```
    hWnd = FindWindow (szAppName, NULL);
    if (hWnd) {
        SetForegroundWindow ((HWND)(((DWORD)hWnd) | 0x01));
        return 0;
    }
#endif
    // Register application main window class.
    wc.style = 0;                           // Window style
    wc.lpfnWndProc = MainWndProc;           // Callback function
    wc.cbClsExtra = 0;                      // Extra class data
    wc.cbWndExtra = 0;                      // Extra window data
    wc.hInstance = hInstance;               // Owner handle
    wc.hIcon = NULL,                        // Application icon
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);// Default cursor
    wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
    wc.lpszMenuName =  NULL;                // Menu name
    wc.lpszClassName = szAppName;           // Window class name

    if (RegisterClass (&wc) == 0) return 0;

    // Load the command bar common control class.
    icex.dwSize = sizeof (INITCOMMONCONTROLSEX);
    icex.dwICC = ICC_BAR_CLASSES;
    InitCommonControlsEx (&icex);

    // Save program instance handle in global variable.
    hInst = hInstance;

    // Create main window.
    hWnd = CreateWindow (szAppName, TEXT ("CmdBar Demo"), WS_VISIBLE,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL,
                hInstance, NULL);
    // Return fail code if window not created.
    if (!IsWindow (hWnd)) return 0;

    // Standard show and update calls
    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);
    return hWnd;
}
//------------------------------------------------------------------------
// TermInstance - Program cleanup
//
int TermInstance (HINSTANCE hInstance, int nDefRC) {
    return nDefRC;
}
//========================================================================
// Message handling procedures for MainWindow
//------------------------------------------------------------------------
// MainWndProc - Callback function for application window
//
LRESULT CALLBACK MainWndProc (HWND hWnd, UINT wMsg, WPARAM wParam,
                              LPARAM lParam) {
```

```
    int i;
    //
    // Search message list to see if we need to handle this
    // message. If in list, call procedure.
    //
    for (i = 0; i < dim(MainMessages); i++) {
        if (wMsg == MainMessages[i].Code)
            return (*MainMessages[i].Fxn)(hWnd, wMsg, wParam, lParam);
    }
    return DefWindowProc (hWnd, wMsg, wParam, lParam);
}
//---------------------------------------------------------------------
// DoCreateMain - Process WM_CREATE message for window.
//
LRESULT DoCreateMain (HWND hWnd, UINT wMsg, WPARAM wParam,
                      LPARAM lParam) {
    HWND hwndCB;

    // Create a minimal command bar that has only a menu and an
    // exit button.
    hwndCB = CommandBar_Create (hInst, hWnd, IDC_CMDBAR);

    // Insert the menu.
    CommandBar_InsertMenubar (hwndCB, hInst, ID_MENU, 0);

    // Add exit button to command bar.
    CommandBar_AddAdornments (hwndCB, 0, 0);
    return 0;
}
//---------------------------------------------------------------------
// DoSizeMain - Process WM_SIZE message for window.
//
LRESULT DoSizeMain (HWND hWnd, UINT wMsg, WPARAM wParam,
                    LPARAM lParam) {
   // This only needed if the window can be resized
    HWND hwndCB = GetDlgItem (hWnd, IDC_CMDBAR);
   // Tell the command bar to resize itself and reposition Close button.
   SendMessage(hwndCB, TB_AUTOSIZE, 0L, 0L);
   CommandBar_AlignAdornments(hwndCB);

    return 0;
}
//---------------------------------------------------------------------
// DoCommandMain - Process WM_COMMAND message for window.
//
LRESULT DoCommandMain (HWND hWnd, UINT wMsg, WPARAM wParam,
                       LPARAM lParam) {
    WORD idItem, wNotifyCode;
    HWND hwndCtl;
    INT  i;

    // Parse the parameters.
    idItem = (WORD) LOWORD (wParam);
    wNotifyCode = (WORD) HIWORD (wParam);
```

```
    hwndCtl = (HWND) lParam;

    // Call routine to handle control message.
    for (i = 0; i < dim(MainCommandItems); i++) {
        if (idItem == MainCommandItems[i].Code)
            return (*MainCommandItems[i].Fxn)(hWnd, idItem, hwndCtl,
                                              wNotifyCode);
    }
    return 0;
}
//------------------------------------------------------------------------
// DoNotifyMain - Process WM_NOTIFY message for window.
//
LRESULT DoNotifyMain (HWND hWnd, UINT wMsg, WPARAM wParam,
                      LPARAM lParam) {
    LPNMHDR pNotifyHeader;
    LPNMTOOLBAR pNotifyToolBar;
    RECT rect;
    TPMPARAMS tpm;
    HMENU hMenu;

    // Get pointer to notify message header.
    pNotifyHeader = (LPNMHDR)lParam;

    if (pNotifyHeader->code == TBN_DROPDOWN) {

        // Get pointer to toolbar notify structure.
        pNotifyToolBar = (LPNMTOOLBAR)lParam;

        if (pNotifyToolBar->iItem == IDC_DPSORT) {

            // Get the rectangle of the drop-down button.
            SendMessage (pNotifyHeader->hwndFrom, TB_GETRECT,
                         pNotifyToolBar->iItem, (LPARAM)&rect);

            // Convert rect to screen coordinates.  The rect is
            // considered here to be an array of 2 POINT structures.
            MapWindowPoints (pNotifyHeader->hwndFrom, HWND_DESKTOP,
                             (LPPOINT)&rect, 2);

            // Prevent the menu from covering the button.
            tpm.cbSize = sizeof (tpm);
            CopyRect (&tpm.rcExclude, &rect);

            hMenu = GetSubMenu (LoadMenu (hInst, TEXT ("popmenu")),0);
            TrackPopupMenuEx (hMenu, TPM_LEFTALIGN | TPM_VERTICAL,
                              rect.left, rect.bottom, hWnd, &tpm);
        }
    }
    return 0;
}
//------------------------------------------------------------------------
// DoDestroyMain - Process WM_DESTROY message for window.
//
```

```
LRESULT DoDestroyMain (HWND hWnd, UINT wMsg, WPARAM wParam,
                       LPARAM lParam) {
    PostQuitMessage (0);
    return 0;
}
//====================================================================
// Command handler routines
//--------------------------------------------------------------------
// DoMainCommandExit - Process Program Exit command.
//
LPARAM DoMainCommandExit (HWND hWnd, WORD idItem, HWND hwndCtl,
                          WORD wNotifyCode) {

    SendMessage (hWnd, WM_CLOSE, 0, 0);
    return 0;
}
//--------------------------------------------------------------------
// DoMainCommandViewStd - Displays a standard edit-centric command bar
//
LPARAM DoMainCommandVStd (HWND hWnd, WORD idItem, HWND hwndCtl,
                          WORD wNotifyCode) {
    HWND hwndCB;

    // If a command bar exists, kill it.
    if (hwndCB = GetDlgItem (hWnd, IDC_CMDBAR))
        CommandBar_Destroy (hwndCB);

    // Create a command bar.
    hwndCB = CommandBar_Create (hInst, hWnd, IDC_CMDBAR);
    // Insert a menu.
    CommandBar_InsertMenubar (hwndCB, hInst, ID_MENU, 0);

    // Insert buttons.
    CommandBar_AddBitmap (hwndCB, HINST_COMMCTRL, IDB_STD_SMALL_COLOR,
                          STD_BMPS, 0, 0);

    CommandBar_AddButtons (hwndCB, dim(tbCBStdBtns), tbCBStdBtns);

    // Add exit button to command bar.
    CommandBar_AddAdornments (hwndCB, 0, 0);
    return 0;
}
//--------------------------------------------------------------------
// DoMainCommandVView - Displays a standard edit-centric command bar
//
LPARAM DoMainCommandVView (HWND hWnd, WORD idItem, HWND hwndCtl,
                           WORD wNotifyCode) {
    INT i;
    HWND hwndCB;
    TCHAR szTmp[64];
    HBITMAP hBmp, hMask;
    HIMAGELIST hilDisabled, hilEnabled;

    // If a command bar exists, kill it.
```

```
    if (hwndCB = GetDlgItem (hWnd, IDC_CMDBAR))
        CommandBar_Destroy (hwndCB);
    // Create a command bar.
    hwndCB = CommandBar_Create (hInst, hWnd, IDC_CMDBAR);

    // Insert a menu.
    CommandBar_InsertMenubar (hwndCB, hInst, ID_MENU, 0);

    // Insert buttons, first add a bitmap and then the buttons.
    CommandBar_AddBitmap (hwndCB, HINST_COMMCTRL, IDB_VIEW_SMALL_COLOR,
                          VIEW_BMPS, 0, 0);

    // Load bitmaps for disabled image.
    hBmp = LoadBitmap (hInst, TEXT ("DisCross"));
    hMask = LoadBitmap (hInst, TEXT ("DisMask"));

    // Get the current image list and copy.
    hilEnabled = (HIMAGELIST)SendMessage (hwndCB, TB_GETIMAGELIST, 0, 0);
    hilDisabled = ImageList_Duplicate (hilEnabled);
    // Replace a button image with the disabled image.
    ImageList_Replace (hilDisabled, VIEW_LIST, hBmp, hMask);

    // Set disabled image list.
    SendMessage (hwndCB,  TB_SETDISABLEDIMAGELIST, 0,
                 (LPARAM)hilDisabled);

    // Add buttons to the command bar.
    CommandBar_AddButtons (hwndCB, dim(tbCBViewBtns), tbCBViewBtns);

    // Add tooltips to the command bar.
    CommandBar_AddToolTips (hwndCB, dim(pViewTips), pViewTips);

    // Add a combo box between the view icons and the sort icons.
    CommandBar_InsertComboBox (hwndCB, hInst, 75,
                               CBS_DROPDOWNLIST | WS_VSCROLL,
                               IDC_COMBO, 6);
    // Fill in combo box.
    for (i = 0; i < 10; i++) {
        wsprintf (szTmp, TEXT ("Item %d"), i);
        SendDlgItemMessage (hwndCB, IDC_COMBO, CB_INSERTSTRING, -1,
                            (LPARAM)szTmp);
    }
    SendDlgItemMessage (hwndCB, IDC_COMBO, CB_SETCURSEL, 0, 0);

    // Add exit button to command bar.
    CommandBar_AddAdornments (hwndCB, 0, 0);
    return 0;
}
//------------------------------------------------------------------------
// DoMainCommandVCombo - Displays a combination of file and edit buttons
//
LPARAM DoMainCommandVCombo (HWND hWnd, WORD idItem, HWND hwndCtl,
                            WORD wNotifyCode) {
    HWND hwndCB;
```

```
    // If a command bar exists, kill it.
    if (hwndCB = GetDlgItem (hWnd, IDC_CMDBAR))
        CommandBar_Destroy (hwndCB);

    // Create a command bar.
    hwndCB = CommandBar_Create (hInst, hWnd, IDC_CMDBAR);

    // Insert a menu.
    CommandBar_InsertMenubar (hwndCB, hInst, ID_MENU, 0);
    // Add two bitmap lists plus custom bmp for drop-down button.
    CommandBar_AddBitmap (hwndCB, HINST_COMMCTRL, IDB_STD_SMALL_COLOR,
                          STD_BMPS, 0, 0);
    CommandBar_AddBitmap (hwndCB, HINST_COMMCTRL, IDB_VIEW_SMALL_COLOR,
                          VIEW_BMPS, 0, 0);
    CommandBar_AddBitmap (hwndCB, NULL,
                          (int)LoadBitmap (hInst, TEXT ("SortDropBtn")),
                          1, 0, 0);

    CommandBar_AddButtons (hwndCB, dim(tbCBCmboBtns), tbCBCmboBtns);

    // Add exit button to command bar.
    CommandBar_AddAdornments (hwndCB, 0, 0);
    return 0;
}
```

CmdBar 範例建立的 3 個命令欄示範了命令欄控制項的不同功能。在 DoMain CommandVStd 常式中建立了第一個命令欄，這個命令欄包括功能表和一系列的按鈕。這個命令欄的按鈕結構是在 CmdBar.cpp 頂部的 tbCBStdBtns 陣列中定義的。

DoMainCommandVView 常式中建立了第 2 個命令欄，它其中包括兩組由複合式對話盒分隔出來的 CheckGroup 按鈕。該命令欄也示範了使用禁用的按鈕上的圖片來做分隔的方法。命令欄上的第 3 個按鈕，列表視圖按鈕被禁用了。這個禁用的列表視圖中的圖像被一個看起來很像 X 的點陣圖取代了。

DoMainCommandVCombo 常式建立了第 3 個命令欄，它示範了如何在下拉式清單按鍵上應用內建的標準圖像、內建的視圖圖像和自定義點陣圖。該命令欄也示範了如何引用含有多個點陣圖的圖像列表中的某個圖像。程式使用 DoNotifyMain 常式來處理下拉式清單按鍵訊息，當收到 TBN_DROPDOWN 通知後，實現了載入和顯示彈出功能表。

5.2.2 其他功能表控制項

近年來，微軟在作業系統下增加了兩個 “功能表控制項”，用來支援下拉功能表。所增加的第一個控制項被稱為命令帶（Command Band）。命令帶控制項就是 rebar 控制項，不過每個 rebar 元素都默認為命令欄。rebar 控制項是控制項的容器，使用

者可在應用程式視窗上自由拖動它。命令帶其實就是在 rebar 中放置的命令欄，學習如何對命令帶控制項程式設計時最需要做的就是了解如何對命令欄程式設計。

很多年前功能表欄（menu bar）控制項就已經在 Pocket PC 中出現了。正如上文所述，功能表欄不像命令欄一樣在應用程式的客戶視窗頂部，而是在桌面視窗的底部。對於程式設計師來說，功能表欄不僅僅是與命令欄看起來不一樣，功能表欄也有其獨特的程式設計介面。功能表欄實際上是一種最上層表單，而不是某個視窗建立的子視窗。在 Windows CE 上的功能表欄就是為了支援某些應用程式與 Windows Mobile 相相容的需求。然而，由於 Windows CE 上的功能表欄和 Windows Mobile 設備上的功能表欄有著顯著的不同，因此相容性也就僅僅存在於程式設計層面而不是使用者介面層面。因為功能表欄的功能比命令欄或命令帶控制項都要弱，只有在系統需要與 Windows Mobile 系統二進位相容時才應該使用功能表欄。

本章節的其餘部分介紹了其他控制項的一些重點。與 Windows Vista 下的同名控制項相比，兩者之間的功能類似但是在 Windows CE 上的實現可能會弱一些。接下來將我認為在寫 Windows CE 程式的時候會用到的控制項上多費一些筆墨。我們從行事曆控制項和時間日期選擇器控制項開始學起。這些控制項在 Windows CE 擅長的個人資訊管理（PIM）類應用程式上運用廣泛。我也會介紹列表視圖控制項，主要關注那些便於 Windows CE 開發者開發使用的特點。最後也會簡要地去瞭解剩餘的通用控制項。

5.2.3 行事曆控制項

行事曆空間提供給大家一種便捷行事曆視圖，使用者可以查找從採用英國格裏高利（Gregorian）曆法的 1752 年 9 月開始的任意的月、周或天。這個控制項可以盡可能多地顯示可以納入控制項的月份。行事曆中的某一天可以高亮顯示以表示有約會。周曆則顯示了這年中的當前一周。使用者可以通過在控制項中點擊月份或年份來切換想要查看的月或年。

在使用行事曆控制項之前，首先要呼叫 InitCommonControlsEx 並用 ICC_DATE_CLASSES 旗標作為參數來初始化通用控制項程式庫。然後就可以通過呼叫 CreateWindow 函式並把 MONTHCAL_CLASS 旗標作為視窗類參數來建立控制項。行事曆控制項的 style 旗標如下所示：

- **MCS_MULTISELECT**：這個控制項允許多個天被同時選中。
- **MCS_NOTODAY**：控制項將不在行事曆下顯示當前日期。
- **MCS_NOTODAYCIRCLE**：控制項不會把當前日期圈起來。

- **MCS_WEEKNUMBERS**：在控制項的每行左邊顯示周數(1~52)。
- **MCS_DAYSTATE**：控制項發送通知訊息給父視窗將當前月份的某些日期顯示為粗體。使用這個樣式可以表明某些日期存在約會或事件安排。

初始化控制項

除了上文提到的樣式之外，還可以使用一系列的訊息或者它們對應的包裹的巨集來設定行事曆控制項。也可以使用 MCM_SETFIRSTDAYOFWEEK 訊息來設定一周中不同的開始日期。也可以使用 MCM_SETRANGE 訊息在控制項中顯示指定範圍內的一段日期。也能設定選定日期，讓使用者選擇單獨的日期或在一定限度的日期範圍內選擇任意幾個日期。設定 MCS_MULTISELECT 樣式定義了使用者可以選擇單個/多個日期。如果你設定了這個樣式，則還可以使用 MCM_SETMAXSELCOUNT 訊息設定任意選擇日期的最大數目。

可以使用 MCM_SETCOLOR 訊息來設定控制項的背景或文字顏色。這個訊息可以給控制項的不同區域分別設定顏色，包括行事曆文字、行事曆背景列標題和標題背景，還有在當前月之前和之後的幾天日期的文字顏色。這個訊息包含一個用來指明要設定控制項哪部分顏色的旗標，還有一個用 COLORREF 來表示的顏色值。

行事曆控制項被設計為按照整月為基數顯示日期，這就是說，即使這個控制項足夠顯示一個半月的日期，它依然僅僅在控制項的中心顯示一個月的日期。也可以使用 MCM_GETMINREQRECT 訊息來計算能夠顯示一個月日期所需的最小矩形。由於在發送 MCM_GETMINTREQRECT 之前行事曆控制項必須已經被成功建立，因此適當調整控制項的尺寸是一個循序漸進的過程。也就是說你必須先建立這個控制項，然後發送 MCM_GETMINTREQRECT 訊息，再根據這個訊息返回的資料來重新調整控制項的大小。

行事曆的通知訊息

行事曆控制項發送給父視窗的只有 3 個通知訊息。其中最重要的是 MCN_GETDAYSTATE 通知。當控制項需要知道把哪些日期顯示為粗體時，就會發送這個通知。這是通過向父視窗查詢一系列用 MONTHDAYSTATE 變數包含的位元欄位值來實現的。這個 MONTHDAYSTATE 變數也只不過是用 32 位元（從 1 到 31 的變數）中的每一個位元（bit）來表示一個月的每一天。

當控制項要顯示一個月的日期，就會發送 MCN_GETDAYSTATE 通知並附帶一個指向 NMDAYSTATE 結構的指標。其中 NMDAYSTATE 結構定義如下：

```
typedef struct {
    NMHDR nmhdr;
    SYSTEMTIME stStart;
    int cDayState;
    LPMONTHDAYSTATE prgDayState;
} NMDAYSTATE;
```

nmhdr 參數與每個 WM_NOTIFY 訊息中的 NMHDR 結構一樣簡單。stStart 欄位表示控制項所請求資料的起始日期。在所有版本的 Windows 中，SYSTEMTIME 的資料格式是一致的。SYSTEMTIME 具體細節描述如下：

```
typedef struct {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

對這個通知而言，只有 wYear、wMonth、wDay 欄位是比較重要的。

cDatState 欄位中包含了 MONTHDAYSTATE 陣列中的元素個數。即使行事曆只顯示一個月的日期，它也會請求本月的前幾天或後幾天的資訊。因為這幾天通常會被用來在行事曆的頂部或底部填充空白。

當使用者改變在控制項中的選定日期時，行事曆控制項會發送一個 MCN_SELCHANGE 通知，並附帶一個 NMSELCHANGE 結構，這個結構包含用反白顯示新的起始和結束日期。當使用者雙擊某個日期時就會發送 MCN_SELCHANGE 通知。也會附帶與選定日期改變通知相同的 NMSELCHANGE 結構。在第 6 章“對話框和屬性表”中的 DlgDemo 範例程式就在屬性表例子中使用了一個行事曆控制項。這簡單的幾分鐘範例就充分示範了行事曆控制項發送的通知。

5.2.4 時間日期選擇器控制項

時間日期選擇器控制項（Date and Time Picker Control）看起來很簡單但卻非常有用，對於使用者要指定日期的所有應用程式來說，時間日期選擇器控制項都是很好的工具。所有需要對字串進行分析、驗證並最終轉化成系統日期或時間的程式設計師都十分喜歡這個控制項。

當使用者要選擇日期時，這個控制項就像一個複合式對話盒，在編輯區域的右邊有一個下拉箭頭的按鈕。點擊這個箭頭會顯示當前這個月的行事曆控制項。選擇這個月中的一天後，行事曆控制項就會消失，而選定的日期就會顯示在這個時間日期選擇器控制項中。當你想設定查詢某一個日期時，這個時間日期選擇器控制項就像一個文字框，在控制項的右邊有一個微調按鈕。

時間日期選擇器控制項有 3 個默認格式：兩個用於顯示日期，一個用於顯示時間。這個控制項也允許你提供一個轉換過的字串使使用者能完全地使用控制項中的區域，甚至允許你在其中插入應用程式定義區域。

建立時間日期選擇器控制項

在建立時間日期選擇器控制項之前要先初始化通用控制項程式庫。如果使用 InitCommonControlsEx 函式，就必須把 ICC_DATE_CLASSES 旗標作為參數。然後可以使用 DATETIMEPICK_CLASS 作為視窗類，呼叫 CreateWindow 函式來建立控制項。日期時間控制項定義了以下樣式：

- **DTS_LONGDATEFOTMAT**：控制項使用長日期格式來顯示日期，例如：Friday,September 14, 2007。實際的長日期格式是在系統登錄表中定義的。
- **DTS_SHORTDATEFORMAT**：控制項使用短日期格式來顯示日期，例如：9/14/07。實際的短日期格式是在系統登錄表中定義的。
- **DTS_TIMEFORMAT**：控制項裏顯示了如 5:50:28 PM，這種格式的時間，這種時間格式也是在系統登錄表中定義的。
- **DTS_SHOWNONE**：控制項有一個核取方塊來指示日期的合法性。
- **DTS_UPDOWNv**：在行事曆控制項的日期視圖中用一個上下微調控件替代了下拉式清單按鍵。
- **DTS_APPCANPARSE**：允許使用者在控制項中直接鍵入文字，當輸入完成後控制項就發送一個 DTN_USERSTRING 通知。

前 3 種樣式只是指定預設的格式字串。這些格式是由登錄表中的地域設定決定。如果使用者在控制板裏選擇了不同的地域設定，這些格式就會發生相應的變化，當格式轉變時需通知給時間日期選擇器控制項。系統發送一個 WM_SETTINGCHANGE 訊息來把格式的轉變通知最上層表單。當應用程式使用時間日期選擇器控制項和預設字體時，應用程式應該把 WM_SETTINGCHANGE 訊息轉發給時間日期選擇器控制項，這樣控制項就可以把日期時間格式更改為新的區域設定中的格式。而 DTS_APPCANPARSE 樣式允許使用者直接在控制項內編輯文字。如果沒有為控制

項設定這個樣式，控制項唯一允許的按鍵是游標移動和數位鍵。例如對於月輸入，如果使用者按下數位鍵 6，月份會自動變成 June。當設定了 DTS_APPCANPARSE 樣式之後，使用者可以直接在編輯區域內輸入任何字元。當輸入完畢之後，控制項會發送 DTN_USERSTRING 通知到它的父視窗，借助這種手段可以對使用者輸入的文字進行驗證。

使用者自定義格式

要使用自定義的日期時間顯示格式，你所要做的僅僅是建立一個格式字串，然後把這個字元串通過 DTM_SETFORMAT 訊息發送給控制項。格式字串可以由以下程式碼中任意形式的組合來構成：

```
String          Description
fragment

"d"      One- or two-digit day.
"dd"     Two-digit day. Single digits have a leading zero.
"ddd"    The three-character weekday abbreviation. As in Sun, Mon...
"dddd"   The full weekday name.

"h"      One- or two-digit hour (12-hour format).
"hh"     Two-digit hour (12-hour format). Single digits have a leading zero.
"H"      One- or two-digit hour (24-hour format).
"HH"     Two-digit hour (24-hour format). Single digits have a leading zero.

"m"      One- or two-digit minute.
"mm"     Two-digit minute. Single digits have a leading zero.

"M"      One- or two-digit month.
"MM"     Two-digit month. Single digits have a leading zero.

"MMM"    Three-character month abbreviation.
"MMMM"   Full month name.
"t"      The one-letter AM/PM abbreviation. As in A or P.
"tt"     The two-letter AM/PM abbreviation. As in AM or PM.

"X"      Specifies a callback field that must be parsed by the application.

"y"      One-digit year. As in 1 for 2001.
"yy"     Two-digit year. As in 01 for 2001.
"yyy"    Full four-digit year. As in 2001.
```

可以通過使用單引號將文字字串引起來，並把它們包含在格式字串中。例如，顯示字串 “Today is :Saturday , December 2 , 2006” 則格式字串應寫成如下格式：

```
'Today is: 'dddd', 'MMMM' 'd', 'yyy
```

單引號引起來的文字字串都不會被解析，例如上面例子中的 “Today is” 還有所有分隔符號，例如空格和逗號。通過一系列 “X” 字元標識的回調欄位，為應用程式在設定日期的顯示格式上提供了極大的靈活性。當控制項在格式字串中檢測到 X

欄位時，控制項會向它的所有者發送一系列的通知訊息來詢問在這個 X 欄位中要顯示什麼。一個格式字串可以包含任意數量的 X 欄位。例如，下面的格式字串有兩個 X 欄位：

```
'Today 'XX' is: ' dddd', 'MMMM' 'd', 'yyy' and is 'XXX' birthday'
```

X 欄位中的 X 字元的個數，僅僅是用來區分應用程式不同的自定義欄位，而並不是指示在這個欄位中應該顯示的字元的確切個數。當控制項發送詢問關於 X 欄位的資訊的通知時，其中會包括一個指向 X 字串的指標。這樣應用程式就可以知道當前處理的是哪一個 X 欄位。當時間日期選擇器控制項需要顯示一個應用程式自定義的 X 欄位時，它會發送兩個通知：DTN_FORMATQUERY 和 DTN_FORMAT。發送 DTN_FORMATQUERY 通知用來獲得要顯示的文字的最大長度。發送 DTN_FORMAT 通知可以得到 X 欄位的實際文字。當使用者反白選擇了一個應用程式自定義的欄位並按下某按鍵時，會發送第 3 個通知 DTN_WMKEYDOWN。這時，應用程式負責決定兩件事：一個是決定哪個鍵是有效的，另一個是決定當按下了一個有效鍵時要修改日期。

5.2.5 清單檢視控制項

清單檢視控制項（List View）應該說是最複雜的通用控制項。它能以大圖示、小圖示、列表和報告 4 種模式中的一種來顯示一系列項。Windows CE 版本的清單檢視控制項支援很多但並不是所有隨 IE 4.0 一起發佈的通用控制項的功能。其中一些庫函式在記憶體受限的 Windows CE 環境下十分有用。這些特性包括管理幾乎任何大小的虛擬列表的能力，也包括含有圖像並且可以通過拖拽重新安排的標題行，也包括縮進某個項的能力，還包括新的報告模式。清單檢視控制項也支援使用者自繪的介面，這為改變控制項的外觀提供了一個相當簡單的方法。有兩種方法可以註冊清單檢視控制項，一種是通過呼叫 InitCommonControls 函式，另一種是通過把 ICC_LISTVIEW_CLASSES 旗標作為參數呼叫 InitCommonControls（**譯者註**：應該是 InitCommonControlsEx 函式，可能是作者的筆誤）函式。然後就可以通過呼叫 CreateWindow 函式並把 WC_LISTVIEW 作為視窗類參數來建立清單檢視控制項。Windows CE 版本的清單檢視控制項支援所有其他版本 Windows 下該控制項支援的控制項樣式，包括用來標識該控制項為虛擬清單檢視控制項的 LVS_OWNERDATA 樣式。

報告模式支援的樣式

除了用來建立列表視圖的標準列表視圖樣式以外，清單檢視控制項還支援一些其他的擴展樣式。不幸的是，這些擴展樣式並不能直接在 CreateWindowEX 函式的擴展

樣式參數中直接使用。但是，通過使用 LVM_GETEXTENDEDLISTVIEWSTYLE 和 LVM_SETEXTENDEDLISTVIEWSTYLE 這兩個訊息，可以得到或設定這些擴展樣式。Windows CE 支援的擴展樣式如下所示：

- **LVS_EX_CHECKBOXES**：控制項在每個項的邊上放一個可打勾的核取方塊。
- **LVS_EX_HEADERDRAGDROP**：控制項允許通過拖拽來重新安排標題行。
- **LVS_EX_GRIDLINES**：在報告模式下控制項會在每項邊上繪製網格邊線。
- **LVS_EX_SUBITEMIMAGES**：在報告模式下控制項在子項的列中顯示圖像。
- **LVS_EX_FULLROWSELECT**：在報告模式下當某項被選中後，控制項會反白顯示整行。
- **LVS_EX_ONECLICKACTIVATE**：只需單擊而不是雙擊，控制項就可以啟動該項。

除了 LVS_EX_CHECKBOXES 和 LVS_EX_ONECLICKACTIVATE 這兩個擴展樣式可以在所有顯示模式下使用，以上所有新的樣式只有在報告模式下才可在列表視圖中使用。這些功能使清單檢視控制項在顯示大量資料列表時表現得十分出色。

需要注意的是 Windows CE 中的清單檢視控制項並不支援一些通用控制項程式庫版本所支援的其他擴展樣式，例如，LVS_EX_INFOTIP、LVS_EX_ONECLICKACTIVATE、LVS_EX_TWOCLICKACTIVATE、LVS_EX_TRACKSELECT、LVS_EX_REGIONAL 和 LVS_EX_FLATSB。

虛擬清單檢視

清單檢視控制項的虛擬清單檢視模式對 Windows CE 設備有很大的幫助。在這個模式下，清單檢視只關注控制項的選擇和焦點狀態。應用程式負責來維護控制項中有關項的所有其他資料。這個模式非常有用的原因有如下兩點：首先，虛擬清單檢視的效率很高。控制項的初始化幾乎是瞬間完成的，因為你所要做的事就是設定控制項中一共有幾項。清單檢視控制項還會給你提示，告訴你它下一步將要尋找和顯示的項目。這樣應用程式就可以在記憶體中緩衝區必要的資料並且把其餘不需要的資料留在資料庫或文件中。如果沒有虛擬清單檢視，當應用程式初始化清單檢視時，就要把整個資料庫或者大量項的列表一次性載入清單檢視。有了虛擬清單檢視，應用程式一次只需要載入控制項所需要顯示的部分資料就可以了。虛擬清單檢視的第二個優點是節省記憶體空間。因為虛擬清單檢視控制項只需要為每個項維護很少的資訊，所以控制項並不需要在記憶體中為應用程式中的資料維護一個巨大的陣列。通過使用上面提到的虛擬清單檢視的緩衝提示原理，應用程式可以管理並決定把什

麼資料放入記憶體中。虛擬清單檢視也有一些限制。當控制項建立完成後，無法動態地設定和清除虛擬清單檢視的 LVS_OWNERDATA 樣式。且在大圖示和小圖示模式下，並不支援虛擬清單檢視的拖拽功能。虛擬清單檢視初始化時預設擁有 LVS_AUTOARRANGE 樣式，而且並不支援 LVM_SETITEM POSITION 訊息。另外，LVS_SORTASCENDING 和 LVS_SORTDESCENDING 這兩個排序樣式，虛擬清單檢視也不支援。即便如此，用虛擬清單檢視儲存大量列表項依然易如反掌。

要實現虛擬清單檢視，應用程式要用 LVS_OWNERDATA 樣式來建立一個清單檢視控制項，並且還要處理 LVN_GEETDISPINFO、IVN_ODCACHEHINT 和 LVN_ODFINDITEM 這 3 個訊息通知。LVN_GETDISPINFO 這個通知對於以前用過清單檢視控制項程式設計的人來說應該十分熟悉。每當清單檢視控制項需要查詢將要顯示的項的一些資訊時，就會發送這個通知。在虛擬清單檢視中，它的用法也是差不多的，但是不同的是，控制項發送這個通知用來獲得每個項目的所有資訊。

虛擬清單檢視透過使用 LVN_ODCACHEHINT 通知來告訴你它需要什麼樣的資料項目。這個通知會發送控制項下一步要用到的起始和結尾項索引。應用程式可以利用這些索引把控制項希望用的項載入緩衝區，以便快速存取。提示的一般是控制項中將要顯示的項。在控制項中，因為在不同的視圖下一屏顯示的項數可能不同，所以讓控制項提醒比讓應用程式猜測控制項將來會使用哪些項更有效率。因為控制項經常需要請求位於第一頁和最後一頁的項的資訊，所以最好能把第一頁和最後一頁上的項單獨緩衝區，這樣對它們的頻繁存取就不會把主緩衝區中稍後要用到的資料清除。管理虛擬清單檢視用到的最後一個必要的通知是 LVN_ODFINDITEM。當控制項需要找出一個項用來回應一個按鍵動作或者回應一個 LVM_FINDITEM 訊息時，它會發送一個 LVN_ODFINDITEM 通知。

5.2.6 CapEdit 控制項

CapEdit 控制項是一個編輯框，它可以把控制項中每個單詞的第一個字母以大寫形式顯示出來。當需要輸入合適的名稱，但是又因缺少鍵盤設備按 Shift 鍵十分不方便時，把這個控制項作為編輯控制項來說是一個很棒的選擇。

要建立 CapEdit 控制項，只要用 WC_CAPEDIT 視窗類作參數來建立一個視窗即可。因為 CapEdit 以編輯控制項的視窗程序來實現它的基礎功能，所以，你會發現當向它發送標準的編輯控制項訊息時，它會如同編輯控制項一樣去回應這個訊息。對於這個控制項來說，唯一特別的一個訊息是 CEM_UPCASEALLWORDS。當這個訊息的 wParam 參數不是 0 時，控制項會把所有單詞的首字母大寫。而當這個訊息的 wParam 參數是 0 時，控制項只會把控制項中第一個單詞的首字母大寫。

5.3 其他一些通用控制項

Windows CE 支援一系列在其他桌面 Windows 版本中支援的通用控制項。Windows CE 支援這些控制項的大多數功能，部分功能由於 Windows CE 的局限而無法實現。下面會對這些所支援的其他一些控制項做一個簡要的描述：

狀態欄（Status Bar）控制項

狀態欄控制項是沒有任何改變地從桌面 windows 版本中移植過來的。通用使用者介面指南並不建議在小型螢幕設備上使用這個控制項，但是它在大型的顯示設備上是非常有用的。

Tab 控制項

windows CE 幾乎支援所有的標準 Tab 控制項的特性。Tab 控制項不支援的特性包括：在滑鼠指標移動到 Tab 控制項上時，使其反白顯示的 TCS_HOTTRACK 樣式和 TCS_EX_REGISTERDROP 擴展樣式。

Trackbar 控制項

Trackbar 可以與另外一個控制項結成“兄弟”，這樣另一個控制項的值就可以根據 Trackbar 的動作來自動地更新。Trackbar 還支援自繪，可以單獨地繪製滑動條（channel）、滑塊(thumb)和刻度標記（tick mark）。

進度（Progress Bar）控制項

進度控制項支援水平和垂直的進度顯示及 32 位元的範圍。這個控制項還支援新的平滑的進度顯示。它摒棄了一塊一塊地移動進度指示符的效果。

上下(Up-Down)控制項

Windows CE 中的上下控制項只支援與編輯框和列表框控制項結為“兄弟”控制項。

工具欄（Toolbar）控制項

Windows CE 工具欄所支援的工具欄提示功能同這個控制項的桌面版本所支援的工具欄提示功能有所不同。在 Windows CE 下為工具欄增加工具欄提示的方法和你為命令欄增加提示的方法一樣，就是傳遞一個已經分配好的指向字串陣列的指標。工具欄和命令欄一樣支援透明和平滑的樣式。