCtrl);

        CRect rectMarker;
        pSizeMarker->GetWindowRect(&rectMarker);
        ScreenToClient(&rectMarker);

        rectNewDlg.SetRect(0, 0, rectMarker.right, rectMarker.bottom);

        ::AdjustWindowRectEx(   &rectNewDlg,
                                GetParent()->GetStyle(),
                                FALSE,
                                GetParent()->GetExStyle());

        GetParent()->SetWindowPos(  NULL,
                                    0, 0,
```

```
                                        rectNewDlg.Width(),
rectNewDlg.Height(),
                                        SWP_NOACTIVATE | SWP_NOMOVE |
SWP_NOZORDER);

        rectBasic = bExpand?rectInitDlg:rectNewDlg;
        ClientToScreen(&rectBasic);

        EnableOptionalChildren(rectBasic, bExpand);

        m_bUnfolded = bExpand;

    }
}
```

- **Implement a function to sequentially visit all the children of the dialog, and enable or disable as appropriate**

  EnableOptionalChildren sequentially goes through the child windows of the **CFileOpenEx** dialog box. If the current window in the walk is outside the static part of the dialog box, it will be enabled or disabled as appropriate.

1. **Add** EnableOptionalChildren **to the CFileOpenEx class as a protected member function.**

```
void EnableOptionalChildren ( const CRect& rectBasic,
                              const BOOL bEnable = TRUE )
```

2. Declare a **CRect** to hold the rectangle of the current window.

```
CWnd *pChild = GetTopWindow();
```

3. Get the first window in the child list.

```
CWnd *pChild = GetTopWindow();
```

4. Loop while you have a child window and get its rectangle.

```
while (pChild)
{
    pChild->GetWindowRect(&rectChild);
```

5. Check to see whether the top-left corner or bottom-right corners are outside rectBasic, and if so, enable or disable as appropriate.

```
if(     !rectBasic.PtInRect(rectChild.TopLeft())
    ||  !rectBasic.PtInRect(rectChild.BottomRight()) )
{
    pChild->EnableWindow(bEnable);
}
```

6. Get the next window in the list.

```
pChild = pChild->GetNextWindow();
```

7. Close the function and save FileOpenEx.cpp. The complete function is shown in this code sample.

```
void CFileOpenEx::EnableOptionalChildren (const CRect& rectBasic,
                                          const BOOL   bEnable)
{
    CRect rectChild;

    CWnd *pChild = GetTopWindow();

    while (pChild)
```

```
    {
        pChild->GetWindowRect(&rectChild);

        if(     !rectBasic.PtInRect(rectChild.TopLeft())
            ||  !rectBasic.PtInRect(rectChild.BottomRight()) )
        {
            pChild->EnableWindow(bEnable);
        }
        pChild = pChild->GetNextWindow();
    }
}
```

➢ **Implement a function to load the file into the preview**

CRichEditCtrl has a stream-oriented input and output provided by **StreamIn** and **StreamOut**. You will be using **StreamIn** to load the preview from the selected file. **StreamIn** uses a callback function to do the actual file processing, and you will specify that function as part of an EDITSTREAM structure.

1. **Add** PreviewContents **to the CFileOpenEx class as a protected member function.**

```
void PreviewContents(LPCSTR lpszFilespec)
```

2. Open the file specified in lpszFilespec.

```
CFile File(lpszFilespec, CFile::modeRead);
```

3. Declare an EDITSTREAM structure. Set the cookie to CFile, clear the error return, and set the callback to OpenCallback.

```
EDITSTREAM      es;

es.dwCookie = (DWORD)&File;
es.dwError      = 0;
es.pfnCallback  = OpenCallback;
```

4. Send the StreamIn message, specifying that the data is to be read as plain text.

```
m_RichEdit.StreamIn(SF_TEXT, es);
```

5. Save FileOpenEx.cpp. The complete function is shown in this code sample.

```
void CFileOpenEx::PreviewContents(LPCSTR lpszFilespec)
{
    CFile File(lpszFilespec, CFile::modeRead);

    EDITSTREAM      es;
    es.dwCookie = (DWORD)&File;
    es.dwError      = 0;
    es.pfnCallback  = OpenCallback;

    m_RichEdit.StreamIn(SF_TEXT, es);
}
```

➢ **Implement the callback function to read the file (OpenCallback)**

The callback specified in the EDITSTREAM structure does the work of reading the data stream into a buffer that **CRichEditCtrl** can process.

1. Before your implementation of **PreviewContents**, declare **OpenCallback** in FileOpenEx.h. An EDITSTREAM callback takes four parameters: the DWORD sent by **StreamIn**, a buffer to receive the stream, the size of the buffer, and a buffer to receive the actual count of bytes read:

```
DWORD CALLBACK OpenCallback(DWORD   dwCookie,
                            LPBYTE  pbBuff,
                            LONGcb,
                            LONG*pcb )
```

```
{
```

2. Cast dwCookie to a CFile pointer.

```
CFile   *pFile = (CFile *)dwCookie;
```

3. Read cb bytes into pbBuff.

```
*pcb = pFile->Read(pbBuff, cb);
```

4. And return that you have processed the call.

```
return 0;
```

5. Save FileOpenEx.cpp. The complete function is shown in this code.

```
DWORD CALLBACK OpenCallback(DWORD   dwCookie,
                            LPBYTE  pbBuff,
                            LONGcb,
                            LONG*pcb )
{
    CFile   *pFile = (CFIle *)dwCookie;

    *pcb = pFile->Read(pbBuff, cb);
    return 0;
}
```

➢ **Enable CDialogFileEx**

One of the advantages of maintaining a good object-oriented development style is the ease of adding, removing, or changing components. This is demonstrated in the ease of incorporating the **CFileOpenEx** class into the ShowDIFF application. You need only include the class header and change two constructor calls to enable **CFileOpenEx**.

1. Open FileOpenEx.cpp.
2. Include the **CDialogFileEx** header.

```
#include "fileopenex.h"
```

3. In **OnButtonFile1Browse** replace the **CFileDialog** declarations with **CFileOpenEx**.

```
CFileDialog dlg(TRUE);
//becomes =>
CFileOpenEx dlg;
```

4. Repeat for **OnButtonFile2Browse**.
5. Save FileOpenEx.cpp and build the project.

The code for this completed exercise is in \Labs\C06\Lab04\Ex03.