第10章 控 件 窗 口

控件窗口是允许用户与应用程序进行交互的按钮、列表框和滚动条。当一个对话框被创建时，用对话框编辑器定义的控件窗口也被创建，然而一些控件窗口可能要求在运行时填充它们，如组合框。此外，一些控件窗口不能在资源模板中被创建。

有关可用的控件窗口的讨论，以及使用它们的例子参见附录 A，本章中的例子讨论怎样动态地创建和填充控件窗口。

例46　在任意位置创建一个控件窗口　讨论怎样在任意位置创建一个控件窗口，这里强调在“任意位置”。

例47　用子分类定制一个通用控件窗口　讨论通过使用子分类让类控制一个控件窗口的方法，子分类在第三章中已有详细描述。

例48　用超分类定制一个通用控件窗口　讨论通过使用超分类让类控制一个控件窗口的方法，超分类在第3章中也有详细描述。

例49　在按钮上放置位图　放弃按钮上常用的文本，而使用位图图像。

例50　动态填充一个组合框　讨论一种显示最新信息的方法，通过用户打开组合框时填充它来实现。

例51　排序一个列表控件　讨论用户单击一个列表控件的标题时，怎样通过调整该标题所在列作出反应。

例52　分割线控件　最后将讨论怎样在一个对话框中创建一条蚀刻线，该线不用自己绘制。

10.1 例46 在任意位置创建一个控件窗口

目标

在任意位置创建一个控件窗口(见图10-1)。

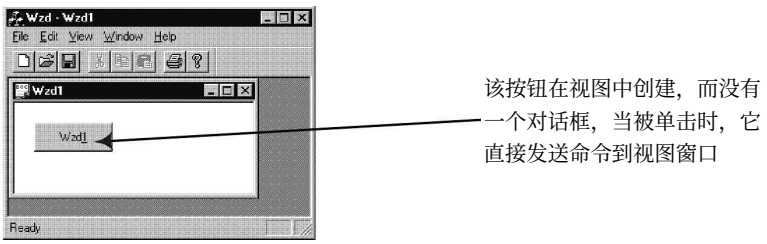

图10-1 把一个按钮控件放置到一个视图中或任意位置

策略

每个控件窗口，如一个按钮或一个编辑框，只是一个特殊的子窗口。它可以在用户界面的任意位置被打开，也可以传送到任何其他窗口。通常，打开一个对话框时， MFC处理打开控件窗口的繁琐工作。然而，可以用两个步骤手工打开控件窗口，首先，通过创建一个 MFC

控件类的实例，如CButton，封装该控件；第二步，通过调用那个类的 Create ( )成员函数，真正打开控件窗口。

步骤

1. 创建MFC控件类的一个实例

确定要创建的控件类型(如按钮、编辑框等)，然后确定封装该控件的MFC类(如CButton封装一个按钮控件窗口)，有关例子参见附录 A。然后把那个类插入到计划用来打开该控件的类中。本例中把CButton类插入到一个对话类中。

```
private:
   CButton    m_ctrlButton;
```

2. 创建控件窗口

1) 用ClassWizard，为已插入MFC控件类的对话框添加一个 WM_CREATE消息处理函数，或者一个WM_INITDIALOG消息处理函数。在 OnCreate ( )消息处理函数中，调用控件类的 Create ( )成员函数，并赋予它风格、扩展风格、标题 (如果有的话)、物主窗口、控件ID、大小和位置。创建一个控件需要提供的项可以随控件类的不同而改变，风格也改变，参见附录 A。本例创建按钮控件窗口。

```
CRect rect(20,20,100,50);
m_ctrlButton.Create("Wzd&1", WS_CHILD|WS_VISIBLE, rect, this,
   IDC_WZD_CONTROL);
```

2) 当一个对话框创建一个对话框模板中的控件时，它的任务不仅仅是创建该控件窗口，例如一个对话框还要设置每个控件窗口的字体。这里，为按钮控件设置字体，该按钮控件具有3-D窗口风格。

```
CFont *pFont=CFont::FromHandle((
   HFONT)::GetStockObject(ANSI_VAR_FONT));
m_ctrlButton.SetFont(pFont);
```

说明

- 前面已讨论创建任何一个控件窗口需要分两步进行：第一，创建 MFC控件类的一个实例；第二，调用它的Create ( )成员函数以创建真正的窗口。删除一个 MFC类和控件窗口可能比较复杂。如果 MFC类的实例被销毁，它将自动销毁它的控件窗口。然而，如果控件窗口先被销毁，则操作系统不知道是 MFC控件窗口类创建了它，这需要手工销毁该MFC控件类。在这种情况下，问题也不大，因为 MFC类被插入到另一个类中，当对话框被关闭时，该类会自动销毁它。参见第 1章有关该主题的详细内容。
- 有关用ClassWizard添加一个消息处理函数的例子参见例 13。

CD说明

在CD上执行该工程时，可以看到一个按钮控件显示在视图的中央。

10.2 例47 用子分类定制一个通用控件窗口

目标

通过子分类，添加功能到一个已有的通用控件。

策略

像按钮和编辑框一样，控件窗口通常用它们具有的系统提供的窗口过程维护它们，即使用MFC控件类创建一个控件，如CButton，也由系统提供的功能处理那个控件窗口的所有消息。子分类是一个过程，它通过在窗口对象中，用窗口过程的地址替换原有的窗口对象地址，允许切入控件窗口和它的原始窗口过程间的消息流。如果一个需要用原始窗口过程处理的消息到来，诸如绘制该控件，则把它传递到原始窗口过程地址。否则，可以用新的令人兴奋的方式处理消息，还好，MFC用CWnd::SubclassWindow ( )减少子分类一个窗口的许多繁琐工作。如果在对话框中为一个控件创建一个控件成员变量，甚至可以不用 SubclassWindow ( ) —— 当对话框进行数据交换时，它可以被自动调用。

要子分类控件窗口，首先创建一个 MFC控件类的派生类，然后直接或间接地通过DoDataExchange ( )，用SubclassWindow ( )把派生类的消息映像插入一个控件窗口和它的原始窗口过程间的消息流中。现在可以在该派生类中方便安全地处理消息，传送它们或者处理其中的一部分并传送其余消息。有关该主题的详细内容参见第 3章。

步骤

1. 创建一个从MFC控件类派生的新控件类

1) 用ClassWizard创建一个从诸多控件类中派生的新类，本例中，从 CEdit派生CWzdEdit，从CComboBox派生CWzdComboBox。

2) 用ClassWizard为想要在控件窗口和它的原始窗口过程间截获的消息添加一个消息处理函数。

如果需要子分类的控件在一个对话框中，可以用 ClassWizard添加一个Control类型的成员变量到对话框类中，它可以自动地子分类一个控件窗口，这可以在下面看到。

2. 用ClassWizard子分类一个控件

1) 打开ClassWizard并选择合适的对话框类，然后选择 Member Variables标签页，找到并选择需要子分类的控件的控件 ID，从可用的成员变量类型列表中选择 Control。本例中，添加了编辑和组合框控件成员变量，它使得 ClassWizard添加下面的代码行到那个类的 .h文件中。

```
//{{AFX_DATA(CWzdDialog)
enum {IDD = IDD_WZD_DIALOG};
CEdit m_ctrlEdit;
CComboBox    m_ctrlComboBox;
//}}AFX_DATA
```

2) 把这些定义放到{{ }}括弧下面，以便ClassWizard不能改变它，以及把这些标准MFC类名改变为新的派生的类名。

```
//{{AFX_DATA(CWzdDialog)
enum {IDD = IDD_WZD_DIALOG};
//}}AFX_DATA
CWzdEdit       m_ctrlEdit;                <<<<
CWzdComboBox   m_ctrlComboBox;    <<<<
```

3) 现在，新类可以完全访问控件窗口产生的所有消息。本例中修改了 CEdit派生类以重载PreTranslateMessage ( )，在那里，把上下箭头键翻译成 Tab和Shift-Tab键，以便敲击上下箭头键时，可以改变一个对话框中控件窗口的键盘输入焦点。有关详细内容参见本节的“ 清

单 —— 编辑类 ”。

3. 用SubclassWindow ( )子分类一个控件

1) 用前面的方法创建一个新的派生类，然后把它们插入到该控件将出现的对话框类、视图类或一些其他类中。

```
CWzdEdit  m_ctrlEdit;
```

2) 用CWnd::GetDlgItem ( )获取需要子分类的控件的句柄，然后用 SubclassWindow ( )子分类该控件。

```
HWND hWnd;
GetDlgItem(IDC_WZD_EDIT2, &hWnd);
m_ctrlEdit.SubclassWindow(hWnd);
```

这种用SubclassWindow ( )的方法能作用于除组合框外的任何控件窗口；一个组合框实际上是由其他两个控件窗口组成的控件：一个编辑框和一个列表框。还是可以用 SubclassWindow ( )子分类一个组合框，但是根据应用程序，可能还需要子分类它的编辑框和列表框。例如，如果需要定制组合框处理鼠标单击编辑框部分的功能，则需要子分类它的编辑框控件，因为编辑框是真正接收鼠标单击消息的部位。

4. 用SubclassWindow ( )子分类一个组合框

1) 要子分类一个组合框的编辑框，首先需要从组合框控件窗口里找到它的窗口。编辑框和列表框都是组合框的子窗口，因此，只要用 CWnd::GetWindow ( )扫描原始组合框的子窗口，然后找到合适的窗口类名，当寻找编辑框时是 EDIT。

```
char className[8];
// examine all of the child windows of the combo box
CWnd* pWnd = m_ctrlComboBox.GetWindow(GW_CHILD);
do {
   ::GetClassName(pWnd->m_hWnd, className, 8);
   // if the class name is "Edit", we've found our control
   if(!stricmp("Edit", className))
   {
      // subclass as before
      m_ctrlComboEditBox.SubclassWindow(pWnd->m_hWnd);
   }
} while((pWnd = pWnd->GetWindow(GW_HWNDNEXT)));
```

在这里可以看到，一旦编辑框控件窗口被找到，可以用前面的方法子分类它。

2) 如果使用组合窗的下拉风格，只有在列表框被下拉时，才能找到它的列表框窗口部分；一旦被下拉，可以用前面的方法子分类它。

说明

控件具有绘制自身和接收消息的所有功能，但却不能轻易修改它们。这就是说，除非创建者有远见，正好提供合适的窗口风格，或提供正好要控制的控件通知。否则，子分类一个控件窗口，通过能够真正地监视控件消息并有选择性地处理其中的一些消息，可以提供更多的控制能力。如果不喜欢控件的外观，截住它的 WM_PAINT消息，并亲自为它绘制一个。如果一个控件没有你需要的或关心的通知消息 —— 现在，通过子分类可以访问它的所有消息。

CD说明

在CD上执行该工程时，单击Test/Wzd命令以打开一个对话框，可以看到这些对话控件都有一个阴影背景，并通过按下上下箭头键在这些控件间移动。

清单——编辑类

```
#if !defined(AFX_WZDEDIT_H__1EC84996_C589_11D1_9B5C_00AA003D8695__INCLUDED_)
#define AFX_WZDEDIT_H__1EC84996_C589_11D1_9B5C_00AA003D8695__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// WzdEdit.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdEdit window

class CWzdEdit : public CEdit
{

// Construction
public:
    CWzdEdit();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdEdit)
    public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdEdit();

    // Generated message map functions
protected:
    //{{AFX_MSG(CWzdEdit)
    afx_msg HBRUSH CtlColor(CDC* pDC, UINT nCtlColor);
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
private:
    CBrush    m_brush;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif // !defined(
    AFX_WZDEDIT_H__1EC84996_C589_11D1_9B5C_00AA003D8695__INCLUDED_)

// WzdEdit.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdEdit.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CWzdEdit

CWzdEdit::CWzdEdit()
{
    m_brush.CreateHatchBrush(HS_BDIAGONAL, RGB(0,255,0));
}

CWzdEdit::~CWzdEdit()
{
}

BEGIN_MESSAGE_MAP(CWzdEdit, CEdit)
    //{{AFX_MSG_MAP(CWzdEdit)
    ON_WM_CTLCOLOR_REFLECT()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdEdit message handlers


HBRUSH CWzdEdit::CtlColor(CDC* pDC, UINT nCtlColor)
```

```
{
    return m_brush;
}

BOOL CWzdEdit::PreTranslateMessage(MSG* pMsg)
{
  if (pMsg->message == WM_KEYDOWN &&
    (pMsg->wParam == VK_UP || pMsg->wParam == VK_DOWN ||
    pMsg->wParam == VK_RETURN))
  {
    int nextprev=GW_HWNDNEXT;
    int firstlast=GW_HWNDFIRST;
    if (pMsg->wParam == VK_UP)
    {
      nextprev=GW_HWNDPREV;
      firstlast=GW_HWNDLAST;
    }
    HWND hWnd = m_hWnd;
    while (hWnd)
    {
      HWND hWndx=hWnd;
      if ((hWnd = ::GetWindow(hWnd, nextprev))==NULL)
        hWnd = ::GetWindow(hWndx, firstlast);
      long style=::GetWindowLong(hWnd,GWL_STYLE);
      if ((style&WS_TABSTOP && !(style&WS_DISABLED) &&
        style&WS_VISIBLE) || hWnd==m_hWnd)
      break;
    }

    ::SetFocus(hWnd);

    return TRUE;
  }

  return CEdit::PreTranslateMessage(pMsg);
}
```

10.3 例48 用超分类定制一个通用控件窗口

目标

从一个控件窗口截取并处理WM_CREATE和WM_NCCREATE消息。不能用子分类做该工作，因为只能子分类一个已有的控件 (即一个已被创建的控件，并已处理了 WM_CREATE和WM_NCCREATE消息)。

策略

要超分类一个控件，只要用系统提供的通用控件窗口类创建一个窗口类，作为模板，然后把窗口过程捆绑到该新窗口类，保存原始窗口类的地址，以便可以发送所有其他消息。因为这一改变将影响每个由该新类创建的控件窗口，因此，在应用程序类中，在应用程序被初

始化前进行替换。

注意 一个窗口类不是一个MFC类，更不是一个C++类，它早于并存在于C++之外，是系统创建任何窗口时使用的模板。参见第1章有关该主题的详细内容。

步骤

1. 创建一个新的窗口类

1) 在应用程序类的 InitInstance ( )函数中，调用一个辅助函数以做实际工作，使InitInstance ( )函数看起来不那么乱。本例中，将超分类所有的按钮控件，因此，辅助函数被叫做SuperclassButtons ( )。

```
BOOL CWzdApp::InitInstance()
{
   SuperclassButtons();

   :  :  :
}
```

2) 在SuperclassButtons ( )中，首先，使用::GetClassInfo ( )，用需要超分类的窗口类的当前信息填充一个WNDCLASS结构(本例中是BUTTON)。

```
void CWzdApp::SuperclassButtons()
{
   WNDCLASS wClass;

   // get existing class information
   ::GetClassInfo(AfxGetInstanceHandle(),"BUTTON",&wClass);
```

3) 接着，把当前BUTTON窗口类的窗口过程保存到一个静态变量中，然后用窗口过程的实例和地址填充该类结构。

```
// at the top of your source
WNDPROC lpfnButtonWndProc;
   :  :  :
   // save old window process and substitute our own
   lpfnButtonWndProc=wClass.lpfnWndProc;
   wClass.hInstance=AfxGetInstanceHandle();
   wClass.lpfnWndProc=ButtonWndProc;
```

4) 最后，用与前面相同的名字注册该类 (即BUTTON)：

```
// register this class
   ::RegisterClass(&wClass);
}
```

系统允许这样做，因为在这里注册一个逻辑窗口类名称，从现在起，当告诉系统用BUTTON窗口类创建一个控件窗口时，它从搜索局部的窗口类开始；当系统发现新类时，它用新类而不是原始类创建窗口。

2. 创建新的窗口过程

1) 在应用程序类的顶部作出下面的声明。

```
LRESULT CALLBACK ButtonWndProc(HWND hWnd, UINT msg,
   WPARAM wParam, LPARAM lParam);
```

2) 用下面的语句创建新的窗口过程，该过程只需处理 WM_CREATE和WM_NCCREAT消息，所有其他消息可以被发送到原始的窗口过程。

```
LRESULT CALLBACK ButtonWndProc(HWND hWnd, UINT msg,
   WPARAM wParam, LPARAM lParam)
{
   switch (msg)
   {
      case WM_CREATE:
         break;
      case WM_NCCREATE:
         break;
   }
   return (lpfnButtonWndProc)(hWnd,msg,wParam,lParam);
}
```

说明

- 参见第3章有关子分类和超分类的详细内容。
- 本例中，创建的是一个局部窗口类；如果包括 CS_SYSTEMGLOBAL类风格，并注册该类，则可以创建一个全局类。然而，一个全局窗口类没有它以前具有的相同的能力。在Windows 3.1中，一个系统全局类可以影响整个系统，然而从 Win95和NT 4.0开始，它只能影响一个应用程序。参见第 1章有关窗口类的更详细内容。
- 本例适用于任何窗口类。参见 1.7节“厂商安装的窗口类”中有关 Windows操作系统提供的窗口类的列表。
- 在本例中，应用程序创建的所有按钮控件都出现在新的窗口过程中。如果想要有更大的选择性，可以在注册前重命名窗口类 —— 可以叫它MYBUTTON。此后，每当想要用该新的窗口类创建一个新的按钮控件时，就可以用 CWnd::CreateEx ( )和新的窗口类名实现。使用对话框编辑器，用该新窗口类创建一个控件，可以添加一个定制控件到对话框模板；一个定制控件允许你指定窗口类名，在创建控件时，系统将使用该窗口类名。
- 这里包括超分类，只是为了内容上的完备性，其实，几乎不需截获 WM_CREATE或WM_NCCREATE消息。如果在一个控件可视以前，要改变它的风格；只要在创建时使它不可视，改变它的风格，然后使它可视。超分类的功能非常强大，但也非常棘手。

CD说明

- 在CD上执行该工程时，在wzd.cpp底部的ButtonWndProc ( )处设置一个断点。现在可以看到，每当一个消息被发送到应用程序的一个按钮控件时，应用程序都被中断。

10.4 例49 在按钮上放置位图

目标

用位图替换按钮控件上的文本，或者提供用来绘制按钮控件的所有位图 (见图10-2)。

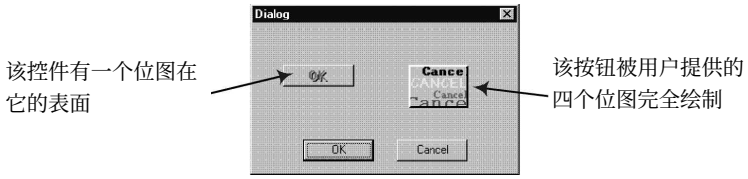

图10-2 在按钮上使用位图

策略

要把一个位图放置到按钮上，只要把按钮的风格设置为 BS_BITMAP，然后用CButton::SetBitmap ( )告知该按钮显示哪个位图；也可以创建一个自绘制的按钮，它不仅允许用户在它表面放置一个位图，而且还允许用户画它的边框。为了用户自身绘制该控件，只要设置按钮的风格为BS_OWNERDRAWN和重载CButton的DrawItem( )成员函数即可。然而，要使该工作不用处理DrawItem( )，可以用CBitmapButton类，它用我们提供的位图(最多四个)装入并绘制按钮。

步骤

1. 添加一个位图到一个按钮

1) 要在一个对话框模板中创建一个具有位图表面的按钮，用对话框编辑器打开它的属性框，并选定Bitmap属性。

2) 把一个位图类插入到使用该模板的对话框类中。

```
CWzdBitmap m_bitmap;
```

本例使用一个将在例57中创建的位图类，它要求位图与按钮有相同的背景色，并且混和后效果更好；需把该类插入对话框类中，因为即使在下一步中将使用 CButton:SetBitmap ( )把一个位图分配给一个按钮，还是有责任维护该位图对象。如果不做这一步，则不能获得任何编译、链接或运行错误，但在按钮上也不能得到一幅位图。

3) 用ClassWizard添加一个WM_INITDIALOG消息处理函数到使用该模板的对话框类，在那里告知按钮显示哪个位图。

```
m_bitmap.LoadBitmapEx(IDB_WZD_BUTTON,TRUE);
((CButton *)GetDlgItem(IDC_WZD_BUTTON1))->SetBitmap(m_bitmap);
```

2. 用CBitmapButton创建一个按钮

1) 用对话框编辑器为合适的按钮选定自绘制属性。

2) 同时用对话框编辑器赋予按钮一个唯一的标题，如 MYBUTTON。

本例用CBitmapButton绘制按钮，为它提供了多达四个位图：一个为按钮按下时，一个为按钮松开时等。使用CBitmapButton来确定应用程序资源中的哪一个位图绘制该按钮的方法有些烦琐。首先，它获得用对话框编辑器赋予它的标题 (本例中为MYBUTTON)；然后，用转换标题得到的文本ID在应用程序资源中寻找四个位图。文本 ID为MYBUTTONU的位图标识按钮松开时绘制的位图；MYBUTTOND在按钮按下时被绘制；MYBUTTONF当按钮有焦点时被绘制；MYBUTTONX在按钮无效时被绘制。可以给予资源一个文本 ID，而不是数字ID，只要在定义时，用双引号括住它即可。

3) 用位图编辑器创建四个按钮并给予每个按钮一个合适的文本 ID；注意只有松开和按下是要求的。还应注意当绘制一个按钮时，需要给予该按钮一个 3D的外观，这可以通过沿着边

框精心放置一些凸出显示或阴影显示的直线来实现。

4) 在对话框类中插入一个CBitmapButton类变量。

5) 用ClassWizard添加一个WM_INITDIALOG消息处理函数到对话框类中。在那里，用CBitmapButton::AutoLoad ( )装入按钮的位图。提供给AutoLoad的ID是想要绘制的按钮控件的ID。

```
CBitmapButton m_bitmapButton;
m_bitmapButton.AutoLoad(IDC_WZD_BUTTON2,this);
```

可以看到，系统绘制的按钮比自绘制的按钮更灵敏。如果快速单击一个系统绘制的按钮多次，它将对每次单击都作出反应，而一个自绘制的按钮只是偶尔作出反应。不管什么原因，系统绘制的按钮可以处理一个双击事件，而自绘制的按钮不能，双击处理必须处理什么呢？记住，连续快速单击某对象必定要产生一些双击事件 (一次双击只是两次连续快速单击)。系统绘制的按钮处理一个双击就象它是一个单击一样，而自绘制的按钮完全忽略它们。为了让自绘制的按钮也能处理双击事件，我们将处理按钮的 WM_LBUTTONDBLCLK消息。

3. 使CBitmapButton更灵敏

1) 用ClassWizard创建CBitmapButton的派生类。

2) 用ClassWizard添加一个WM_LBUTTONDBLCLK消息处理函数到该类中；在那里，将要发送一个新增的WM_LBUTTONDOWN消息。

```
void CWzdButton::OnLButtonDblClk(UINT nFlags, CPoint point)
{
   SendMessage(WM_LBUTTONDOWN, (WPARAM)nFlags,
      (LPARAM)MAKELONG(point.x, point.y));
}
```

很快，按钮又变灵敏了。

说明

CBitmapButton类减少了一个自绘制按钮的许多工作。然而，它也使得有时难以获得这样的一些控件；例如，如果想创建一个按钮表面具有系统色的按钮，则不能用CBitmapButton实现它，因为在运行时不能访问该按钮位图。在这种情况下，需要重载DrawItem ( )并绘制按钮。

CD说明

在CD上执行该工程时，单击Test/Wzd命令以打开一个对话框，它显示了两个位图按钮。

10.5 例50 动态填充一个组合框

目标

打开组合框时用数据填充它 (见图10-3)。

策略

因为下拉组合框的列表只有用户单击了下拉按钮后才是可视的，因此可以在任何时间填充它。可以在它还在对话框模板中时，用对话框编辑器添加项到一个组合框；也可以在对话框类的OnInitDialog ( )成员函数中，第一次创建组合框时添加项。否则，如果数据要求及时更

新，甚至可以在组合框打开它的下拉列表时添加项。本例通过处理两个来自组合框控件的控件通知，讨论上述的最后一种情况。

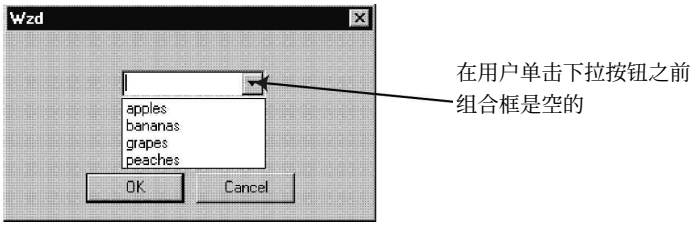

图10-3 在组合框打开以刷新数据前填充它

步骤

动态填充一个组合框

1) 用对话框编辑器添加一个组合框到一个对话框模板中。

2) 用ClassWizard创建一个使用该对话框模板的对话框类。

3) 用ClassWizard添加一个成员变量到该对话框类，它与该组合框交换 CString值(见例39)。本例中，该变量名为m_sSelection。

4) 使用该对话框类时，用合法值初始化该成员变量。

5) 用ClassWizard为组合框的CBN_DROPDOWN控件通知(见例59)添加一个消息处理函数；在那里，应用CComboBox::AddString ( )把文本填充到组合框中；然后，选择与当前组合框的编辑框相匹配的条目(如果有的话)。

```
void CWzdDialog::OnDropdownWzdCombo()
{
   CComboBox *pCombo=(CComboBox *)GetDlgItem(IDC_COMBO_BOX);
   pCombo->ResetContent();
   pCombo->AddString("apples");
   pCombo->AddString("peaches");
   pCombo->AddString("bananas");
   pCombo->AddString("grapes");
   pCombo->SelectString(-1,m_sSelection);
}
```

6) 用ClassWizard添加一个CBN_CLOSEUP控件通知处理函数到对话框类中，在那里获取用户选中的选项(如果有的话)，并把它填充到组合框的编辑框中。

```
void CWzdDialog::OnCloseupWzdCombo()
{
   int    nSel;
   if ((nSel=m_ctrlCombo.GetCurSel())!=CB_ERR)
   {
      m_ctrlCombo.GetLBText(nSel, m_sSelection);
   }
   else
   {
      m_ctrlCombo.ResetContent();
      m_ctrlCombo.AddString(m_sSelection);
      m_ctrlCombo.SelectString(-1,m_sSelection);
   }
}
```

注意如果没有任何项被选中，该处理函数只是恢复编辑框的原始内容。

7) 当对话框关闭时，则m_sSelection值包含用户作出的任何选项。

说明

- 当编辑一个组合框时，使用对话框编辑器有两项技巧：可以在组合框下拉时增大组合框列表的大小，也可以添加数据项到它的列表中。要增大组合框下拉列表的尺寸，首先把该组合框添加到对话框模板，然后单击下拉按钮，一个新的轮廓出现，这时可以拖动它使它变得更大；要添加数据项到列表中，打开组合框的属性框，并选择第二个选项卡，确保在数据项间按Control+Enter键。
- 组合框有三种风格：Simple风格的组合框有一个列表总是可视的；Dropdown风格只在用户单击下拉按钮时才显示列表，它也允许用户输入他们自己的数据到组合框的编辑框中；Droplist风格与Dropdown风格相似，但它不允许用户输入自己的数据。
- 组合框类，CComboBox，有一个成员函数称为CComboBox::Dir ( )，它能用磁盘上一个子目录的内容填充它的列表。本例中可以用 Dir ( )作为CComboBox::AddString ( )的字符源，虽然考虑到它有点不灵活，需要重新格式化它们的输出。
- CBN_DROPDOWN消息是一个控件通知，它由组合框发送，允许父窗口处理该控件不能处理的消息。本例中，控件知道怎样绘制自己和接收输入，但只有父窗口知道怎样用数据填充它。有关控件通知的详细内容参见第 3章。

CD说明

在CD上执行该工程时，单击 Test/Wzd菜单命令以打开一个具有空组合框的对话框；可以在WzdDialog.cpp内设置一个断点，以观察每次用户单击下拉按钮时组合框被填充的情况。

10.6 例51 排序一个列表控件

目标

当用户单击列表控件的标题时，排序列表控件的列。注意，一个列表控件只有在使用Report窗口风格时才有多列(见图10-4)。

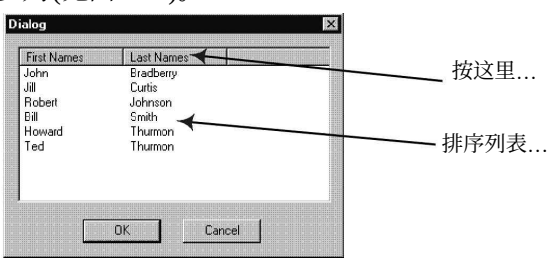

图10-4 通过处理LVN_COLUMNCLICK排序一个列表控件

策略

每当用户单击列表控件的一个标题时，它将发送一个 LVN_COLUMNCLICK控件通知；通过使用CListCtrl::SortItems ( )函数处理该通知。

步骤

列表控件允许用DWORD变量存储列表中的每个项，该变量完全是用户定义的并且通常是一个数据类的实例的指针，或者完全定义列表中该项的结构。事实上，当排序一个列表控件时，所有要做的工作就是逐个比较 DWORD变量。本例以定义自己的数据类开始，该类只是为列表项中每个列包含一个成员变量。

注意 如果列表中的项已经存在于文档中的一些数据集里，则不在这里创建一个特殊的数据类，而使用其他类。

1. 创建一个数据类

1) 创建一个描述列表中每个项的数据类。本例中，数据类包含一个第一个名字和最后一个名字的成员变量。

```
class CWzdInfo : public CObject
{
public:
   DECLARE_SERIAL(CWzdInfo)
   CWzdInfo();
   CWzdInfo(CString sFirst,CString sLast);

   // misc info
   CString m_sFirst;
   CString    m_sLast;
};
```

2) 以DWORD指针形式，把一个数据类的实例添加到列表控件的每一行。

```
m_ctrlList.SetItemData(0,(DWORD)new CWzdInfo("Bill", "Smith"));
```

2. 排序一个列表控件

1) 用ClassWizard添加一个LVN_COLUMNCLICK消息处理函数到列表控件的父窗口中，父窗口既可以是一个对话框也可以是一个视图。在该处理函数中，调用列表控件的 SortItems ( ) 成员函数。

```
void CWzdDialog::OnColumnclickWzdList(NMHDR* pNMHDR,
   LRESULT* pResult)
{
   NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
   m_ctrlList.SortItems((PFNLVCOMPARE)CompareColumnItems,
      pNMListView->iSubItem);

   *pResult = 0;
}
```

CompareColumnItems ( )由我们提供，告诉SortItems ( )哪个列表项在哪个行项的前面；接着，创建CompareColumnItems ( )函数。

2) 用下面的语句创建CompareColumnItems ( )函数。

```
int CALLBACK CompareColumnItems(CWzdInfo* pItem1,
   CWzdInfo* pItem2, LPARAM lCol)
{
   int nCmp = 0;
   switch(lCol)
   {
```

```
    case 0: //column 1
        nCmp = pItem1->m_sFirst.CompareNoCase(
            pItem2->m_sFirst);
        break;
    case 1: //column 2
        nCmp = pItem1->m_sLast.CompareNoCase(pItem2->m_sLast);
        break;
    }
    return nCmp;
}
```

列表控件类的SortItems ( )函数传递两个列表控件项的DWORD变量到该函数。因为这里已捆绑了一个数据类的指针，事实上传送的是要比较的两个数据类的指针。如果第一项应该在第二项之前，则返回值为负；相反则为正；如果相等则为 0。

3. 销毁数据类

如果是创建一个新的数据类实例，而不是使用一个已有的文档数据集，则需要在列表控件被销毁时删除这些实例；通常可以在这些实例所在的对话框类或视图类的析构函数中删除它们。

```
void CWzdDialog::OnDestroy()
{
    CDialog::OnDestroy();

    // destroy all data items
    int i= m_ctrlList.GetItemCount();
    while(i>-1)
    {
        delete (CWzdInfo *)m_ctrlList.GetItemData(i--);
    }
}
```

说明

- 列表控件还有一种窗口风格，它使列表框自动地按第一列的字母顺序排列列表项，使用这种方法可以根据认为合适的方法调整任何一列。
- LVN_COLUMNCLICK消息是一个控件通知，它由列表控件发送，允许父窗口处理它所不能执行的处理。该控件知道怎样移动周围的分隔线，但只有父窗口知道哪个在哪个之前。有关控件通知的详细内容参见第 3章。

CD说明

在CD上执行该工程时，单击 Test/Wzd菜单命令，打开一个具有列表控件的对话框，然后单击列表控件中的任一列标题以字母顺序调整那列。

10.7 例52 分隔线控件

目标

添加一条垂直或水平的蚀刻线到对话框中，如图 10-5所示。

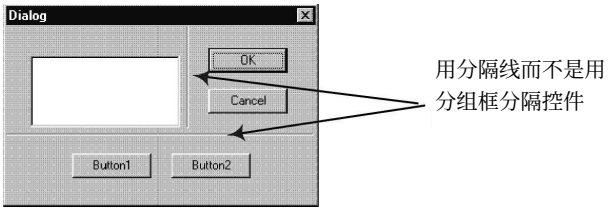

图10-5 对话框中的分隔线

策略

听起来容易，不是吗？好吧，事实上它很容易——但只是不很明显而已。不能在伴随对话框编辑器的控件中找到一个蚀刻的分隔线控件。与分隔线工具最接近的是分组框控件；可以用一个分组框控件，并使它的名字空白，但是，这时所能得到的只是一个矩形而已；如果再找一遍，还可以发现Picture Control也可以创建一个蚀刻的矩形，但是还是不是线；再经过一番搜索，可以发现通过创建一个SS_ETCHEDHORZ或SS_ETCHEDVERT风格的静态控件，通过编程的方式创建一个分隔线控件。但是对话框编辑器不能设置这些窗口风格，而且用 Create ( )创建分隔线控件也是非常辛苦的，尤其当需要精确放置这样一个控件时。

因此，该怎么办呢？事实上，可以用 Picture Control来创建一个分隔线，前面只是与你开了个玩笑。

步骤

在对话框中创建一个蚀刻分隔线

1) 用对话框编辑器添加一个Picture Control到对话框模板。
2) 激活那个控件的属性。在Color组合框中，选择Etched，同时Type应该为Frame。
3) 现在选定控件上的一个移动句柄，使该控件一直缩小到一条直线为止。

说明

- 可以用相同的方法创建一个黑、白或灰色分隔线，只要在 Picture Control的属性中选择一种不同的颜色即可。
- 用Picture Control创建的控件事实上是静态文本控件窗口。通常可能会认为一个静态文本控件在对话框里只显示文本，然而一个静态文本控件还有其他窗口风格，并使它可以创建多种非文本显示，包括一条蚀刻线。参见附录 A中有关静态文本控件的其他风格。
- 按钮控件也能处理许多窗口风格，有下压按钮、复选框和单选按钮；更令人难以置信的是，一个分组框，事实上是一个具有 BS_GROUPBOX风格的按钮控件。参见附录 A中有关按钮风格的内容。

CD说明

在CD上执行该工程时，单击Test/Wzd菜单命令以打开一个具有两条蚀刻线的对话框。