Lab 12.1: Building an ActiveX Control from an Existing Class

Objectives

After completing this lab, you will be able to:

- Package an existing class as an ActiveX control.
- Use the ActiveX Control Test Container.
- Add property pages to an ActiveX control.

Prerequisites

Familiarity with the topics covered in this chapter.

Lab Setup

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

Estimated time to complete this lab: **40 minutes**.

Exercises

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

Exercise 1: Building an ActiveX Control

In this exercise, you will create an ActiveX control by encapsulating the existing Device-Independent Bitmap (DIB) class. After you create the control, you will test the control using the ActiveX Control Test Container.

Exercise 2: Setting Properties for an ActiveX Control

In this exercise, you will set the properties for the ActiveX control you created in Exercise 1.

Exercise 3: Adding a Property Sheet Interface

In this exercise, you will use a property sheet to create a design-mode interface for the ActiveX control you implemented in Exercise 2.

There is no setup for this lab. Copy the implementation of the DIB class from \Labs\C12\Lab01\Baseline. The completed code for these exercises is in \Labs\C12\Lab01\Xxx, where Xxx is the exercise number.

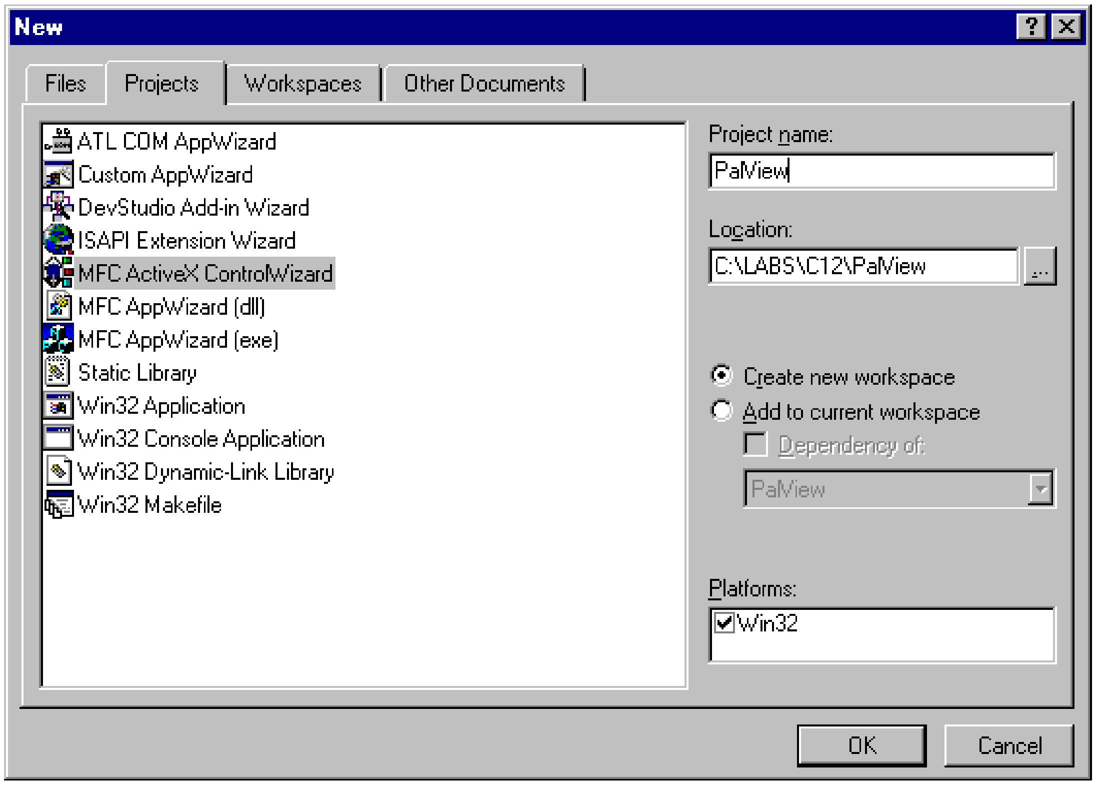
Exercise 1: Building an ActiveX Control

The code that forms the basis for this exercise is in \Labs\C12\Lab01\Baseline. Copy the files into your project directory.

In this exercise, you will encapsulate a DIB and palette-display class as an ActiveX control. The only property you will implement in this exercise is the file whose palette is to be displayed. You will then test the control using the ActiveX Control Test Container.

➢ **Build the framework control**

1. Start Microsoft Developer Studio.
2. From the File menu, choose New.
3. In the New dialog box, choose the Projects tab.
4. Choose the MFC ActiveX ControlWizard. Name the project PalView.



Project name: PalView

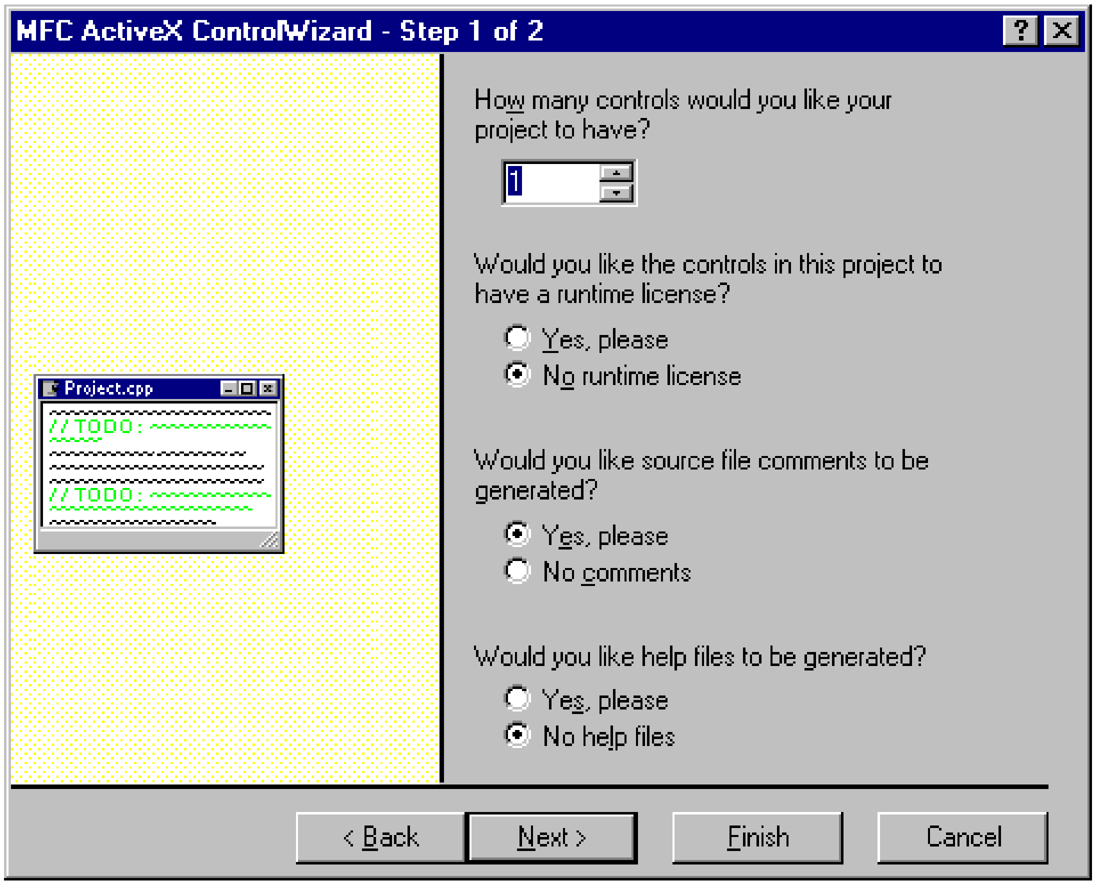
Location: C:\LABS\C12\PalView

[✓ Create new workspace] Add to current workspace

Dependency of: PalView

Platforms: ✓ Win32

5. Click OK to start the MFC ActiveX ControlWizard. In Step 1 of the ActiveX ControlWizard, create one control and leave the other options at their default settings.



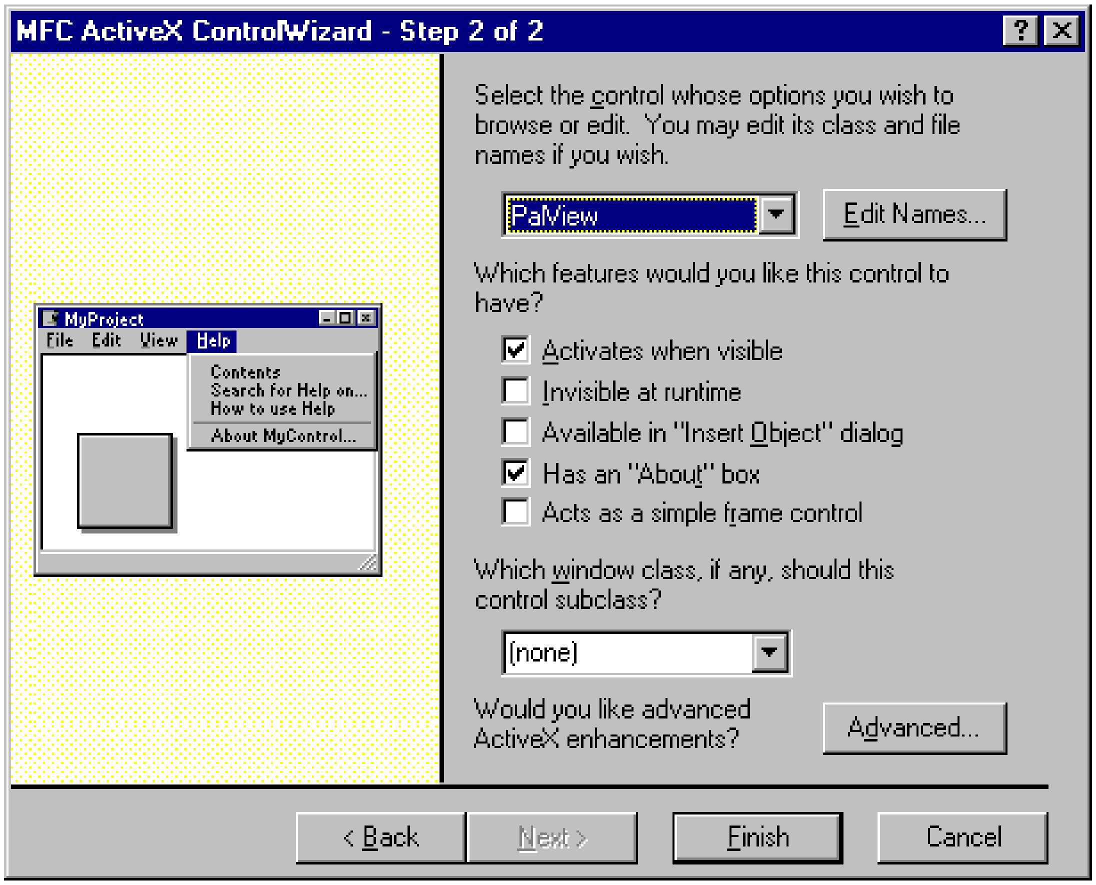
How many controls would you like your project to have? 1

Would you like the controls in this project to have a runtime license?
Yes, please [✓ No runtime license]

Would you like source file comments to be generated?
[✓ Yes, please] No comments

Would you like help files to be generated?
Yes, please [✓ No help files]

6. Click Next. Step 2 of the ActiveX ControlWizard displays. Accept the default settings for all options.



MFC ActiveX ControlWizard - Step 2 of 2

Select the control whose options you wish to browse or edit. You may edit its class and file names if you wish.

PalView

Which features would you like this control to have?

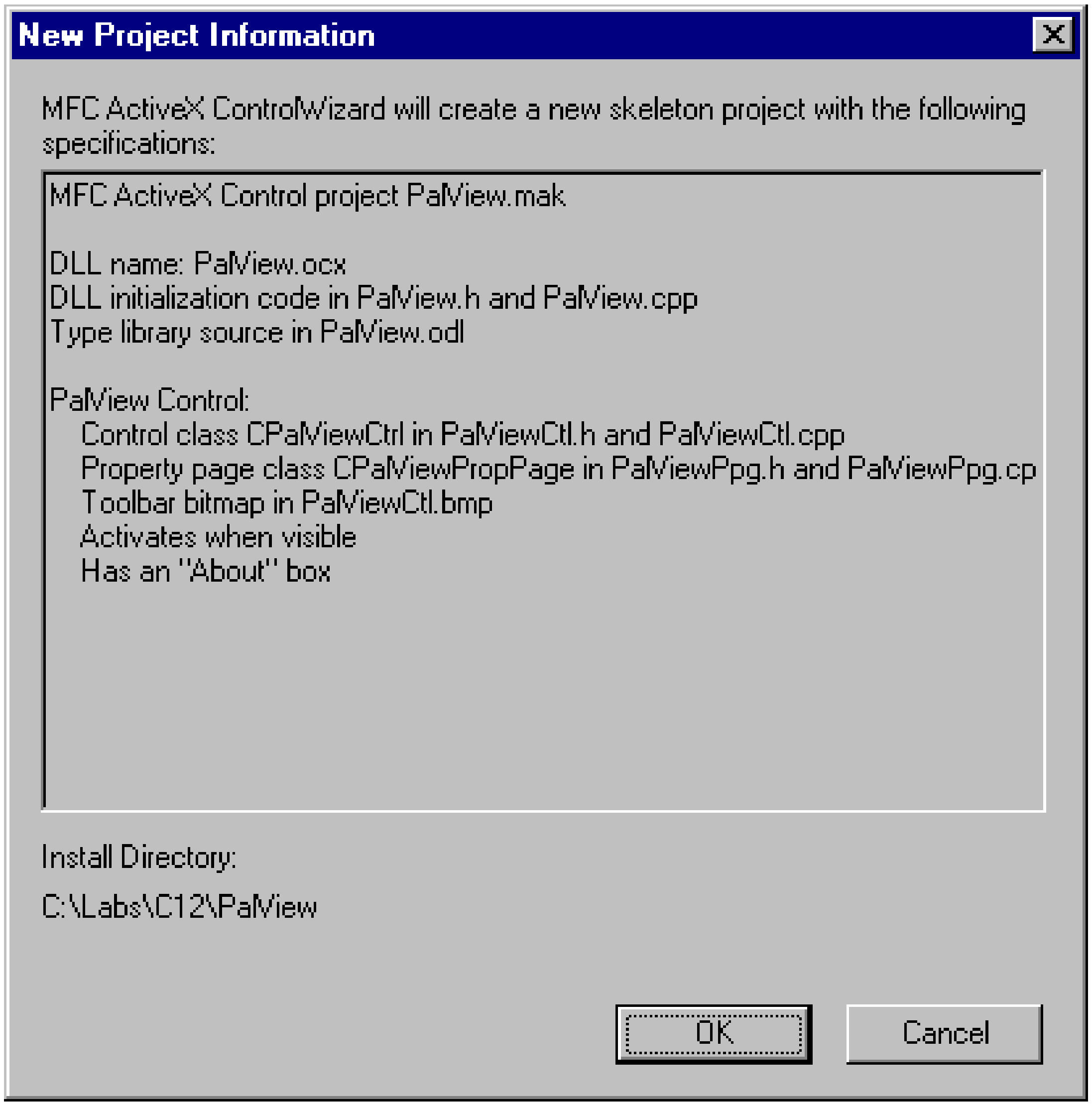
- [x] Activates when visible
- [ ] Invisible at runtime
- [ ] Available in "Insert Object" dialog
- [x] Has an "About" box
- [ ] Acts as a simple frame control

Which window class, if any, should this control subclass?

(none)

Would you like advanced ActiveX enhancements?

7. Click Finish. The New Project Information dialog box appears.



MFC ActiveX ControlWizard will create a new skeleton project with the following specifications:

```
MFC ActiveX Control project PalView.mak

DLL name: PalView.ocx
DLL initialization code in PalView.h and PalView.cpp
Type library source in PalView.odl

PalView Control:
    Control class CPalViewCtrl in PalViewCtl.h and PalViewCtl.cpp
    Property page class CPalViewPropPage in PalViewPpg.h and PalViewPpg.cp
    Toolbar bitmap in PalViewCtl.bmp
    Activates when visible
    Has an "About" box
```

Install Directory:

C:\Labs\C12\PalView

8. Click OK to create the project.
9. From the Project menu, choose Add to Project, and choose Files. Find and insert Dib.Cpp and DibPal.Cpp into the project. You do not need to insert their headers; Developer Studio will resolve their dependencies.

➢ **Integrate CDibPal into CPalViewCtrl**

1. Right-click **CPalViewCtrl** in ClassView and add a protected member variable.

   ```
   CDIBPal*m_pDibPal
   ```

2. Open PalViewCtl.Cpp.
3. Include Dib.H and DibPal.H before including PalViewCtl.H.
4. Update the constructor to initialize m_pDibPal to zero (0). The complete constructor follows.

   ```
   CPalViewCtrl::CPalViewCtrl()
   {
   InitializeIIDs(&IID_DPalView, &IID_DPalViewEvents);
   m_pDibPal = 0;
   }
   ```

5. Update the destructor to delete m_pDibPal if it exists. The complete destructor follows.

```
CPalViewCtrl::~CPalViewCtrl()
{
    if (0 != m_pDibPal)
    {
        delete m_pDibPal;
        m_pDibPal = 0;
    }
}
```

➢ **Update CPalViewCtrl::OnDraw to use CDibPal::Draw**

1. Modify **CPalViewCtrl::OnDraw** to use **CDIBPal::OnDraw** if **CDIBPal** has been instantiated.

```
if(0 != m_pDibPal)
{
    m_pDibPal->Draw(pdc, rcBounds, TRUE);
}
```

Otherwise, fill the control with a white brush.

```
else
    pdc->FillRect(rcBounds,
        CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
```

2. Save PalViewCtl.Cpp. The complete function follows.

```
void CPalViewCtrl::OnDraw(
CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    if(0 != m_pDibPal)
    {
        m_pDibPal->Draw(pdc, rcBounds, TRUE);
    }
    else
    {
        pdc->FillRect(rcBounds,
            CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
    }
}
```

➢ **Add a function to extract the palette from a DIB file**

1. Right-click CPalViewCtl in ClassView and add a protected member function.

```
void ExtractPalFromDIBFile(LPCTSTR lpszFileSpec)
```

2. Add these variables to the implementation of **ExtractPalFromDIBFile**.

```
CDIB    Dib;
CFile   DibFile;
```

3. Open the file whose path is in lpszFileSpec.

```
if(DibFile.Open(lpszFileSpec, CFile::modeRead))
```

4. Copy the file into a CDIB instance.

```
if(Dib.Load(&DibFile))
```

5. Create a palette and extract the color table from the DIB into the palette.

```
CDIBPal* pTemp = new CDIBPal;
if(pTemp->Create(&Dib))
```

6. If you already have a CDIBPal instance, delete it.

```
if(0 != m_pDibPal)
{
    delete m_pDibPal;
    m_pDibPal = 0;
}
```

7. Assign the palette you created to m_pDibPal.

```
m_pDibPal = pTemp;
```

8. Draw the palette in 3D style.

```
m_pDibPal->SetDraw3D(TRUE);
```

9. Save PalViewCtl.Cpp. The complete function follows.

```
void CPalViewCtrl::ExtractPalFromDIBFile(LPCSTR lpszFileSpec)
{
    CDIB    Dib;
    CFile   DibFile;

    if(DibFile.Open(lpszFileSpec, CFile::modeRead))
    {
        // Load the bitmap file into a CDIB object
        if(Dib.Load(&DibFile))
        {
            CDIBPal* pTemp = new CDIBPal;
            ASSERT(pTemp != NULL);
            //  This will extract the color table fron the CDIB object
            if(pTemp->Create(&Dib))
            {
                // Get rid of any existing CDIBPal object
                if(0 != m_pDibPal)
                {
                    delete m_pDibPal;
                    m_pDibPal = 0;
                }
                // Ready to go
                m_pDibPal = pTemp;

                // Make sure the new palette reflects our
                // current state
                m_pDibPal->SetDraw3D(TRUE);
            }
            else
            {
                delete pTemp;
            }
        }
    }
}
```

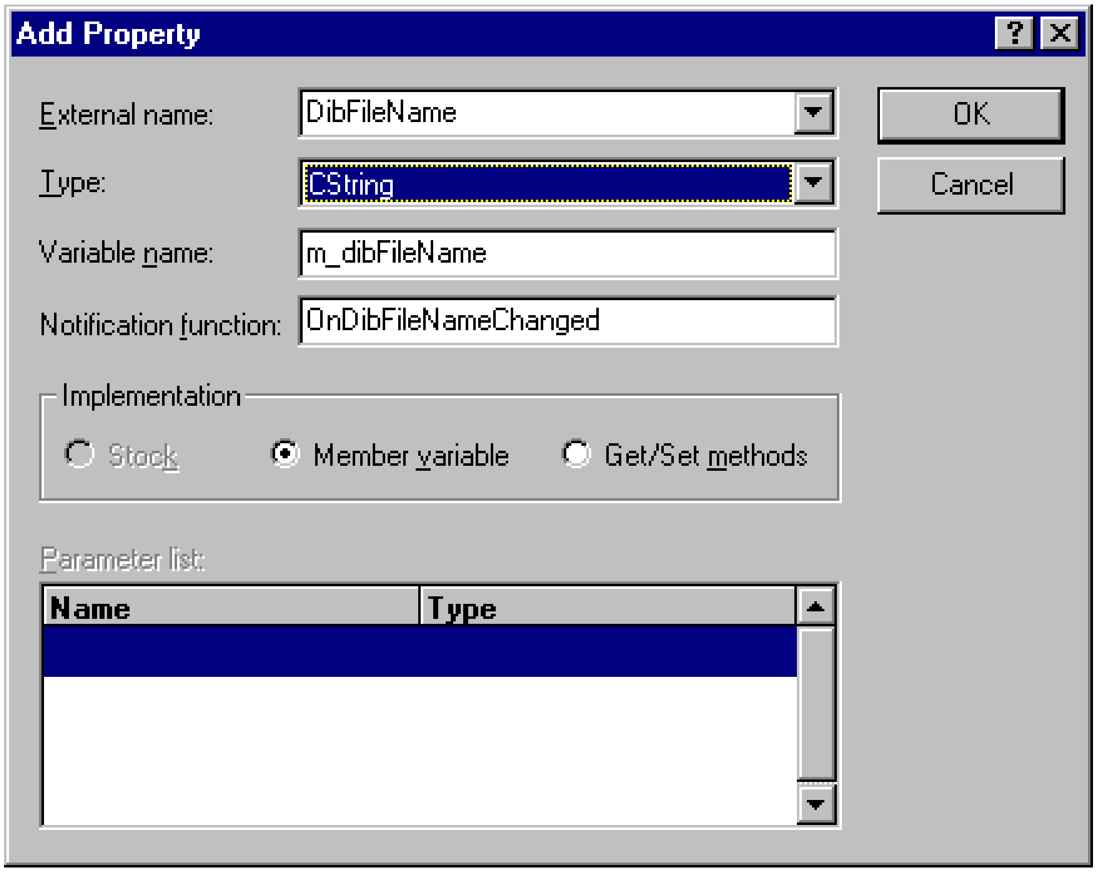
The palette view control has several externally changeable properties. The Microsoft Foundation Class Library provides two mechanisms for a container to change these properties:

- Member variables with notification.
- Get and Set methods.

Generally, use member variables when the property is a simple value stored in the control itself; use Get and Set methods when the property is a complex value that you want to interpret, or when the property is a member of an embedded class.

➢ **Add DibFileName property**

1. In the ClassWizard, choose the Automation page and click Add Property. The Add Property dialog box displays.



External name: DibFileName
Type: CString
Variable name: m_dibFileName
Notification function: OnDibFileNameChanged
Implementation: Stock [✓ Member variable] Get/Set methods

2. Create a new DibFileName property, using the following settings.

External Name	Type	Variable Name	Notification Function
DibFileName	CString	m_dibFileName	OnDibFileNameChanged

3. In PalViewCtl.Cpp, modify the new **OnDibFileNameChanged** function to extract the palette from **DibFileName** and repaint the control. The complete function follows.

```
void CPalViewCtrl::OnDibFileNameChanged()
{
ExtractPalFromDIBFile(m_dibFileName);
InvalidateControl();
SetModifiedFlag();
}
```

4. In the constructor, initialize m_dibFileName:

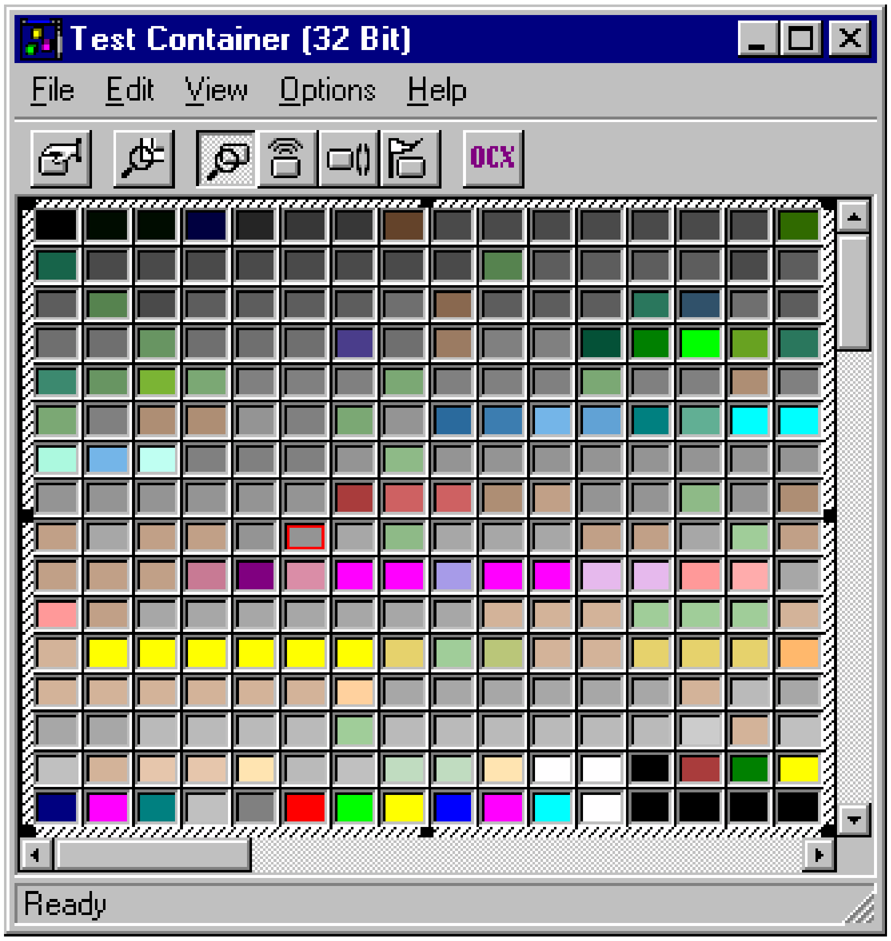
```
m_dibFileName = _T ("");
```

➢ **Build and run the control using the ActiveX Control Test Container**

1. Save all files and build PalView.Ocx.

If you have not copied the four files from the BASELINE directory to your project subdirectory, do so before building the project. Also, add these four files to the project by choosing Project/Add to Project/Files into Project.

2. Developer Studio provides a test container for ActiveX controls. You can run the test container by choosing the ActiveX Control Test Container option from the Tools menu (if you wish to simply test the control), or by choosing Execute from the Build menu. Identify the location of TstCon32.Exe. (It is in the \Program Files\DevStudio\VC\Bin directory in the default installation.) Developer Studio will inform you that there is no debug information in TstCon32. The ActiveX Control Test Container will then be displayed.



3. From the Edit menu, choose Insert OLE Control, then choose the PalView control. Resize the control to an appropriate size.
4. From the View menu, choose the Properties item. The Properties dialog displays. Because you have not implemented a Property Sheet for this control, invoking the Properties verb displays only the blank default property sheet provided by the ActiveX ControlWizard.
5. From the Property combo box, choose the DibFileName property, and in the Value edit control, type the name of a bitmap or DIB file, such as the one in \Windows\Forest.Bmp. When you click Apply, the palette displays in the control.

The completed code for this exercise is in \Labs\C12\Lab01\Ex01.

Exercise 2: Setting Properties for an ActiveX Control

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\C12\Lab01\Ex01.

In this exercise, you will implement the rest of the properties for the control.

➢ Implement the ShowColorSelection and Show3D properties

1. The ShowColorSelection and Show3D properties are member variables of **CPalViewCtrl**. They are implemented in the same way as the DibFileName property in Exercise 1, as member variables with notification functions. Using the ClassWizard, create the following properties.

External Name	Type	Variable Name	Notification Function
ShowColorSelection	BOOL	m_showColorSelection	OnShowColorSelectionChanged
Show3D	BOOL	m_show3D	OnShow3DChanged

2. When ShowColorSelection is changed, redraw the control in order show or hide the red outlining rectangle indicating the color tile selected. Invalidate the control in OnShowColorSelectionChanged.

```
void CPalViewCtrl::OnShowColorSelectionChanged()
{
InvalidateControl();
SetModifiedFlag();
}
```

3. Show3D must be passed to **CDIBPal**. Call SetDraw3D in OnShow3DChanged.

```
void CPalViewCtrl::OnShow3DChanged()
{
if (0 != m_pDibPal)
{
    m_pDibPal->SetDraw3D(m_show3D);
    InvalidateControl();
}
    SetModifiedFlag();
}
```

➢ Implement the SelectionIndex property

1. SelectionIndex exists only in **CDIBPal**. Implement this property as a pair of Get/Set methods.

External Name	Type	Get	Set
SelectionIndex	short	GetSelectionIndex	SetSelectionIndex

2. To get the selection index, use **CDIBPal::GetSelectionIndex** to query the control. Return -1 if you do not have a **CDIBPal** instance.

```
short CPalViewCtrl::GetSelectionIndex()
{
short nSelectionIndex = -1;
if (0 != m_pDibPal)
{
    nSelectionIndex = m_pDibPal->GetSelectionIndex();
}
return nSelectionIndex;
}
```

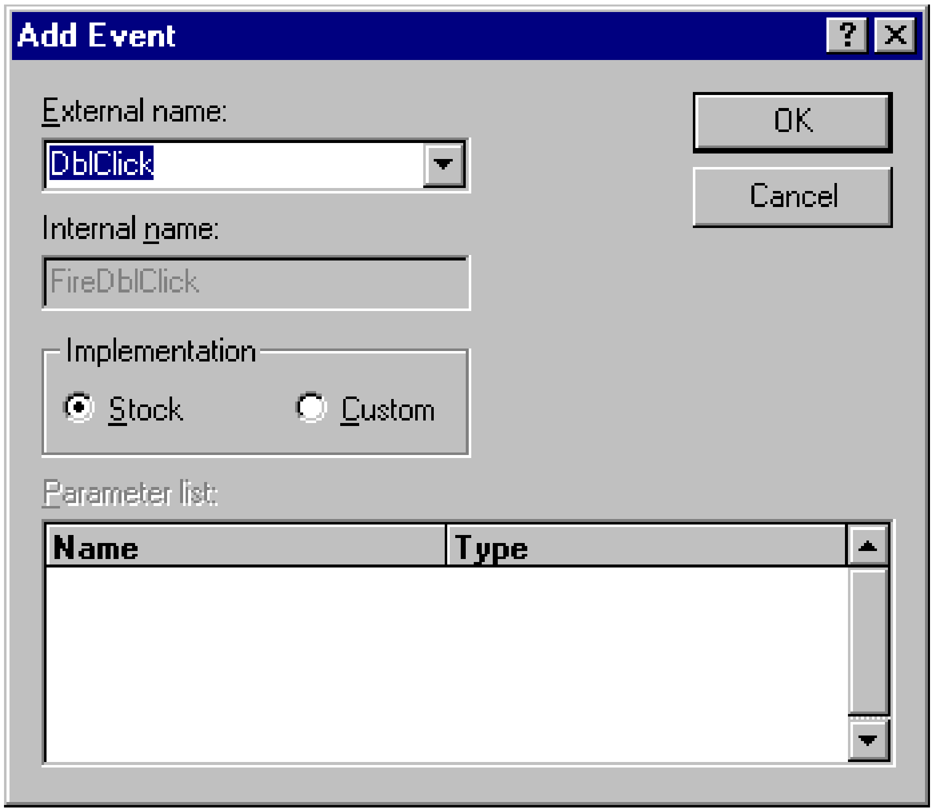
3. To set the selection index, use **CDIBPal::SetSelectionIndex**.

```
void CPalViewCtrl::SetSelectionIndex(short nNewValue)
{
if(nNewValue >= 0 && nNewValue < 256)
{
    if (0 != m_pDibPal)
    {
        m_pDibPal->SetSelectionIndex(nNewValue);
```

```
            FireSelChange(nNewValue);
            InvalidateControl();
        }
    }
    SetModifiedFlag();
    }
```

➢ **Add event handling for left-click and double-click**

1. Start the ClassWizard. Choose the ActiveX Events tab and click Add Event. The Add Event dialog will display. Add a Stock event for **DblClick**.



Add Event

External name: DblClick

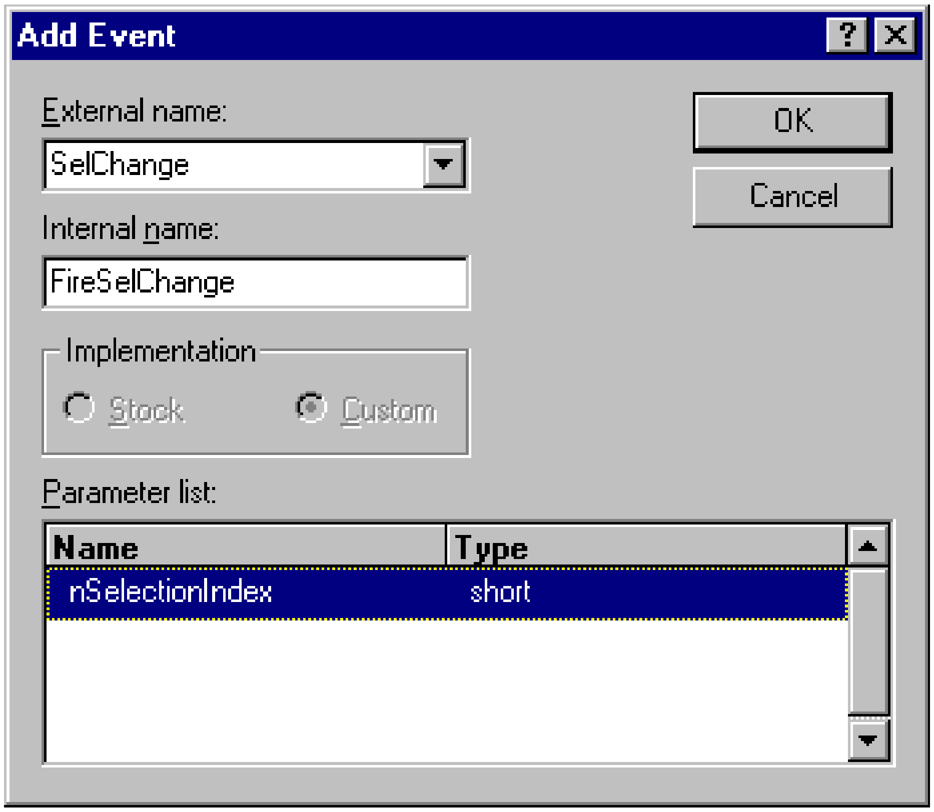
Internal name: FireDblClick

Implementation: [✓ Stock] Custom

Parameter list:

Name	Type

2. Add a custom event for **SelChange**. Add a short parameter, nSelectionIndex.


Add Event

External name: SelChange

Internal name: FireSelChange

Implementation: ( ) Stock [✓ Custom]

Parameter list:

Name	Type
nSelectionIndex	short

3. Open the Message Maps tab in the ClassWizard and add a handler for WM_LBUTTONDOWN. Edit the code for **OnLButtonDown**.
4. Validate that the ActiveX control has instantiated a **CDIBPal**, and if so, get the size of the control.

```
if(0 != m_pDibPal)
{
    int x, y;
    GetControlSize(&x, &y);
    CRect rcControl(0, 0, x, y);
```

5. Pass the **CRect** and the **CPoint** to **CDIBPal::HitTest** to get the cell that was clicked. Then, set the selection index.

```
int nPalIndex = m_pDibPal->HitTest(rcControl, point);
if(-1 != nPalIndex)
{
    SetSelectionIndex(nPalIndex);
}
```

6. Save PalViewCtl.Cpp. The code for completed function follows.

```
void CPalViewCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
if(0 != m_pDibPal)
{
    int x, y;
    GetControlSize(&x, &y);
    CRect rcControl(0, 0, x, y);

    //  Ask the DibPal object which palette index would
    //  be hit by the mouse click
    int nPalIndex = m_pDibPal->HitTest(rcControl, point);
```

```
        if(-1 != nPalIndex)
        {
            //Change the selection property
            SetSelectionIndex(nPalIndex);
        }
    }
    COleControl::OnLButtonDown(nFlags, point);
    }
```

➤ **Implement a method to get the current color from the palette**

1. Use the Automation tab of the ClassWizard to add a method to get a color from the palette.

External Name	Return Type	Parameter Name	Parameter Type
GetColorFromPalette	long	nPalIndex	short

2. Edit the code. Check that the palette index is between 0 and 255.

```
COLORREF    crRet = 0; // Default to BLACK
if(nPalIndex >= 0 && nPalIndex < 256)
{
```

3. Return **CDIBPal::GetColorFromIndex** if there is a **CDIBPal** object.

```
if(0 != m_pDibPal)
{
    crRet = m_pDibPal->GetColorFromIndex(nPalIndex);
}
return crRet;
```

4. Save PalViewCtl.Cpp. The complete function follows.

```
long CPalViewCtrl::GetColorFromPalette(short nPalIndex)
{
    COLORREF    crRet = 0; // Default to BLACK
    if(nPalIndex >= 0 && nPalIndex < 256)
    {
        if(0 != m_pDibPal)
        {
            crRet = m_pDibPal->GetColorFromIndex(nPalIndex);
        }
    }
    return crRet;
}
```

➤ **Modify ControlWizard-provided functions**

DoPropExchange is called by the framework when loading or storing a control from a persistent storage representation, such as a stream or property set. This function normally makes calls to the **PX_** family of functions to load or store specific user-defined properties of an ActiveX control. Add **PX_** calls for the properties you just implemented.

```
PX_Bool(pPX, _T(“ShowColorSelection”), m_showColorSelection, TRUE);
PX_Bool(pPX, _T(“Show3D”), m_show3D, FALSE);
PX_String(pPX, _T(“DibFileName”), m_dibFileName, _T(“”));
```

1. After loading DibFileName, get the palette from that file.

```
if (pPX->IsLoading())
```

```
{
    ExtractPalFromDIBFile(m_dibFileName);
}
```

2. Save PalViewCtl.Cpp. The compete function follows.

```
void CPalViewCtrl::DoPropExchange(CPropExchange* pPX)
{
ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
COleControl::DoPropExchange(pPX);

PX_Bool(pPX, _T("ShowColorSelection"),
    m_showColorSelection, TRUE);
PX_Bool(pPX, _T("Show3D"), m_show3D, FALSE);
PX_String(pPX, _T("DibFileName"), m_dibFileName, _T(""));

if (pPX->IsLoading())
{
    ExtractPalFromDIBFile(m_dibFileName);
}

}
```

3. Initialize m_showColorSelection and m_show3D in the **CPalViewCtrl** constructor. The complete constructor follows.

```
CPalViewCtrl::CPalViewCtrl()
{
InitializeIIDs(&IID_DPalView, &IID_DPalViewEvents);
m_pDibPal = 0;
m_dibFileName = _T("");

m_showColorSelection = TRUE;
m_show3D = FALSE;

}
```

4. In ExtractPalFromDIBFile, modify the call to **CDIBPal::SetDraw3D** to use m_show3D.

```
m_pDibPal->SetDraw3D(m_show3D);
```

5. In **CPalViewCtrl::OnDraw** modify the call to **CDIBPal::Draw** to use m_showColorSelection.

```
m_pDibPal->Draw(pdc, rcBounds, m_showColorSelection);
```

6. Save PalViewCtl.Cpp. Build and run PalView.Ocx using the ActiveX Control Test Container.

The completed code for this exercise is in \Labs\C12\Lab01\Ex02.

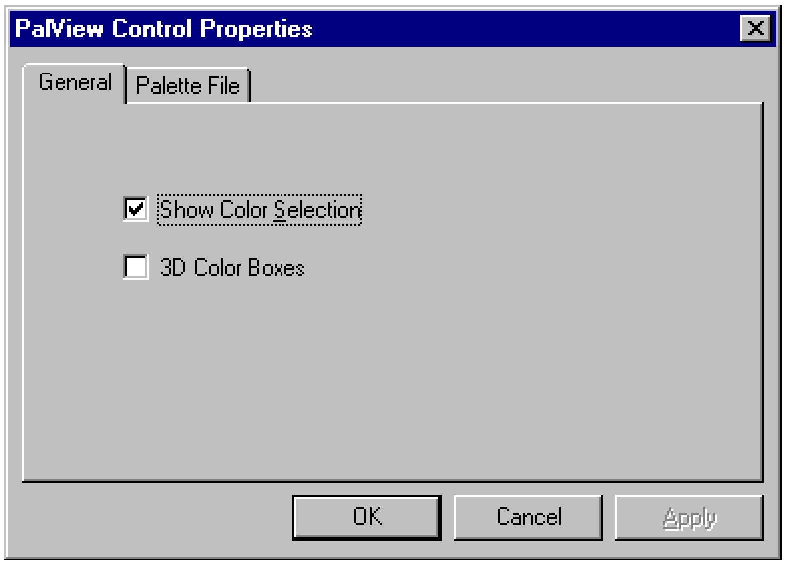
Exercise 3: Adding a Property Sheet Interface

Continue with the files you created in 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\C12\Lab01\Ex02.

You have implemented a programmer's interface to the palette view. Now, you will use a property sheet to create a design-mode interface. The ControlWizard provides a property sheet with one blank page. In this exercise, you will implement this page and add another page.

➢ Modify the AppWizard-provided property page

The first property page will have two check boxes: one for the ShowColorSelection property and the other for the Show3D property.



1. Delete the static text control on the IDD_PROPPAGE_PALVIEW dialog template. Add and position controls on this template according to the following table.

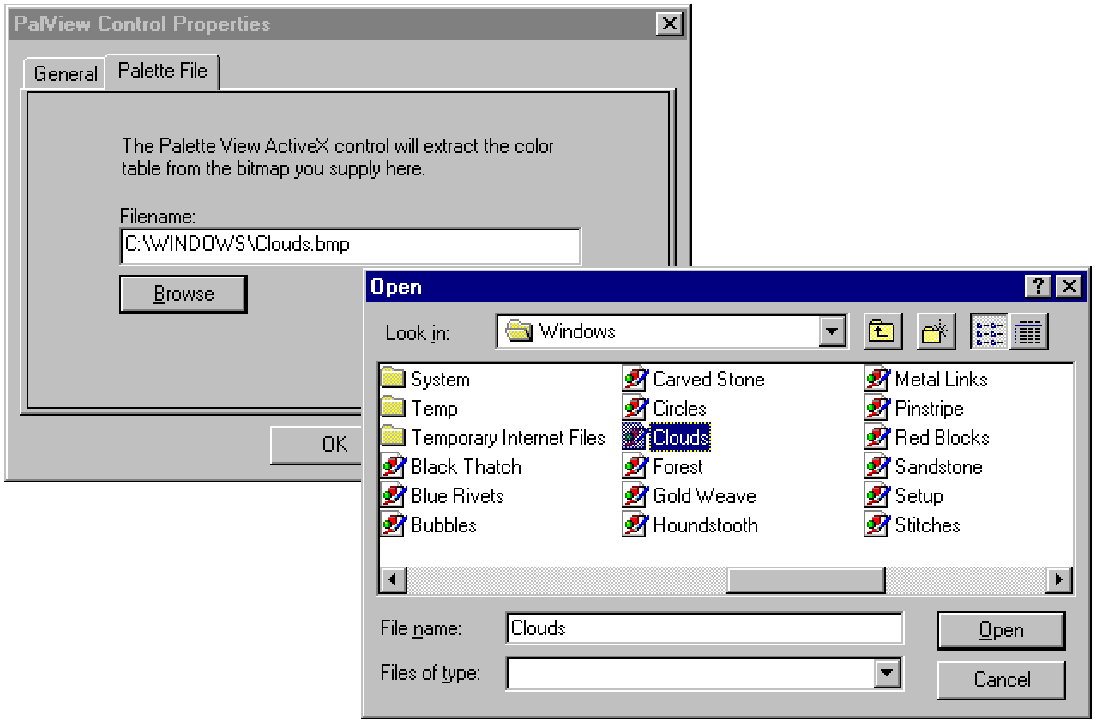
Type	ID	Caption
Checkbox	IDC_COLORSELECTION	Show Color &Selection
Checkbox	IDC_3D_BOXES	&3D Color Boxes

2. Create member variables and link them to their properties according to the following table. Press the CTRL key, and double-click controls to display their variables.

ID	Variable Name	Type	Property Name
IDC_COLORSELECTION	m_bShowColorSelection	BOOL	ShowColorSelection
ICD_3D_BOXES	m_bShowID	BOOL	Show3D

➢ Add a new property page

The new property page will enable your users to set the palette to be displayed.



1. Create a new dialog template. From the Insert menu, choose Resource to display the properties resources. Expand the Dialog icon, choose IDD_OLE_PROPPAGE_SMALL and add this dialog template with the following controls:

Type	ID	Caption
Dialog	IDD_PROPPAGE_DIBFILE	
Pushbutton	IDC_BROWSE	&Browse
Edit control	IDC_DIB_FILENAME	
Left aligned static	IDC_STATIC	Filename:
Left aligned static	IDC_STATIC	The Palette View ActiveX control will extract the color table from the bitmap file you supply here.

2. Use the ClassWizard to create a new class based on IDD_PROPPAGE_DIBFILE. Make the base class **COlePropertyPage**. Name the new class **CDibFilePropPage** and use DibPrpg.Cpp and DibPrpg.H as file names.
3. Create a member variable for the edit control.

Variable Name	ID	Type	Property Name
m_DibFileName	IDC_DIB_FILENAME	CString	DibFileName

4. In **CDibFilePropPage**, create a command handler for IDC_BROWSE.
5. In **CDibFilePropPage::OnBrowse**, display a common file dialog and set the text of the file name edit control with the results. The complete function follows.

```
void CDibFilePropPage::OnBrowse()
{
CFileDialog dlg(TRUE);
if(dlg.DoModal() == IDOK)
{
    CEdit * pEdit = (CEdit *)GetDlgItem(IDC_DIB_FILENAME);
    if(0 != pEdit)
    {
        pEdit->SetWindowText(dlg.GetPathName());
    }
}
}
```

6. In the string table resource, add two strings for captions.

ID	Caption
IDS_DIBPAGE	Palette File
IDS_DIBPAGE_CAPTION	Palette File

7. In the **UpdateRegistry** function, modify the call to **AfxOleRegisterPropertyPageClass** to use IDS_DIBPAGE instead of 0. Note that ClassView does not display this member function; it is located in the .Cpp file. Code for the completed function should look like this:

```
BOOL CDibFilePropPage::CDibFilePropPageFactory::UpdateRegistry(BOOL
bRegister)
{
// TODO: Define string resource for page type;
// replace ‘0’ below with ID.
if (bRegister)
    return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
        m_clsid, IDS_DIBPAGE);
else
    return AfxOleUnregisterClass(m_clsid, NULL);
}
```

8. In **CDibFilePropPage::CDibFilePropPage**, change the initialization to use IDS_DIBPAGE_CAPTION instead of 0. The complete function follows.

```
CDibFilePropPage::CDibFilePropPage() :
    COlePropertyPage(IDD, IDS_DIBPAGE_CAPTION)
{
    //{{AFX_DATA_INIT(CDibFilePropPage)
    m_DibFileName = _T(“”);
    //}}AFX_DATA_INIT
}
```

9. Save this file.

➢ **Setup the new property page in CPalViewCtrl**

1. Include DibFilePropPage.H in PalViewCtl.Cpp.
2. Change the BEGIN_PROPPAGEIDS macro to indicate that you will have two pages in the property sheet.

```
BEGIN_PROPPAGEIDS(CPalViewCtrl, 2)
```

3. Add an additional PROPPAGEID macro for **CDibFilePropPage**.

```
PROPPAGEID(CDibFilePropPage::guid)
```

4. Save PalVwCtl.Cpp. The complete macro block follows.

```
BEGIN_PROPPAGEIDS(CPalViewCtrl, 2)
    PROPPAGEID(CPalViewPropPage::guid)
    PROPPAGEID(CDibFilePropPage::guid)
```

```
  END_PROPPAGEIDS(CPalViewCtrl)
```

➢ **Build and use the control**

The completed code for this exercise is in \Labs\C12\Lab01\Ex03.