Module 19: Internet Explorer 4 Common Controls

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:

Internet Explorer 4 Common Controls The Common Control Description The Date and Time Picker CTime VS. COleDateTime Class The Month Calendar The Internet Protocol (IP) Address Control The Extended Combo Box The MYMFC25A Example

Internet Explorer 4 Common Controls

When Microsoft developers released Internet Explorer 4 (IE4), they included a new and improved version of the COMCTL32.DLL, which houses Microsoft Windows **Common Controls**. Since this update to the common controls was not part of an operating system release, Microsoft calls the update Internet Explorer 4 Common Controls. IE4 Common Controls updates all of the existing controls and adds a variety of advanced new controls. Microsoft Visual C++ 6.0 and Microsoft Foundation Class (MFC) 6.0 have added a great deal of support for these new controls. In this module, we'll look at the new controls and show examples of how to use each one. If you haven't worked with Windows controls or Windows Common Controls, be sure you read Module 5 before proceeding with IE4 Common Controls. While Microsoft Windows 95 and Microsoft Windows NT 4.0 do not include the new COMCTL32.DLL, future versions of Windows will. To be safe, you will need to redistribute the COMCTL32.DLL for these existing operating systems as part of your installation. Currently you must ship a "developer's edition" of Internet Explorer to be able to redistribute these controls. However, this might change once a version of Windows ships with the updated controls. Currently we have IE 6 and IE 7 in beta mode.

The Common Control Description

Example MYMFC25A uses each of the IE4 common controls. Figure 1 shows the dialog from that example. Refer to it when you read the control descriptions that follow.

Dialog			
New Date/Time Controls		- IP Address Edit Control	
Choose a date (short):	3/14/2006 03/14/2006	Choose your IP address:	172 . 16 . 6 . 120 72.16.6.120
Choose a date (long):	Sunday , February 05, 2 - 02/05/2006	- Extended Combo Box	
Choose a date (short with allow short and spinners):	▼ 3/27/2006 3/27/2006	Old fashion combo:	aura
Choose a time:	8/13/2002 · 13:24:33	Cool new Imagelist support:	red 💌
Choose your birtday:	11/09/2005		<pre>red red blue juellow cyan purple green Doremon Tweety</pre>
27 28 24 5	29 30 1 2 3 6 7 8 9 10	cel	r Mack 🔽

Figure 1: The Internet Explorer 4 Common Controls dialog.

The Date and Time Picker

A common field on a dialog is a place for the user to enter a date and time. Before IE4 controls provided the date and time picker, developers had to either use a third-party control or subclass an MFC edit control to do significant data validation to ensure that the entered date was valid. Fortunately, the new date and time picker control is provided as an advanced control that prompts the user for a date or time while offering the developer a wide variety of styles and options. For example, dates can be displayed in short formats (8/14/68) or long formats (August 14, 1968). A time mode lets the user enter a time using a familiar hours/minutes/seconds AM/PM format.

The control also lets you decide if you want the user to select the date via in-place editing, a pull-down calendar, or a spin button. Several selection options are available including single and multiple select (for a range of dates) and the ability to turn on and off the "circling" in red ink of the current date. The control even has a mode that lets the user select "no date" via a check box. In Figure 1, the first four controls on the left illustrate the variety of configurations available with the date and time picker control.

The new MFC 6.0 class CDateTimeCtrl provides the MFC interface to the IE4 date and time picker common control. This class provides a variety of notifications that enhance the programmability of the control.

CDateTimeCtrl provides member functions for dealing with either CTime or COleDateTime time structures. You set the date and time in a CDateTimeCtrl using the SetTime() member function. You can retrieve the date and time via the GetTime() function. You can create custom formats using the SetFormat() member function and change a variety of other configurations using the CDateTimeCtrl interface.

CTime VS. COleDateTime Class

Most "longtime" MFC developers are accustomed to using the CTime class. However, since CTime's valid dates are limited to dates between January 1, 1970, and January 18, 2038, many developers are looking for an alternative. One

popular alternative is COleDateTime, which is provided for OLE automation support and handles dates from 1 January 100 through 31 December 9999. Both classes have various pros and cons. For example, CTime handles all the issues of daylight savings time, while COleDateTime does not. Many developers choose COleDateTime because of its much larger range. Any application that uses CTime will need to be reworked in approximately 40 years, since the maximum value is the year 2038. To see this limitation in action, select a date outside the CTime range in MYMFC25A. The class you decide to use will depend on your particular needs and the potential longevity of your application.

The Month Calendar

The large display at the bottom left of Figure 1 is a **Month Calendar**. Like the date and time picker control, the month calendar control lets the user choose a date. However, the month calendar control can also be used to implement a small **Personal Information Manager** (PIM) in your applications. You can show as many months as room provides, from one month to a year's worth of months, if you want. MYMFC25A uses the month calendar control to show only two months. The month calendar control supports single or multiple selection and allows you to display a variety of different options such as numbered months and a circled "today's date." Notifications for the control let the developer specify which dates are in boldface. It is entirely up to the developer to decide what boldface dates might represent. For example, you could use the bold feature to indicate holidays, appointments, or unusable dates. The MFC 6.0 class CMonthCalCtrl implements this control.

To initialize the CMonthCalCtrl class, you can call the SetToday() member function. CMonthCalCtrl provides members that deal with both CTime and COleDateTime, including SetToday().

The Internet Protocol (IP) Address Control

If you write an application that uses any form of Internet or TCP/IP functionality, you might need to prompt the user for an Internet Protocol (IP) Address. The IE4 common controls include an IP address edit control as shown in the top right of Figure 1. In addition to letting the user enter a 4-byte IP address, this control performs an automatic validation of the entered IP address. CIPAddressCtrl provides MFC support for the IP address control. An IP address consists of four "fields" as shown in Figure 2. The fields are numbered from left to right.



Figure 2: The fields of an Internet Protocol (IP) address control.

To initialize an IP address control, you call the SetAddress() member function in your OnInitDialog() function. SetAddress() takes a DWORD, with each BYTE in the DWORD representing one of the fields. In your message handlers, you can call the GetAddress() member function to retrieve a DWORD or a series of BYTES to retrieve the various values of the four IP address fields.

The Extended Combo Box

The "old-fashioned" combo box was developed in the early days of Windows. Its age and inflexible design have been the source of a great deal of developer confusion. With the IE4 controls, Microsoft has decided to release a much more flexible version of the combo box called the extended combo box.

The extended combo box gives the developer much easier access to and control over the edit-control portion of the combo box. In addition, the extended combo box lets you attach an image list to the items in the combo box. You can display graphics in the extended combo box easily, especially when compared with the old days of using owner-drawn combo boxes. Each item in the extended combo box can be associated with three images: a selected image, an unselected image, and an overlay image. These three images can be used to provide a variety of graphical displays in the

combo box, as we'll see in the MYMFC25A sample. The bottom two combo boxes in Figure 1 are both extended combo boxes. The MFC CComboBoxEx class provides comprehensive extended combo box support. Like the list control introduced in Module 6, CComboBoxEx can be attached to a CImageList that will automatically display graphics next to the text in the extended combo box. If you are already familiar with CComboBox, CComboBoxEx might cause some confusion: instead of containing strings, the extended combo box contains items of type COMBOBOXEXITEM, a structure that consists of the following fields:

- **UINT mask**: A set of bit flags that specify which operations are to be performed using the structure. For example, set the CBEIF_IMAGE flag if the image field is to be set or retrieved in an operation.
- **int iItem**: The extended combo box item number. Like the older style of combo box, the extended combo box uses zero-based indexing.
- LPSTR pszText: The text of the item.
- **int cchTextMax**: The length of the buffer available in pszText.
- int iImage: Zero-based index into an associated image list.
- **int iSelectedImage**: Index of the image in the image list to be used to represent the "selected" state.
- **int iOverlay**: Index of the image in the image list to be used to overlay the current image.
- **int iIndent**: Number of 10-pixel indentation spaces.
- LPARAM lParam: 32-bit parameter for the item.

You will see first-hand how to use this structure in the MYMFC25A example.

The MYMFC25A Example

To illustrate how to take advantage of the Internet Explorer 4 Common Controls, we'll build a dialog that demonstrates how to create and program each control type. The steps required to create the dialog are shown below.

Run AppWizard to generate the MYMFC25A project. Choose **New** from the Visual C++ **File** menu, and then select **Microsoft AppWizard (exe)** from the **Projects** page. Accept all the defaults but one: choose **Single Document Interface** (SDI). The options and the default class names are shown here.

New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of mymfc25A: Single Document Interface Application targeting: Win32
Classes to be created: Application: CMymfc25AApp in mymfc25A.h and mymfc25A.cpp Frame: CMainFrame in MainFrm.h and MainFrm.cpp Document: CMymfc25ADoc in mymfc25ADoc.h and mymfc25ADoc.cpp View: CMymfc25AView in mymfc25AView.h and mymfc25AView.cpp
Features: + Initial toolbar in main frame + Initial status bar in main frame + Printing and Print Preview support in view + 3D Controls + Uses shared DLL implementation (MFC42.DLL) + ActiveX Controls support enabled + Localizable text in: English [United States]
Project Directory: F:\mfcproject\mymfc25A
Cancel

Figure 3: MYMFC25A IE4 common controls project summary.

Create a new dialog resource with ID IDD_DIALOG1. Place the controls as shown in Figure 1.

You can drag the controls from the control palette. Remember that IE4 **Common Controls** are at the bottom part of the palette. The following table lists the control types and their IDs.

Tab Sequence	Control Type	Child Window ID
1	Group Box	IDC_STATIC
2	Static	IDC_STATIC
3	Date Time Picker	IDC_DATETIMEPICKER1
4	Static	IDC_STATIC1
5	Static	IDC_STATIC
6	Date Time Picker	IDC_DATETIMEPICKER2
7	Static	IDC_STATIC2
8	Static	IDC_STATIC
9	Date Time Picker	IDC_DATETIMEPICKER3
10	Static	IDC_STATIC3
11	Static	IDC_STATIC
12	Date Time Picker	IDC_DATETIMEPICKER4
13	Static	IDC_STATIC4
14	Static	IDC_STATIC
15	Month Calendar	IDC_MONTHCALENDAR1
16	Static	IDC_STATIC5
17	Group Box	IDC_STATIC
18	Static	IDC_STATIC
19	IP Address	IDC_IPADDRESS1
20	Static	IDC_STATIC6

21	Group Box	IDC_STATIC
22	Static	IDC_STATIC
23	Extended Combo Box	IDC_COMBOBOXEX1
24	Static	IDC_STATIC7
25	Static	IDC_STATIC
26	Extended Combo Box	IDC_COMBOBOXEX2
27	Static	IDC_STATIC8
28	Pushbutton	IDOK
29	Pushbutton	IDCANCEL

Table 1: MYMFC25A controls and their IDs.

The following figure shows each control and its appropriate tab order.

Dialog	×
Land Date/Time Controls	Address Edit Control
2ose a date (short):	18 ose your IP address: 19 20 ent selection
5bse a date (long): 6 Saturday , March 25, 2 ▼ 7ent selection	21 tended Combo Box
8ose a date (short with allow short and spinners):	22 tashion combo: 23
12 June a time: 12 June 12 Jun	25 new Imagelist support:
14 ose your birtday: 16 ent selection	
March, 2006	
Sun Mon Tue Wed Thu Fri Sat	
26 27 28 1 2 3 4	
12 13 14 15 16 17 18	
19 20 21 22 23 24 23	
26 27 28 29 30 31 1	
2 3 4 5 6 7 8	
28 OK 29 Ca	ancel

Figure 4: MYMFC25A controls and their tab order.

Until we set some properties, your dialog will not look exactly like the one in Figure 1.

Use ClassWizard to create a new class, CDialog1, derived from CDialog. ClassWizard will automatically prompt you to create this class because it knows that the IDD_DIALOG1 resource exists without an associated C^{++} class and just go ahead.



Figure 5: A new class creation dialog prompt for IDD_DIALOG1.

New Class		? 🔀
Class informatior <u>N</u> ame: File name:	CDialog1 Dialog1.cpp	OK Cancel
<u>B</u> ase class: <u>D</u> ialog ID:	CDialog IDD_DIALOG1	
Automation None Automation C Createable b	y type ID; mymfc25A.Dialog1	

Figure 6: CDialog1 class information.

Then, create a message handler for the WM_INITDIALOG message.

MFC ClassWizard	J			? 🛛
MFC ClassWizard Message Maps Project: mymfc25A F:\\mymfc25A\Dia Object IDs: CDialog1 ID_APP_ABOUT ID_APP_ABOUT ID_EDIT_COPY ID_EDIT_COPY ID_EDIT_CUT ID_EDIT_CUT ID_EDIT_PASTE ID_EDIT_UNDO Member functions: V DoDataExcha	Member Variables	Automation ActiveX Events Class <u>n</u> ame: CDialog1 A\Dialog1.cpp Messages: WM_HELPINFO WM_HECROLL WM_HECROLL WM_KEYDOWN WM_KEYDOWN WM_KEYUP WM_KILLFOCUS WM_LBUTTONDBLCLK NITDIALOG	Class Info	Add Class Add Function Delete Function Edit Code
Description: Se	nt to a dialog box befor	e the dialog box is displayed	01	Cancel

Figure 7: Creating a message handler for the WM_INITDIALOG message.

Set the properties for the dialog's controls. To demonstrate the full range of controls, we will need to set a variety of properties on each of the IE4 common controls in the example. Here is a brief overview of each property you will need to set:

• The **Short Date** and **Time Picker**. To set up the first date and time picker control to use the short format, select the properties for IDC_DATETIMEPICKER1, as shown in the following figure.



Figure 8: Modifying Date Time Picker control properties.

• The **Long Date** and **Time Picker**. Now configure the second date and time picker control (IDC_DATETIMEPICKER2) to use the long format as shown below.



Figure 9: Modifying Date Time Picker control properties.

• The **Short** and **NULL Date** and **Time Picker**. This is the third date and time picker control, IDC_DATETIMEPICKER3. Configure this third date and time picker to use the short format and the styles shown here.

Date Time Picker Pro	perties	×
9 ? General	Styles Extended Styles	
<u>F</u> ormat: Short Date Short Date Long Date Time	 ✓ <u>Right Align</u> ✓ Show <u>None</u> ✓ Use <u>Spin Control</u> ✓ Allow <u>E</u>dit 	

Figure 10: Modifying Date Time Picker control properties.

• The **Time Picker**. The fourth date and time picker control, IDC_DATETIMEPICKER4, is configured to let the user choose time. To configure this control, select **Time** from the **Format** combo box on the **Styles** tab as shown.



Figure 11: Modifying Date Time Picker control properties.

• The Month View. To configure the month view, you will need to set a variety of styles. First, from the Styles tab, choose Day States, as shown here.



Figure 12: Modifying Month Calendar control properties.

If we leave the default styles, the month view does not look like a control on the dialog. There are no borders drawn at all. To make the control fit in with the other controls on the dialog, select **Client Edge** and **Static Edge** from the **Extended Styles** tab, as shown below.

Month Ca	lendar Prop	oerties		×
-m ?	General	Styles Extende	d Styles	
☑ <u>C</u> lier ☑ Stat ☑ <u>M</u> oo	nt edge ic e <u>dg</u> e Jal frame	☐ Transgarent ☐ Accept files	 Right aligned text <u>Right-to-left reading order</u> Left scroll bar 	

Figure 13: Modifying Month Calendar control properties.

- The IP Address. This control (IDC_IPADDRESS1) does not require any special properties.
- The **First Extended Combo Box**. Make sure that you enter some items, as shown here, and also make sure the list is tall enough to display several items. Use **Ctrl** + **Enter** to go to new line.



Figure 14: Modifying Extended Combo Box control properties.

• The Second Extended Combo Box. Enter three items: Doremon, Tweety, Mack, Pink Panther, Ultraman Ace and Jaws. Later in the example, we will use these items to show one of the ways to draw graphics in an extended combo box.



Figure 15: Modifying Extended Combo Box control properties.

Add the CDialog1 variables. Start ClassWizard and click on the **Member Variables** tab to view the **Member Variables** page. Enter the following member variables for each control listed.

Control ID	Data Member	Туре
IDC_DATETIMEPICKER1	m_MonthCall	CDateTimeCtrl
IDC_DATETIMEPICKER2	m_MonthCal2	CDateTimeCtrl
IDC_DATETIMEPICKER3	m_MonthCal3	CDateTimeCtrl
IDC_DATETIMEPICKER4	m_MonthCal4	CDateTimeCtrl
IDC_IPADDRESS1	m_ptrIPCtrl	CIPAddressCtrl
IDC_MONTHCALENDAR1	m_MonthCal5	CMonthCalCtrl
IDC_STATIC1	m_strDate1	CString
IDC_STATIC2	m_strDate2	CString
IDC_STATIC3	m_strDate3	CString
IDC_STATIC4	m_strDate4	CString
IDC_STATIC5	m_strDate5	CString
IDC_STATIC6	m_strIPValue	CString
IDC_STATIC7	m_strComboEx1	CString
IDC_STATIC8	m_strComboEx2	CString

Table 2.

MFC ClassWizard					? 🔀
Message Maps Me	ember Variables 🛛 Au	tomation Ad	ctiveX Events Cl	lass Info	
Project: mymfc25A F:\\mymfc25A\Dialog	▼ g1.h, F:\\mymfc25A\E	Class <u>n</u> ame: CDialog1 Dialog1.cpp		•	Add Class ▼
Control IDs: IDC_DATETIMEPICKE IDC_IPADDRESS1 IDC_MONTHCALEND IDC_STATIC1 IDC_STATIC2 IDC_STATIC3 IDC_STATIC4 IDC_STATIC5 IDC_STATIC5 IDC_STATIC7 IDC_STATIC8 Description: CString Maximum Characters:	Typ ER4 CD a CIP AR1 CM CStr CStr CStr CStr CStr CStr CStr CStr	e h ateTimeCtrl r AddressCtrl r onthCalCtrl r ring r ring r ring r ring r ring r ring r	Member n_MonthCal4 n_ptrIPCtrl n_MonthCal5 n_strDate1 n_strDate2 n_strDate3 n_strDate4 n_strIPValue n_strIPValue n_strComboEx1 m_strComboEx2		Delete Variable
				ОК	Cancel

Figure 16: Adding the CDialog1 variables.

Program the short date time picker. In this example, we don't mind if the first date time picker starts with the current date, so we don't have any OnInitDialog() handling for this control. However, if we wanted to change the date, we would make a call to SetTime() for the control in OnInitDialog(). At runtime, when the user selects a new date in the first date and time picker, the companion static control should be automatically updated.

To achieve this, we need to use ClassWizard to add a handler for the DTN_DATETIMECHANGE message. Start ClassWizard (or CTRL-W) and choose IDC_DATETIMEPICKER1 from the **Object IDs** list and DTN_DATETIMECHANGE from the **Messages** list. Accept the default message name and click **OK**. Repeat this step for each of the other three IDC_DATETIMEPICKER IDs. Your ClassWizard should look like the illustration here.

Project:	Class <u>n</u> ame:	Add Class 🔻
mymfc25A	CDialog1	
:\\mymfc25A\Dialog1.h, F:	\\mymfc25A\Dialog1.cpp	Add Function
)bject [Ds:	Messages:	Delete Function
IDC_COMBOBOXEX1 IDC_COMBOBOXEX2	Add Member Function ?	Edit Code
	Member function name:	
IDC_DATETIMEPICKER3	nDatetimechangeDatetimepicker4	
IDC_DATETIMEPICKER4	Lancel	
lember functions:	Message: DTN_DATETIMECHANGE	
V DoDataExchange	ODJECTID: IDC_DATETIMEFICKEN4	
W OnDatetimechangeDate	tinON_IDC_DATETIMEPICKER1:DTN_DATETIMECH/	
W OnDatetimechangeDate	tinON_IDC_DATETIMEPICKER2:DTN_DATETIMECH/	
W OnDatetimechangeDate	tinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH4	
W OnInitDialog	on wm initdialog	1

Figure 17: Adding a handler for the DTN_DATETIMECHANGE message.

Next add the following code to the handler for **Datetimepicker1** created by ClassWizard:

```
void CDialog1::OnDatetimechangeDatetimepicker1(NMHDR* pNMHDR, LRESULT* pResult)
     {
         CTime ct;
         m_MonthCall.GetTime(ct);
         m_strDate1.Format(_T("%02d/%02d/%2d"), ct.GetMonth(),ct.GetDay(),ct.GetYear());
         UpdateData(FALSE);
         *pResult = 0;
     }
void CDialog1::OnDatetimechangeDatetimepicker1(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Add your control notification handler code here
    CTime ct;
    m_MonthCall.GetTime(ct);
    m_strDate1.Format(_T("%02d/%02d/%2d"), ct.GetMonth(),ct.GetDay(),ct.GetYear());
    UpdateData(FALSE);
    *pResult = 0;
}
```

Listing 1.

This code uses the m_MonthCall data member that maps to the first date time picker to retrieve the time into the CTime object variable ct. It then calls the CString::Format member function to set the companion static string. Finally the call to UpdateData(FALSE) triggers MFC's DDX and causes the static to be automatically updated to m_strDatel.

Program the long date time picker. Now we need to provide a similar handler for the second date time picker.

```
{
         CTime ct;
         m_MonthCal2.GetTime(ct);
         m_strDate2.Format(_T("%02d/%02d/%2d"), ct.GetMonth(),ct.GetDay(),ct.GetYear());
         UpdateData(FALSE);
         *pResult = 0;
     }
void CDialog1::OnDatetimechangeDatetimepicker2(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Add your control notification handler code here
    CTime ct;
    m_MonthCal2.GetTime(ct);
    m_strDate2.Format(_T("%02d/%02d/%2d"), ct.GetMonth(),ct.GetDay(),ct.GetYear());
    UpdateData(FALSE);
    *pResult = 0;
}
```

Listing 2.

Program the third date time picker. The third date time picker needs a similar handler, but since we set the **Show None** style in the dialog properties, it is possible for the user to specify a **NULL** date by checking the inline check box. Instead of blindly calling GetTime(), we have to check the return value. If the return value of the GetTime() call is nonzero, the user has selected a NULL date. If the return value is zero, a valid date has been selected. As in the previous two handlers, when a CTime object is returned, it is converted into a string and automatically displayed in the companion static control.

```
void CDialog1::OnDatetimechangeDatetimepicker3(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Add your control notification handler code here
    //NOTE: this one can be null!
    CTime ct;
    int nRetVal = m_MonthCal3.GetTime(ct);
    if (nRetVal) //If not zero, it's null; and if it is,
                 // do the right thing.
    {
        m_strDate3 = "NO DATE SPECIFIED!!";
    }
    else
    {
        m_strDate3.Format(_T("%02d/%02d/%2d"),ct.GetMonth(),
                           ct.GetDay(),ct.GetYear());
    UpdateData(FALSE);
I
    *pResult = 0;
}
```

Listing 3.

Program the time picker. The time picker needs a similar handler, but this time the format displays hours/minutes/seconds instead of months/days/years:

I

```
void CDialoq1::OnDatetimechangeDatetimepicker4(NMHDR* pNMHDR, LRESULT* pResult)
     {
         CTime ct;
         m_MonthCal4.GetTime(ct);
         m_strDate4.Format(_T("%02d:%02d:%2d"), ct.GetHour(), ct.GetMinute(),
     ct.GetSecond());
         UpdateData(FALSE);
         *pResult = 0;
     }
void CDialog1::OnDatetimechangeDatetimepicker4(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Add your control notification handler code here
    CTime ct;
    m MonthCal4.GetTime(ct);
    m_strDate4.Format(_T("%02d:%02d:%2d"),
                       ct.GetHour(),ct.GetMinute(),ct.GetSecond());
    UpdateData(FALSE);
    *pResult = 0;
}
```

Listing 4.

Program the Month Selector. You might think that the month selector handler is similar to the date time picker's handler, but they are actually somewhat different. First of all, the message you need to handle for detecting when the user has selected a new date is the MCN SELCHANGE message. Select this message in the ClassWizard, as shown here.

Message Maps Member Variables A	utomation ActiveX Events Class Info	
<u>Project:</u>	Class <u>n</u> ame:	Add Class 👻
mymfc25A	CDialog1	Add Exection
F:\\mymfc25A\Dialog1.h, F:\\mymfc25A\	Dialog1.cpp	Add Function
Object IDs:	Messages:	Delete Function
IDC_DATETIMEPICKER1	MCN_GETDAYSTATE MCN_SELCHANGE MCN_SELECT	Edit Code
IDC_DATETIMEPICKER4 IDC_IPADDRESS1 IDC_MONTHCALENDAR1	NM Add Member Function	? 🛛
	Member function name:	OK
Member functions: V DoDataExchange W OnDatetimechangeDatetinON_IDC_DA W OnDatetimechangeDatetinON_IDC_DA	TETH Message: MCN_SELCHANGE TETH Object ID: IDC_MONTHCALENDAR1	Cancel
W UnDatetimechangeDatetinUN_IDC_DA W OnDatetimechangeDatetinON_IDC_DA Description: Notifies parent that selected		
		Canad

Figure 18: Adding message handler for IDC_MONTHCALENDAR1 ID.

In addition to the different message handler, this control uses GetCurSel() as the date time picker instead of GetTime(). The code below shows the MCN SELCHANGE handler for the month calendar control.

```
void CDialog1::OnSelchangeMonthcalendar1(NMHDR* pNMHDR, LRESULT* pResult)
     {
         CTime ct;
         m_MonthCal5.GetCurSel(ct);
         m_strDate5.Format(_T("%02d/%02d/%2d"), ct.GetMonth(),ct.GetDay(),ct.GetYear());
         UpdateData(FALSE);
         *pResult = 0;
     }
void CDialog1::OnSelchangeMonthcalendar1(NMHDR* pNMHDR, IRESULT* pResult)
{
    // TODO: Add your control notification handler code here
    CTime ct;
    m_MonthCal5.GetCurSel(ct);
    m_strDate5.Format(_T("%02d/%02d/%2d"), ct.GetMonth(),ct.GetDay(),ct.GetYear());
    UpdateData(FALSE);
    *pResult = 0;
}
```

Listing 5.

Program the IP control. First we need to make sure the control is initialized. In this example, we initialize the control to 0 by giving it a 0 DWORD value. If you do not initialize the control, each segment will be blank. To initialize the control, add this call to the CDialog1::OnInitDialog function:

m_ptrIPCtrl.SetAddress(0L);

I

Listing 6.

Now we need to add a handler to update the companion static control whenever the IP address control changes. First we need to add a handler for the IPN_FIELDCHANGED notification message using ClassWizard, as shown here.

Message Maps Member Variables Automation ActiveX Events Class Info Project: Class name: Add Class mymfc25A CDialog1 Add Class Ft\\nymfc25A\Dialog1.h, Ft\\mymfc25A\Dialog1.cpp Delete Function Dbject [Ds: Messages: Delete Function IDC_DATETIMEPICKER4 IPN_FIELDCHANGED Edit Code IDC_MONTHCALENDAR1 IDC_STATIC1 Edit Code IDC_STATIC1 IDC_STATIC2 Edit Code Member functions: V OnDatetimechangeD atetinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH/ W OnDatetimechangeD atetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH/ V W OnDatetimechangeD atetinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH/ V W OnDatetimechangeD atetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH/ V W OnDatetimechangeD atetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH/ V W OnFieldchangedIpaddress ON_IDC_IPADDRESS1:IPN_FIELDCHANGED V W OnlintDialoa ON WM INITDIALOG V	MFC ClassWizard	? 🛛
Project: Class <u>n</u> ame: Add Class mymfc25A CDialog1 Add Class F:\\mymfc25A\Dialog1.h, F:\\mymfc25A\Dialog1.cpp Delete Function Object [Ds: Messages: Delete Function IDC_DATETIMEPICKER4 IPN_FIELDCHANGED Edit Code IDC_MONTHCALENDAR1 IDC_STATIC2 Edit Code IDC_STATIC3 IDC_STATIC3 IDC_STATIC4 IDC_STATIC4 OnDatetimechangeD atetinON_IDC_DATETIMEPICKER2:DTN_DATETIMECH4 IM W OnDatetimechangeD atetinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH4 IM W OnDatetimechangeD atetinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH4 IM W OnlatetimechangeD atetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH4 IM W OnlatetimechangeD atetinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH4 IM W OnlatetimechangeD atetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH4 IM W OnlatetimechangeD atetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH4 IM W OnlatetimechangeD atetinON_IDC_PADDRESS1:IPN_FIELDCHANGED IM W OnliniDialoo ON WM INITDIALOG IM	Message Maps Member Variables Automation ActiveX Events Class Info	Ι.
Member functions: W OnDatetimechangeDatetinON_IDC_DATETIMEPICKER2:DTN_DATETIMECH4 W OnDatetimechangeDatetinON_IDC_DATETIMEPICKER3:DTN_DATETIMECH4 W OnDatetimechangeDatetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH4 W OnFieldchangedIpaddressON_IDC_IPADDRESS1:IPN_FIELDCHANGED W OnInitDialog ON WM_INITDIALOG	Project: Class name: mymfc25A CDialog1 F:\\mymfc25A\Dialog1.h, F:\\mymfc25A\Dialog1.cpp Object [Ds: Messages: IDC_DATETIMEPICKER4 IPN_FIELDCHANGED IDC_MONTHCALENDAR1 IDC_STATIC2 IDC_STATIC2 IDC_STATIC3 IDC_STATIC4 Image: Class name	Add Class Add Function Delete Function Edit Code
Description: Indicates the field in the control has changed		

Figure 19: Adding a handler for the IPN_FIELDCHANGED notification message using ClassWizard.

Next we need to implement the handler as follows:

```
void CDialog1::OnFieldchangedIpaddress1(NMHDR* pNMHDR, LRESULT* pResult)
{
    DWORD dwIPAddress;
    m_ptrIPCtrl.GetAddress(dwIPAddress);

    m_strIPValue.Format("%d.%d.%d %x.%x.%x",
    HIBYTE(HIWORD(dwIPAddress)),
    LOBYTE(HIWORD(dwIPAddress)),
    LOBYTE(LOWORD(dwIPAddress)),
    LOBYTE(LOWORD(dwIPAddress)),
    LOBYTE(HIWORD(dwIPAddress)),
    LOBY
```

```
HIBYTE(LOWORD(dwIPAddress)),
            LOBYTE(LOWORD(dwIPAddress)));
         UpdateData(FALSE);
         *pResult = 0;
     }
void CDialog1::OnFieldchangedIpaddress1(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Add your control notification handler code here
    DWORD dwIPAddress;
    m_ptrIPCtrl.GetAddress(dwIPAddress);
    m_strIPValue.Format("%d.%d.%d.%d
                                         %x.%x.%x.%x".
        HIBYTE(HIWORD(dwIPAddress)),
        LOBYTE(HIWORD(dwIPAddress)),
        HIBYTE(LOWORD(dwIPAddress)),
        LOBYTE(LOWORD(dwIPAddress)),
        HIBYTE(HIWORD(dwIPAddress)),
        LOBYTE(HIWORD(dwIPAddress)),
        HIBYTE(LOWORD(dwIPAddress))
        LOBYTE(LOWORD(dwIPAddress)));
    UpdateData(FALSE);
I
    *pResult = 0;
}
```

```
Listing 7.
```

The first call to CIPAddressCtrl::GetAddress retrieves the current IP address into the local dwIPAddress DWORD variable. Next we make a fairly complex call to CString::Format to deconstruct the DWORD into the various fields. This call uses the LOWORD macro to first get to the bottom word of the DWORD and the HIBYTE/LOBYTE macros to further deconstruct the fields in order from field 0 to field 3.

Add a handler for the first extended combo box. No special initialization is required for the extended combo box, but we do need to handle the CBN_SELCHANGE message. The following code shows the extended combo box handler. Can you spot the ways that this differs from a "normal" combo box control?

MFC ClassWizard 🛛 🛛 🔀			
Message Maps Member Variables Au	utomation ActiveX Events Class Info	1 .	
Project: mymfc25A F:\\mymfc25A\Dialog1.h, F:\\mymfc25A\	Class <u>n</u> ame: CDialog1 Dialog1.cpp	Add Class ▼ Add Function	
Object IDs: ID_PREV_PANE ID_VIEW_STATUS_BAR ID_VIEW_TOOLBAR IDC_COMBOBOXEX1 IDC_COMBOBOXEX2 IDC_DATETIMEPICKER1 IDC_DATETIMEPICKER2	Messages: CBN_DBLCLK CBN_DROPDOWN CBN_EDITUPDATE CBN_ERRSPACE CBN_SELCHANGE CBN_SELENDCANCEL	Delete Function	
Member functions: W OnDatetimechangeDatetinON_IDC_DATETIMEPICKER4:DTN_DATETIMECH4 W OnFieldchangedIpaddressON_IDC_IPADDRESS1:IPN_FIELDCHANGED W OnInitDialog ON_WM_INITDIALOG W OnSelchangeComboboxexON_IDC_COMBOBOXEX1:CBN_SELCHANGE W OnSelchangeMonthcalencON_IDC_MONTHCALENDAR1:MCN_SELCHANGE Description: Indicates a new combo box list item is selected			
	0	K Cancel	

Figure 20: Adding a handler for the first extended combo box, IDC_COMBOBOXEX1.

```
void CDialog1::OnSelchangeComboboxex1()
      {
         COMBOBOXEXITEM cbi;
         CString str ("dummy_string");
         CComboBoxEx * pCombo = (CComboBoxEx *)GetDlgItem(IDC_COMBOBOXEX1);
         int nSel = pCombo->GetCurSel();
         cbi.iItem = nSel;
         cbi.pszText = (LPTSTR)(LPCTSTR)str;
         cbi.mask = CBEIF_TEXT;
         cbi.cchTextMax = str.GetLength();
         pCombo->GetItem(&cbi);
         SetDlgItemText(IDC_STATIC7,str);
         return;
     }
void CDialog1::OnSelchangeComboboxex1()
{
    // TODO: Add your control notification handler code here
    COMBOBOXEXITEM cbi;
    CString str ("dummy_string");
    CComboBoxEx * pCombo = (CComboBoxEx *)GetDlgItem(IDC_COMBOBOXEX1);
    int nSel = pCombo->GetCurSel();
    cbi.iItem = nSel;
    cbi.pszText = (LPTSTR)(LPCTSTR)str;
    cbi.mask = CBEIF_TEXT
    cbi.cchTextMax = str.GetLength();
    pCombo->GetItem(&cbi);
    SetDlgItemText(IDC_STATIC7,str);
    return;
```

b

Listing 8.

The first thing you probably noticed is the use of the COMBOBOXEXITEM structure for the extended combo box instead of the plain integers used for items in an older combo box. Once the handler retrieves the item, it extracts the string and calls SetDlgItemText() to update the companion static control.

Add **Images** to the **Items** in the second extended combo box. The first extended combo box does not need any special programming. It is used to demonstrate how to implement a simple extended combo box very similar to the older, non-extended combo box. The second combo box requires a good bit of programming. First we created six bitmaps and eight icons that we need to add to the resources for the project, as shown in the following illustration.



Figure 21: A complete bitmap and icon set that you have to create.

Of course, you are free to use any bitmaps and icons.



?× Insert Resource Resource type: <u>N</u>ew Reference Accelerator Import... 🎬 Bitmap ⊕ <u>}</u> Cursor ⊕ 📴 Dialog Custom... 🧕 HTML Cancel 🛗 Icon 喜 Menu 🔤 String Table 😐 Toolbar 10 Version

Figure 22: Inserting new resource, a bitmap.





Figure 24: Inserting a new icon.



Figure 25: Editing the icon and modifying the properties.



Figure 26: A completed set of the bitmaps and icons for MYMFC25A project.

There are two ways to add our graphics to an extended combo box. The first method is to attach images to existing combo box items. Remember that we used the dialog editor to add the **Doremon**, **Tweety**, **Mack**, etc. items to the combo box. The second method is to add new items and specify their corresponding images at the time of addition. Before we start adding graphics to the extended combo box, let's create a public CImageList data member in the CDialog1 class named m_imageList. Be sure you add the data member to the header file (**Dialog1.h**) for the class.

🖃 📴 myi	👘 mymfc25A classes		
🗄 📲 CAboutDlg			
֥t	Dialog1		
÷ - t	<u>G</u> o to Definition		
֥ [Go To Dialog <u>E</u> ditor		
	Add Member Function		
÷	Add Member <u>V</u> ariable		
	Add Virtual Eunction		
	Add Windows Message Handler		
	References		
	👫 Derived Classes		
	🚪 <u>B</u> ase Classes		
	Add to Gallery		
	🗳 Ne <u>w</u> Folder		
	Group by Access		
	 Docking View 		
	Hide		
	Properties		

Figure 27: Adding a public CImageList data member to the CDialog1 class.

Add Member Variable	? 🗙
Variable <u>Type:</u>	OK
	Cancel
variable <u>N</u> ame:	

Figure 28: Entering the variable type and name.

Now we can add some of the bitmap images to the image list and then "attach" the images to the three items already in the extended combo box. Add the following code to your CDialog1's OnInitDialog() method to achieve this:

```
//Initialize the IDC_COMBOBOXEX2
CComboBoxEx* pCombo = (CComboBoxEx*) GetDlgItem(IDC_COMBOBOXEX2);
//First let's add images to the items there.
//We have six images in bitmaps to match to our strings:
//CImageList * pImageList = new CImageList();
m_imageList.Create(32,16,ILC_MASK,12,4);
```

```
CBitmap bitmap;
```

```
bitmap.LoadBitmap(IDB_BMBIRD);
m_imageList.Add(&bitmap, (COLORREF)0xFFFFFF);
bitmap.DeleteObject();
bitmap.LoadBitmap(IDB_BMBIRDSELECTED);
m_imageList.Add(&bitmap, (COLORREF)0xFFFFF);
bitmap.DeleteObject();
bitmap.LoadBitmap(IDB_BMDOG);
m_imageList.Add(&bitmap, (COLORREF)0xFFFFF);
bitmap.DeleteObject();
bitmap.LoadBitmap(IDB_BMDOGSELECTED);
m_imageList.Add(&bitmap, (COLORREF)0xFFFFF);
bitmap.DeleteObject();
bitmap.LoadBitmap(IDB_BMFISH);
m_imageList.Add(&bitmap, (COLORREF)0xFFFFFF);
bitmap.DeleteObject();
bitmap.LoadBitmap(IDB_BMFISHSELECTED);
m_imageList.Add(&bitmap, (COLORREF)0xFFFFFF);
bitmap.DeleteObject();
//Set the imagelist
pCombo->SetImageList(&m_imageList);
//Now attach the images to the items in the list.
COMBOBOXEXITEM
                 cbi;
cbi.mask = CBEIF_IMAGE|CBEIF_SELECTEDIMAGE|CBEIF_INDENT;
CString strTemp;
int nBitmapCount = 0;
for (int nCount = 0;nCount < 3;nCount++)</pre>
   cbi.iItem = nCount;
   cbi.pszText = (LPTSTR)(LPCTSTR)strTemp;
   cbi.cchTextMax = 256;
   pCombo->GetItem(&cbi);
    cbi.iImage = nBitmapCount++;
   cbi.iSelectedImage = nBitmapCount++;
   cbi.iIndent = (nCount & 0x03);
   pCombo->SetItem(&cbi);
```

}

```
// CDialog1 message handlers
BOOL CDialog1::OnInitDialog()
£
    CDialog::OnInitDialog();
    // TODO: Add extra initialization here
    m_ptrIPCtrl.SetAddress(0L);
        //Initialize the IDC_COMBOBOXEX2
    CComboBoxEx* pCombo = (CComboBoxEx*) GetDlgItem(IDC_COMBOBOXEX2);
    //First let's add images to the items there.
    //We have six images in bitmaps to match to our strings:
    //CImageList * pImageList = new CImageList();
    m_imageList.Create(32,16,ILC_MASK,12,4);
    CBitmap bitmap;
    bitmap.LoadBitmap(IDB_BMBIRD);
    m_imageList.Add(&bitmap, (COLORREF)0xFFFFFF);
    bitmap.DeleteObject();
    bitmap.LoadBitmap(IDB_BMBIRDSELECTED);
    m_imageList.Add(&bitmap, (COLORREF)0xFFFFFF);
    bitmap.DeleteObject();
    bitman LoadBitman(IDB BMDOG).
```

Listing 9.

First the extended combo box initialization code creates a pointer to the control using GetDlgItem(). Next it calls Create() to create memory for the images to be added and to initialize the image list. The next series of calls loads each bitmap, adds them to the image list, and then deletes the resource allocated in the load.

CComboBoxEx::SetImageList is called to associate the m_imageList with the extended combo box. Next a COMBOBOXEXITEM structure is initialized with a mask, and then the for loop iterates from 0 through 2, setting the selected and unselected images with each pass through the loop. The variable nBitmapCount increments through the image list to ensure that the correct image ID is put into the COMBOBOXEXITEM structure. The for loop makes a call to CComboBoxEx::GetItem to retrieve the COMBOBOXEXITEM structure for each item in the extended combo box. Then the loop sets up the images for the list item and finally calls CComboBoxEx::SetItem to put the modified COMBOBOXEXITEM structure back into the extended combo box and complete the association of images with the existing items in the list.

Add **Items** to the **Extended Combobox**. The other technique available for putting images into an extended combo box is to add them dynamically, as shown in the code added to OnInitDialog() below:

```
HICON hIcon[8];
int n;
//Now let's insert some color icons
hIcon[0] = AfxGetApp()->LoadIcon(IDI_WHITE);
hIcon[1] = AfxGetApp()->LoadIcon(IDI_BLACK);
hIcon[2] = AfxGetApp()->LoadIcon(IDI_RED);
hIcon[3] = AfxGetApp()->LoadIcon(IDI_BLUE);
hIcon[4] = AfxGetApp()->LoadIcon(IDI_YELLOW);
hIcon[5] = AfxGetApp()->LoadIcon(IDI_CYAN);
hlcon[6] = AfxGetApp()->LoadIcon(IDI_PURPLE);
hIcon[7] = AfxGetApp()->LoadIcon(IDI_GREEN);
for (n = 0; n < 8; n++) {
    m_imageList.Add(hIcon[n]);
static char* color[] = {"white", "black", "red",
                          "blue", "yellow", "cyan",
                          "purple", "green"};
cbi.mask = CBEIF_IMAGE CBEIF_TEXT CBEIF_OVERLAY CBEIF_SELECTEDIMAGE;
```

```
for (n = 0; n < 8; n++) {
          cbi.iItem = n;
          cbi.pszText = color[n];
          cbi.iImage = n+6; // 6 is the offset into the image list from
          cbi.iSelectedImage = n+6; // the first six items we added...
          cbi.iOverlay = n+6;
          int nItem = pCombo->InsertItem(&cbi);
          ASSERT(nItem == n);
      3
HICON hIcon[8];
int n;
//Now let's insert some color icons
hIcon[0] = AfxGetApp()->LoadIcon(IDI_WHITE);
hIcon[1] = AfxGetApp()->LoadIcon(IDI_BLACK);
hIcon[2] = AfxGetApp()->LoadIcon(IDI_RED);
hIcon[3] = AfxGetApp()->LoadIcon(IDI_BLUE);
hIcon[4] = AfxGetApp()->LoadIcon(IDI_YELLOW);
hIcon[5] = AfxGetApp()->LoadIcon(IDI_YELLOW);
hIcon[6] = AfxGetApp()->LoadIcon(IDI_PURPLE);
hIcon[7] = AfxGetApp()->LoadIcon(IDI_GREEN);
for (n = 0; n < 8; n++) {
     m_imageList.Add(hIcon[n]);
}
cbi.mask = CBEIF_IMAGE|CBEIF_TEXT|CBEIF_OVERLAY|
             CBEIF_SELECTEDIMAGE;
for (n = 0; n < 8; n++) {
    cbi.iItem = n;
     cbi.pszText = color[n];
     //6 is the offset into the image list from
    cbi.iImage = n+6;
     // the first six items we added...
     cbi.iSelectedImage = n+6;
    cbi.iOverlay = n+6;
     int nItem = pCombo->InsertItem(&cbi);
     ASSERT(nItem == n);
Ж
    return TRUE; // return TRUE unless you set t
    // EXCEPTION: OCX Property Pages sh
```

Listing 10.

The for loop fills out the COMBOBOXEXITEM structure and then calls CComboBoxEx::InsertItem with each item to add it to the list.

}

Add a handler for the second extended combo box. The second extended combo box handler is essentially the same as the first:

MFC ClassWizard	? 🛛
Message Maps Member Variables Automation ActiveX Events Class Info	1 .
Project: Class name: mymfc25A CDialog1 F:\\mymfc25A\Dialog1.h, F:\\mymfc25A\Dialog1.cpp	Add Class
Object [Ds: Messages: ID_VIEW_TOOLBAR IDC_COMBOBOXEX1 IDC_COMBOBOXEX1 IDC_COMBOBOXEX2 IDC_DATETIMEPICKER1 IDC_DATETIMEPICKER2 IDC_DATETIMEPICKER3 CBN_SELENDCANCEL IDC_DATETIMEPICKER4 CBN_SETFOCUS	Delete Function
Member functions: W OnFieldchangedlpaddressON_IDC_IPADDRESS1:IPN_FIELDCHANGED W OnInitDialog ON_WM_INITDIALOG W OnSelchangeComboboxe>ON_IDC_COMBOBOXEX1:CBN_SELCHANGE W OnSelchangeComboboxe>ON_IDC_COMBOBOXEX2:CBN_SELCHANGE W OnSelchangeComboboxe>ON_IDC_COMBOBOXEX2:CBN_SELCHANGE W OnSelchangeComboboxe>ON_IDC_MBOBOXEX2:CBN_SELCHANGE W OnSelchangeMonthcalencON_IDC_MONTHCALENDAR1:MCN_SELCHANGE Description: Indicates a new combo box list item is selected	
	K Cancel

Figure 29: Adding a handler for the second extended combo box, IDC_COMBOBOXEX2.

```
void CDialog1::OnSelchangeComboboxex2()
     {
         COMBOBOXEXITEM cbi;
         CString str ("dummy_string");
         CComboBoxEx * pCombo = (CComboBoxEx *)GetDlgItem(IDC_COMBOBOXEX2);
         int nSel = pCombo->GetCurSel();
         cbi.iItem = nSel;
         cbi.pszText = (LPTSTR)(LPCTSTR)str;
         cbi.mask = CBEIF_TEXT;
         cbi.cchTextMax = str.GetLength();
         pCombo->GetItem(&cbi);
         SetDlgItemText(IDC_STATIC8, str);
         return;
     }
void CDialog1::OnSelchangeComboboxex2()
{
    // TODO: Add your control notification handler code here
    COMBOBOXEXITEM cbi;
    CString str ("dummy_string");
    CComboBoxEx * pCombo = (CComboBoxEx *)GetDlgItem(IDC_COMBOBOXEX2);
    int nSel = pCombo->GetCurSel();
    cbi.iItem = nSel;
cbi.pszText = (LPTSTR)(LPCTSTR)str;
    cbi.mask = CBEIF_TEXT
    cbi.cchTextMax = str.GetLength();
    pCombo->GetItem(&cbi);
    SetDlgItemText(IDC_STATIC8,str);
    return;
```

Ð

Listing 11.

Connect the view and the dialog. Add code to the virtual OnDraw() function in **mymfc25AView.cpp**. The following code replaces the previous code:

```
void CMymfc25AView::OnDraw(CDC* pDC)
{
    pDC->TextOut(30, 30, "Press the left mouse button here.");
}
// CMymfc25AView drawing
void CMymfc25AView::OnDraw(CDC* pDC)
{
    // TODO: add draw code for native data here
    pDC->TextOut(30, 30, "Press the left mouse button here.");
}
```

Listing 12.

Use ClassWizard to add the OnLButtonDown() member function to the CMymfc25AView class. Edit the AppWizard-generated code as follows:

MFC ClassWizar	d			? 🛛
Message Maps	Member Variables	Automation ActiveX Events	Class Info	1
<u>P</u> roject:		Class <u>n</u> ame:		Add Class 🔻
mymfc25A	•	CMymfc25AView	-	Add Function
F:\\mymfc25A\m	ymfc25AView.h, F:\\m	ymfc25AView.cpp		
Object <u>I</u> Ds:		Messages:		Delete Function
CMymfc25AView ID_APP_ABOUT ID_APP_EXIT ID_EDIT_COPY		WM_KEYUP WM_KILLFOCUS WM_LBUTTONDBLCLK WM_LBUTTONDDUWN		Edit Code
ID_EDIT_CUT ID_EDIT_PASTE ID_EDIT_UNDO	~	WM_LBUTTONUP WM_MOUSEMOVE WM_MOUSEWHEEL		
Member <u>f</u> unctions:				
V OnBeginPrinti	ing		^	
V OnDraw	g			
W OnLButtonDo	own ON_WM_L	BUTTONDOWN		
V OnPreparePri	intina		×	
Description: Indicates when left mouse button is pressed				
			01	Cancel

Figure 30: Adding the OnLButtonDown() member function to the CMymfc25AView class to handle the left mouse click.

```
void CMymfc25AView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CDialog1 dlg;
    dlg.DoModal();
}
```

```
// CMymfc25AView message handlers
void CMymfc25AView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CDialog1 dlg;
    dlg.DoModal();
}
```

Listing 13.

Add a statement to include Dialog1.h in file mymfc25AView.cpp.

```
#include "stdafx.h"
#include "mymfc25A.h"
#include "mymfc25ADoc.h"
#include "mymfc25AView.h"
#include "Dialog1.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __F:
#endif
```

Listing 14.

Compile and run the program. Now you can experiment with the various IE4 common controls to see how they work and how you can apply them in your own applications.



Figure 31: MYMFC25A program output.

Dialog		
New Date/Time Controls—		IP Address Edit Control
Choose a date (short):	3/16/2006 • 03/16/2006	Choose your IP address: 172 16 160 111 172.16.160.111
Choose a date (long):	Tuesday , March 07, 2 - 03/07/2006	Extended Combo Box
Choose a date (short with allow short and spinners):	▼ 3/25/2006 • • 03/25/2006 •	Old fashion combo: Caroline
Choose a time:	4/25/2006 · 13:27:38	Cool new Imagelist support:
Choose your birtday:	03/08/2006	
Sun Mon 26 27 5 6 12 13 19 20 2 26 27 2 3	March, 2006 Image: Constraint of the second state of the sec	ncel

Figure 32: MYMFC25A program output, full of Internet Explorer 4 common controls.

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.