第9章　对话框和对话条

继视图之后，对话框是应用程序具有的接收用户信息的另一种重要机制。在一个对话框中发现的控件窗口和标准，也可以在对话条中发现。对话条是对话框、工具栏和窗体视图的综合，其中窗体视图是对话框和视图的综合。

例38　使用对话框编辑器　用对话编辑器创建一个对话框模板，对话框模板只是记录了一组控件窗口的大小和位置。当用该模板创建一个对话框时，这些窗口也将被创建。

例39　创建一个对话框类　用ClassWizard把上例中创建的对话框模板封装到 C++类中，并允许数据自动与那个类的控件成员变量进行交换。

例40　模式对话框　创建一个模式对话框，也就是说在应用程序继续执行之前，它将一直等待该对话框的终止。

例41　无模式对话框　创建一个无模式对话框，尽管用户还必须释放该对话框，但它允许应用程序继续运行。

例42　在无模式对话框的控件间切换焦点　讨论在对话框的控件间怎样恢复切换焦点的能力，该能力在模式对话框中是自动产生的，而在无模式对话框中被禁用。

例43　对话框中的动画　本例讨论怎样修饰一个对话框，使它在冗长的操作中显示描述操作进度的动画。

例44　消息框　本例讨论特定的Message Box对话框的能力，它能提示用户作出 Yes或No的回答。

例45　对话条　本例创建一个标准对话条，它是对话框和工具栏的综合，对话条的内部内容可以在对话框编辑器中设计，但是它能像工具栏一样停放在应用程序窗口的边上。

9.1　例38　使用对话框编辑器

目标

在应用程序资源中添加或修改一个对话框模板，该模板可以用来创建一个对话框或属性页。

步骤

1. 利用Developer Studio创建一个新对话框模板

要创建一个新的对话框模板，单击 Developer Studio的Insert/Resource…菜单命令以打开Insert Resource对话框，选择Dialog，然后单击New按钮。

2. 用对话框编辑器编辑一个已有的对话框模板。

1) 要编辑一个已有的模板，单击工作空间窗口的 Resource View选项卡，然后在Dialog文件夹中找到那个模板的ID，并双击它。

2) 对话框编辑器的工作是把控件从控件工具栏拖到对话框中；控件刚被创建时，它的属

性框是打开的；但也可以通过右击该控件，然后在弹出式菜单中选择 Properties命令打开它们。属性随控件类型不同而不同，并能粗略反映那个控件可能的窗口风格；不是一个控件可用的所有风格都显示在属性对话框中；有关详细列表参考 MFC文档，有关任何一个特定控件的可获得的最有意义的风格的列表，参见附录 A。如果一个风格对于一个控件是可用的，但没有列在属性对话框中，则需要自己创建该控件(见例 46)。

注意 打开对话框编辑器还会自动显示控件工具栏(见图9-1)；如果没有打开，可以通过单击Tools/Customize菜单命令以打开环境的Customize对话框，然后单击ToolBar选项卡，在列表框中找到Contols项并确信它被选中。

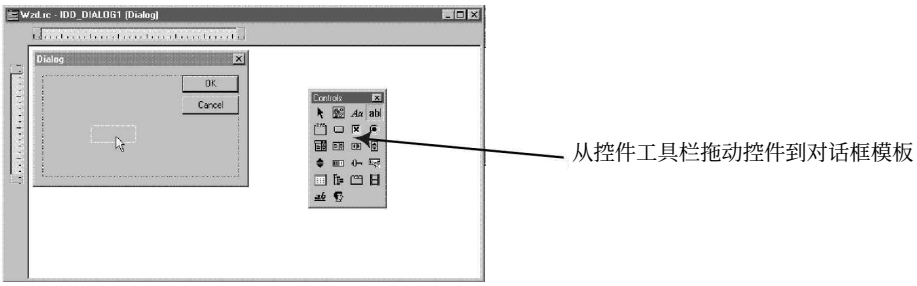

图9-1 对话框编辑器自动显示控件工具栏(Controls Toolbar)

3) 要调整模板中控件的位置，用 Developer Studio的Layout命令下的菜单命令，有关这些命令的更详细的消息参见MFC文档。

4) Layout/TabOrder菜单命令允许指定在按Tab键时这些控件接收输入焦点的顺序。事实上，所有这些命令所做的是在资源文件中记录控件声明。 Tab Order还决定控件窗口在对话框中被画的顺序，在两个控件窗口重叠时，可以确定哪个在上面，具有更高级的 tab顺序的控件将显示在上面。

5) Developer Studio的Layout/Test菜单命令允许预览那个模板显示对话框时的外观。要退出该模板，可以单击任何按钮或按 Escape键。

6) 特定的对话框模板风格及其使用，在全书范围内都讨论。

说明

- 当创建一个对话框时，对话框模板决定了创建哪些控件以及这些控件所具有的风格，参见例40、41有关从对话框模板创建一个对话框。
- 对话框模板的大小以及该模板中控件的大小和位置是基于对话框单位的——不是像素。当一个对话框模板用来创建一个对话框时，所有的对话单位根据对话框使用的字体转换为像素。字体越大，最终显示的对话框也越大。通常这不是问题，如果想动态地添加自己的控件到对话框中，只要在控件显示的位置放置一个占位控件，如一个静态文本控件，然后使用GetWindowRect ( )和SetWindowRect ( )，把占位控件大小转换为新控件。然而，如果需要直接使用该模板，参见第4章讨论的对话框单位和像素间的转换。

CD说明

在CD上没有该例子相关的工程。

9.2 例39 创建一个对话框类

目标

在已创建一个前面例子中看到的对话框模板后，再创建一个对话框类，以简化从该模板创建对话框的工作。

策略

用ClassWizard创建一个对话框类，它有助于从对话框模板创建一个对话框；用ClassWizard添加与对话框中的控件相对应的成员变量到该类中。

步骤

1. 用ClassWizard创建一个对话框类

在对话框编辑器中，相应的对话框模板打开时，单击 Developer Studio的View/ClassWizard菜单命令以打开ClassWizard对话框。ClassWizard一出现将询问是否需要创建一个新的对话框类以支持该模板，回答 Yes，然后在New Class对话框中输入一个合适的类名，并确信从CDialog派生，再单击OK。

注意 要从ClassWizard中删除一个类，不仅需要从工程中删除该类的.cpp和.h文件，而且还要删除工程目录中的 .clw文件。该文件由 ClassWizard创建用来跟踪类，如果ClassWizard不能找到.clw文件，它将创建一个新的。

2. 添加控件消息处理函数到对话框类

一旦在ClassWizard中打开对话框类，可以在ClassWizard的Message Maps标签处看到一组对话框模板的控件ID的列表；可以用ClassWizard为每个控件添加一个消息处理函数，首先选择合适的控件ID，然后在Messages列表框中找到并选择一个消息 ID。根据指定的控件类型显示的相应消息，不同的控件类型显示不同的消息。然后单击 Add Function按钮真正地添加该消息处理函数(见图9-2)。

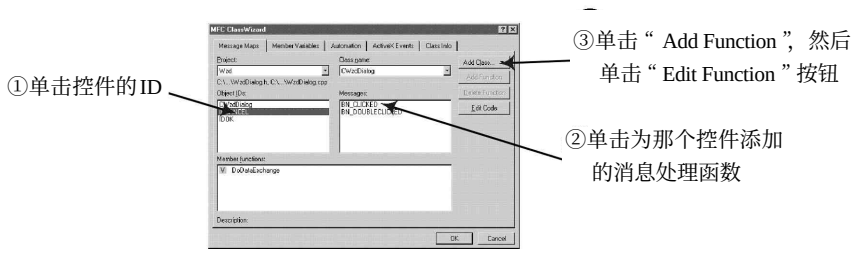

图9-2 用ClassWizard添加一个消息处理函数

3. 添加成员变量到对话框类

1) 用ClassWizard添加成员变量到对话框类中，它们用来与对话框中的控件交流数据。要添加这些成员变量，单击 ClassWizard的Member Variables标签；然后单击需要添加成员变量的控件ID，然后单击Add Variable按钮打开Add Member Variable对话框(见图9-3)。

2) 输入变量名后，就要选择该变量与控件交互的方式，可用的交互方式取决于变量的控

件，例如一个编辑框将与一个字符串变量或一个整型变量交换内容，要与一个整型变量交换内容，一个编辑框必须在内部把它的内容转换为一个整型；所有控件允许以控件变量 (control variable)的方式与它们交互。事实上，这种方法可以容易地子分类对话框中的一个控件。然而，在CWnd::UpdateData ( )被调用之前，对话框中的控件不与成员变量交换数据。 UpdateData (FALSE)将获得成员变量的值，并把它们存储到相应的控件窗口； UpdateData (TRUE)将获取当前在控件中的值，并把它们存储到相应的成员变量中。

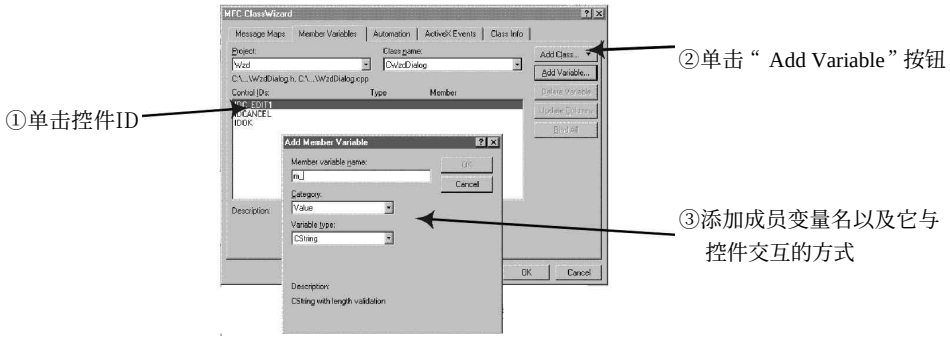

图9-3 用ClassWizard添加一个成员变量

3) 成员变量一旦输入就不能修改它，而必须删除该成员变量，然后重输入它。

4) 可以让多个成员变量访问同一个控件，例如一个控件可以同时与一个字符串、数字和控件成员变量交换数据；然而，不能自动地添加这些变量，而需要少许的人工干预。首先，用ClassWizard添加一种类型的成员变量；然后，把 ClassWizard创建的定义移到源文件 (包括.cpp和.h文件)中为ClassWizard维护的{{ }}括弧外面，一旦在这些{{ }}括弧外面，就可以自由地为该控件添加另一种类型的成员变量。

5) 还可以在一个具有这样的控件 (具有一个或多个成员变量)的对话框类中拥有一个消息处理函数；按钮控件就是很好的例子，可以有一个处理按钮被单击时的消息处理函数，但还可以有一个子分类按钮的成员变量以便可以轻易地用下面的语句隐藏该按钮。

```
m_wndButton.ShowWindow (SW_HIDE)
```

6) 对于一些控件，还可以添加有效性验证条件。例如，可以为一个字符串变量控件指定一个最大长度；对于一个接收一个数字的编辑框，可以指定一个数字范围。

说明

- 添加到对话框类的成员变量可以被创建该对话框的类轻易访问；有关细节参见下一个例子。
- ClassWizard显示的控件消息不可能总是包括那个控件能产生的所有消息；在这种情况下，需要自己添加控件消息映像宏，参见附录 B有关合适的宏。
- 数据交换在对话框类的DoDataExchange ( )成员函数中进行，它用MFC支持的函数，为所有标准变量类型实现数据交换。

CD说明

CD上没有本例的相应工程。

9.3 例40 模式对话框

目标

在挂起应用程序情况下，提示用户输入(见图9-4)。

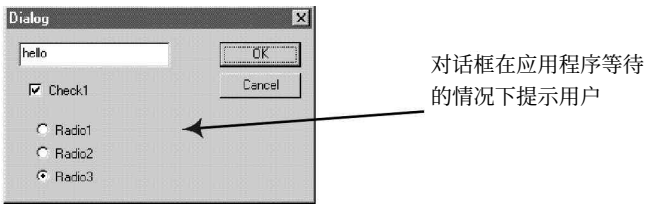

图9-4 用一个模式对话框挂起应用程序直到用户作出响应

策略

调用上个例子创建的对话框类中的 DoModal ( )成员函数，创建一个模式对话框。首先把输入的值放到对话框类的成员变量中，然后在对话框关闭时接收它们。

步骤

创建一个模式对话框

1) 如上两个例子所示，创建一个对话框资源和类，为对话框模板中的每一个控件添加一个成员变量到对话框类中。

2) 在堆栈中创建对话框类的一个实例。

```
CwzdDialog  dlg;
```

3) 在对话框类中，用调用类的值初始化成员变量；然后，调用对话框类的 DoModal ( )函数；如果DoModal返回IDOK，则把那些成员变量的值返回给对话框类。

```
dlg.m_sName=sName;
dlg.m_nRadio=nRadio;
dlg.m_bCheck=bCheck;
if (dlg.DoModal()==IDOK)
{
   sName=dlg.m_sName;
   nRadio=dlg.m_nRadio;
   bCheck=dlg.m_bCheck;
}
```

说明

- 对于真正的C++封装，不能直接访问对话框类的成员变量，然而，一个对话框类的成员变量和属性页通常被直接访问。由于每个类的存在期是如此短，这种直接的访问并不会产生太大的坏处。
- 当用对话框编辑器创建对话框模板时，可以给予对话框一个 System Modal(系统模式)风格；过去用Window 3.1的时候，这意味着在用户作出反映以前，对话框不仅挂起应用

程序中的所有活动，而且挂起系统的所有行为。而在当前真正的多任务应用程序环境中，所有系统模式能做的是使你的对话框成为最顶层窗口，这意味着任何窗口都不能覆盖它(当然，除另一个最顶层窗口外)，用户应该不厌其烦地回答你的提问。通常在影响整个系统的关键情况下，使用系统模式对话框，包括从系统资源到打印机失败的任何事件。

CD说明

在CD上执行该工程时，单击 Test/Wzd菜单命令，显示一个模式对话框；在关闭该对话框以前，将禁止对应用程序的任何输入。

9.4 例41 无模式对话框

目标

创建一个对话框，即使用户没有关闭该对话框，也能允许应用程序继续运行。在应用程序继续工作时，无模式对话框可用来代替沙漏光标 (见图9-5)。

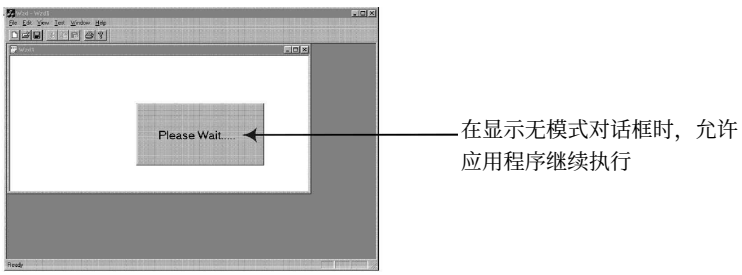

图9-5 用一个无模式对话框允许应用程序继续运行，而不管用户有无作出反映

策略

如同例38、例39那样创建一个对话框模板和对话框类，然后用新的对话框类的 Create ( ) 成员函数创建一个无模式对话框；要关闭无模式对话框，可以用 CWnd::DestroyWindow ( )。

步骤

1. 创建一个无模式对话框

用对话框编辑器在应用程序中创建一个对话框资源，然后用 ClassWizard创建一个对话框类，这时可以用该对话框类创建一个无模式对话框。

```
CWzdDialog *pDlg;
pDlg = new CWzdDialog;
pDlg->Create(IDD_WZD_DIALOG);    // id of dialog box resource
pDlg->ShowWindow(SW_SHOW);       // dialog is initially hidden
```

注意对话框类的实例在堆中创建。如果象上例中创建模式对话框那样，已把它创建在堆栈里，则当函数返回时，对话框类被析构，自动销毁对话框；如果不想把对话框创建在堆中，还可以把它作为创建该对话框的类的成员变量插入。

```
CwzdDialog   m_dlg;
```

2. 销毁一个无模式对话框

1) 要销毁一个无模式对话框，可以用：

```
pDlg—> DestroyWindow ( ) ;
```

2) 用户可以通过单击对话框窗口右上角的关闭按钮销毁一个无模式对话框，然而，当用户用这种方法关闭一个对话框窗口时，操作系统不知道是否应该删除对话框类。因此用户必须在对话框类中做这一步。用ClassWizard重载对话框类的PostNcDestroy ( )函数，删除它自己的实例。

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

   delete this;
}
```

3. 使用一个等待对话框

1) 例34介绍怎样创建一个沙漏光标，指示用户应用程序正忙。对于一个更长时间的等待，建议使用一个无模式对话框，只要在它上放置一个静态文本“ Please wait… ”消息、一个进度指示控件或者一个动画控件，甚至还可以在它上面放置一个按钮以停止继续执行。然后，在应用程序变忙以前，创建一个等待对话框。

```
CWzdWaitDialog dlg;;
dlg.Create(IDD_WAIT_DIALOG);
dlg.ShowWindow(SW_SHOW);
dlg.UpdateWindow();
// processing.....
```

这里对CWnd::UpdateWindow ( )的调用前面没有讨论过，因为对话框窗口可能难以取得正常的WM_PAINT消息，因此，需要用UpdateWindow ( )强制取得一个。

2) 一旦函数调用结束，等待对话框类会销毁自己。在它的析构函数中，它会销毁对话框窗口。

说明

在对话框的控件间切换焦点的能力，在无模式对话框中被认为是关闭的。有关怎样重新激活这一功能参见例42。

CD说明

在CD上执行该工程时，在Test菜单中可以看到3个菜单项。Test/Create将创建一个无模式对话框，它不用象模式对话框那样挂起应用程序。 Test/Destroy将销毁该对话框。Test/Wait将显示一个等待对话框，创建它的函数结束一个简单的增式循环 (大约2秒钟)，循环结束后等待对话框消失。

9.5 例42 在无模式对话框的控件间切换焦点

目标

在无模式对话框控件间切换焦点。当创建一个模式对话框时，该功能可以自动获得，而

无模式对话框则不能。

策略

在对话框编辑器中创建该对话框模板时，在属性对话框中再为该对话框选择一种风格。

步骤

启用对话框控件切换焦点

在该对话框的对话框编辑器中，右击该对话框以产生它的属性，然后选择 Extended Styles 标签并单击Control Parent，这样就启用了切换焦点的功能。

说明

- 单击Control Parent使创建的对话框具有WS_EX_CONTROLPARENT窗口风格。事实上，不仅仅是一个对话框可以获得该功能，创建的任何窗口都可以具有WS_EX_CONTROLPARENT风格。当在该父窗口中创建控件窗口时，使每个控件包括 WS_TABSTOP风格。一旦创建，按TAB键时，将使键盘输入焦点按照创建控件窗口的顺序在它们之间切换。
- 如果想用上下箭头键切换不同窗口间的焦点，参见例 47，在该例子中，把上下箭头键敲击转换为Tab和Shift-Tab键敲击。

CD说明

在CD上执行该工程时，单击 Test/Wzd菜单命令，打开一个无模式对话框，按 Tab键使输入焦点在控件间切换。

9.6 例43 对话框中的动画

目标

在对话框中显示一个动画，指示函数调用的进度 (见图9-6)。

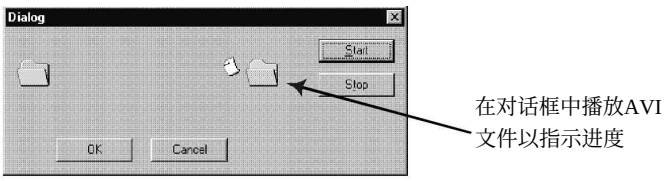

图9-6 用对话框中的动画指示进度

策略

用对话框编辑器添加一个动画控件到对话框模板，接着在对话框类的 OnInitDialog ( )中用一个.avi文件装入该控件，然后用动画控件类的成员函数开始和停止播放该 .avi文件。

步骤

1. 输入.avi文件到应用程序的资源

单击Developer Studio的Insert/Resource菜单项，打开Insert Resource对话框，然后单击Import按钮并找到.avi文件。对于资源类型，输入“ AVI ”(没有双引号)，赋予该资源一个合适的ID，如本例中用IDR_FILECOPY。

2. 添加一个动画控件到对话框

1) 用对话框编辑器添加一个动画控件到对话框模板 (动画控件是控件工具栏中的按钮，外观看起来象电影片断)。对于它的属性，选取centered和transparent。

2) 如果还没有创建对话框类，用 ClassWizard为该对话框模板创建一个对话框类。要装入.avi文件到该控件，可以用下面的代码 (假设IDC_ANIMATE_CTRL是该控件的ID)。

```
CAnimateCtrl *pCtrl=(CAninmate *)GetDlgItem(IDC_ANIMATE_CTRL);
pCtrl->Open(IDR_FILECOPY);
```

3. 播放.AVI文件

1) 要求该控件播放.AVI文件，可以用：

```
pCtrl->Play(0,    // first frame
   -1,            // last frame (-1=play every frame)
   -1);           // number of times to play avi (-1==
                  // play until manually stopped)
```

2) 要停止播放可以用：

```
pCtrl->Stop();
```

说明

- .avi文件是由一系列位图文件构成的，每个位图叫做一帧。有许多公司创建可以作为网络共享件的.avi文件，它们的平均价格约40美元。
- 对话框上的动画通常是用来显示一个冗长操作的进度的较华丽的方法。为了完美的效果，应该在该操作开始前用 CWnd::ShowWindow (SW_HIDE)隐藏该控件。然后当开始运行时，应该在显示该控件的同时开始播放动画。

```
pCtrl->ShowWindow(SW_SHOW);
pCtrl->Play(0,-1,-1);
```

- 当该过程结束时，要停止该控件并重新隐藏它，用：

```
pCtrl->ShowWindow(SW_HIDE);
pCtrl->Stop();
```

CD说明

在CD上执行该工程时，单击 Test/Wzd菜单命令打开一个对话框，单击 Start，开始播放动画，然后单击Stop停止它。

9.7 例44 消息框

目标

在不用创建一个新的对话框模板和对话框类的情况下，用一个简单的 Yes或No问题提示用户(见图9-7)。

策略

用两个框架函数创建一个消息框： AfxMessageBox ( )和AfxFormatString ( )。

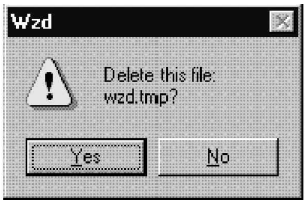

图9-7 使用一个简单的Yes或No问题的消息框

步骤

使用消息框

1) 创建一个简单的消息框，可以用：

```
AfxMessageBox ("Hello");
```

一个只有OK按钮的模式对话框被创建，它等待用户单击。

2) 创建一个使用应用程序资源文件中字符串表中字符串的简单消息框，用：

```
AfxMessageBox (IDS_HELLO);
```

3) 要动态地编辑字符串表中的字符串，用 AfxFormatString1 ( )：

```
AfxFormatString1(msg, IDS_HELLO_1, "Goodbye");
AfxMessageBox(msg);
```

本例中，IDS_HELLO_1代表字符串表中的“ Hello %1 ”字符串，AfxFormatString1 ( )用%1替换“ Goodbye ”，这种方法允许用户保持绝大多数的消息字符串在字符串表中，保持绝大多数字符串在字符串表中使应用程序可以被轻松地转变为另一种语言，如西班牙语或法语，只需用另一个字符串表替换一个字符串表，也可以把该字符串表放入资源库中 (见例85)。

4) AfxFormatString2( )替换两个字符串而不是 1个，在下面的例子中，消息字符串是“ Hello %1 %2 ”。

```
AfxFormatString2(msg, IDS_HELLO_2, "and", "Goodbye");
AfxMessageBox(msg);
```

5) 要询问Yes或No问题，应该添加一个风格标志到 AfxMessageBox ( )中。

```
if (AfxMessageBox(msg, MB_YESNO)==IDYES)
{
}
```

其他可能的风格包括MB_ABORTRETRYIGNORE、MB_OKCANCEL、MB_RETRYCANCEL、MB_YESNO和MB_YESNOCANCEL。其他可能的选择测试包括 IDNO、IDABORT、IDCANCEL、IDIGNORE、IDOK和IDRETRY。

6) 要使一个按钮 (除了第一个按钮外)成为默认的按钮 (当用户按 Enters时被单击)，用MB_DEFBUTTON2或MB_DEFBUTTON3与风格标志进行“ 或 ”操作。

```
if (AfxMessageBox(msg,MB_YESNO|MB_DEFBUTTON2)==IDYES)
{
}
```

7) 要在消息框中显示除了感叹号以外的图标，可以用 MB_ICONSTOP、MB_ICONINFORMATION或MB_ICONQUESTION与风格标志进行“ 或 ”操作。

```
AfxMessageBox (msg,MB_ICONSTOP);
```

说明

用AfxFormatString1 ( )和AfxFormatString2 ( )使用的%1和%2标志可以以任意顺序出现，

也可以按照你每次用CwinApp::DoMessageBox ( )执行AfxMessageBox ( )；要截获应用程序创建的所有消息框，只要用ClassWizard重载应用程序类的DoMessageBox ( )函数；在该函数中，可以给消息添加一个标准标题、时间，甚至把消息保存到日志中。

CD说明

在CD上执行该工程时，在 WzdView.cpp的OnTestWzd ( )处设置一个断点；然后单击Test/Wzd菜单命令，在它创建不同消息框时，单步执行 OnTestWzd ( )。

输入说明

AfxMessageBox( )是一种让最新输入的应用程序立即显示消息的非常好的方法。通常，在一个窗口应用程序中，必须拥有或指定一个窗口以创建另一个窗口，所有应用程序的窗口属于应用程序主窗口，而它自身又属于桌面窗口，用这种方法，所有窗口可以被窗口管理器刷新。但是使用 AfxMessageBox( )，则类成员函数或者调用该类成员函数的静态函数不需要拥有一个窗口， AfxMessageBox（）自动查找并使用应用程序的主窗口作为它的物主窗口。因此，即使一个普通的数据类可以使用 AfxMessageBox ( )显示一个消息(虽然数据类通常不显示消息)。唯一的要求是把Stdafx.h包括在使用AfxMessageBox ( )的源文件中。如果那样有问题的话，可以用Windows API调用。

```
::MessageBox ( );
```

但这时必须包括“ Windows.h ”。

9.8 例45 对话条

目标

创建一个对话条，它具有和对话框一样的控件，但是像工具栏一样浮动并停放在应用程序主窗口的边上(见图9-8)。

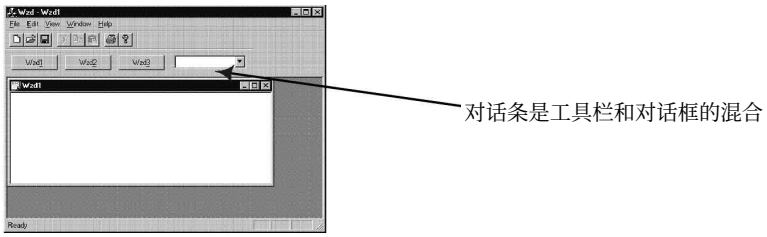

图9-8 一个对话条是对话框和工具栏的混合

策略

首先，用对话框编辑器创建一个对话框模板；然后，用 ClassWizard为模板创建一个对话框类；添加处理对话框控件的函数到该对话框中。同时，由于一个对话条不能接收WM_INITDIALOG消息，还将添加对话条最初创建时调用的 InitDialog ( )函数。添加一个菜单命令到应用程序的 View菜单，用以隐藏和显示对话条；要允许应用程序更新对话条中的控

件，将重载CMainFrame的OnCmdMsg ( )。

步骤

1. 创建对话框模板和对话框类

1) 用对话框编辑器创建一个对话框模板 (见例38)，尽量在水平或垂直方向上设计小一些，对话条的目的是像工具栏一样保留在屏幕上，因此，它不应该遮蔽任何必须的视图。如果不能使模板变小，则用一个无模式对话框替换它。

2) 用ClassWizard为该对话资源创建一个类 (见例39)。但是，创建对话条所必须的基类不在用ClassWizard创建一个类的选项中。因此选择 CDialog，然后在创建的资源文件中，把所有对CDialog类的引用改为CDialogBar。

2. 填充对话框类

1) 添加一个 InitDialog ( ) 函数到该对话框类中，并在那里初始化控件。也可以用SetWindowText ( )给予对话条一个名称。

```
SetWindowText ( "Wzd Dialog Bar");
```

该名称只有在对话条浮动时才显示。

2) 这时，可以用ClassWizard添加消息处理函数和成员变量到对话框类中，对于没有为创建一个消息处理函数的消息，将被自动传送到该对话框的父窗口类，本例中为 CMainFrame。用这种方法按钮控件可以象工具栏按钮一样起作用，因为单击它们，将使最终的命令消息在命令消息路径上的任何类中被处理 (即主框架类、文档类等)。

3) 有关对话条类的例子清单参见本节的 “ 清单 —— 对话条类 ”。

3. 添加对话条到主框架类

1) 把对话条类插入到CMainFrame类中。

```
CWzdDialogBar   m_WzdDialogBar;
```

2) 在CMainFrame类的OnCreate ( )函数中创建所有应用程序的工具栏之后，创建对话条。

```
if (!m_WzdDialogBar.Create(this, IDD_WZD_DIALOG,
    CBRS_TOP,           // control bar is initially at top of frame
//  CBRS_BOTTOM,    // control bar is initially at bottom of frame
//  CBRS_LEFT,          // control bar is initially on left side of frame
//  CBRS_RIGHT,         // control bar is initially on right side of frame -1) ||
    !m_WzdDialogBar.InitDialog())
    {
        TRACE0("Failed to create dialog bar\n");
        return -1;      // fail to create
    }

    m_WzdDialogBar.EnableDocking(
    CBRS_ORIENT_HORZ    // can only dock to top or bottom of frame
//  CBRS_ORIENT_VERT    // can only dock to left or right of frame
//  CBRS_ORIENT_ANY     // can dock to any side of frame
    );
    DockControlBar(&m_WzdDialogBar);
```

注意在创建对话条后立即调用对话条类的 InitDialog ( )函数。

3) 在对话条中的按钮没有与应用程序的命令消息路径相关联之前，它们保持无效。要使它们变成有效，用ClassWizard重载CMainFrame类的OnCmdMsg ( )函数，并在那里调用对话条类的OnCmdMsg ( )成员函数。

```
BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
   AFX_CMDHANDLERINFO* pHandlerInfo)
{
   if (m_WzdDialogBar.OnCmdMsg(nID,nCode,pExtra,pHandlerInfo))
   return TRUE;

   return CMDIFrameWnd::OnCmdMsg(nID, nCode, pExtra,
      pHandlerInfo);
}
```

一个对话条的关闭按钮不能真正的销毁该窗口，它只是隐藏它，本例中将添加一个菜单命令，用它来隐藏或显示对话条，该菜单命令显示在应用程序的 View菜单下面。

4. 添加一个对话条视图命令

1) 用菜单编辑器添加一个对话条命令到应用程序的 View菜单。

2) 用ClassWizard为该新命令添加一个命令处理函数和一个用户界面处理函数到CMainFrame类中，如下面一样填充这些处理函数。本例中 m_WzdDialogBar是对话条变量的名称。

```
void CMainFrame::OnViewDialogbar()
{
   ShowControlBar(&m_WzdDialogBar, (m_WzdDialogBar.GetStyle()
      & WS_VISIBLE) == 0, FALSE);
}

void CMainFrame::OnUpdateViewDialogbar(CCmdUI* pCmdUI)
{
   pCmdUI->SetCheck((m_WzdDialogBar.GetStyle() &
      WS_VISIBLE) != 0);

}
```

说明

- 前面已提到，如果不能把需要的控件压缩到一个窗口中的工具栏的尺寸，则可能要考虑创建一个无模式对话框。如果寻找的功能只是应用程序中的一个特定操作，这时也可以考虑使用一个无模式对话框。要创建一个无模式对话框，参见例 41。
- 当退出或重新进入应用程序时，例5中讨论过的SaveControlBars ( )和LoadControlBars ( )也将存储对话条的确切位置和大小。

CD说明

在CD上执行该工程时，可以看到一个对话条在窗口的顶部，它有四个控件，还可以看到该条可以像工具栏一样被移动和停放。

清单——对话条类

```
#if !defined(AFX_WZDDIALOGBAR_H__1EC84998_C589_11D1_9B5C_00AA003D8695__INCLUDED_)
#define AFX_WZDDIALOGBAR_H__1EC84998_C589_11D1_9B5C_00AA003D8695__INCLUDED_

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

// WzdDialogBar.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdDialogBar dialog

class CWzdDialogBar : public CDialogBar
{

// Construction
public:
    CWzdDialogBar(CWnd* pParent = NULL);   // standard constructor

    BOOL InitDialog();

// Dialog Data
    //{{AFX_DATA(CWzdDialogBar)
    enum {IDD = IDD_WZD_DIALOG};
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdDialogBar)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CWzdDialogBar)
    afx_msg void OnWzdButton1();
    afx_msg void OnWzdButton3();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

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

#endif // !defined(
   AFX_WZDDIALOGBAR_H__1EC84998_C589_11D1_9B5C_00AA003D8695__INCLUDED_)

// WzdDialogBar.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdDialogBar.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdDialogBar dialog


CWzdDialogBar::CWzdDialogBar(CWnd* pParent /*=NULL*/)
{
   //{{AFX_DATA_INIT(CWzdDialogBar)
      // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}


void CWzdDialogBar::DoDataExchange(CDataExchange* pDX)
{
   CDialogBar::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CWzdDialogBar)
      // NOTE: the ClassWizard will add DDX and DDV calls here
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CWzdDialogBar, CDialogBar)
   //{{AFX_MSG_MAP(CWzdDialogBar)
   ON_BN_CLICKED(IDC_WZD_BUTTON1, OnWzdButton1)
   ON_BN_CLICKED(IDC_WZD_BUTTON3, OnWzdButton3)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdDialogBar message handlers
```

```
BOOL CWzdDialogBar::InitDialog()
{
    UpdateData(FALSE);
    SetWindowText("Wzd Dialog Bar");
    return TRUE;
}

void CWzdDialogBar::OnWzdButton1()
{
    int i=0;
}

void CWzdDialogBar::OnWzdButton3()
{
    // TODO: Add your control notification handler code here

}
```