学生应知信息知识

Visual C++ 简明教程（下）

狄登峰 主编

目　　录

第八章　改进应用程序界面

在“消息与命令”一章中，初步介绍了在 MFC 应用程序中如何使用菜单和工具栏进行工作。实际上，有关改进程序界面方面的知识还有很多。本章中详细介绍如何改进应用程序的界面,使之具有更好的用户交互性。

8.1　控制条类

在 Windows 应用程序中，工具条、状态条或对话框条等是经常可见的，为用户提供了快速选择程序功能的能力。在 MFC 基本类库中，工具条、状态条和对话框条都是控制条的子类。MFC 对控制条提供了很好的支持，在下面的几节中，我们将比较详细地介绍控制条的有关知识。

8.1.1　控制条

控制条类（CControlBar）是工具条类（CToolBar）、状态条类（CStatusBar）和对话框条类（CDialogBar），以及 Visual C++ 6.0 中新增的集合条类（CReBar）的父类，同时，CControlBar 类由 CWnd 类派生，图 8-1 显示了有关 CControlBar 的继承关系。

图 8-1 CControlBar 类的继承关系

由于 CControlBar 类是 CWnd 类的子类，因此，控制条通常都是一个停泊在主框架窗口四周的小窗口。控制条可以包含基于 hWnd 句柄的控件，这些控件都是 Windows 窗口，可以产生和响应 Windows 消息；控制条也可以包含非基于 hWnd 句柄的项，这些项通常有应用程序类或主框架窗口类来管理。比如说，列表框和文本框就是基于 hWnd 句柄的控件，而状态条面板和图象按钮则是非基于 hWnd 句柄的项。

CControlBar 类为其子类提供了基本的功能，如在父窗口中的合适位置停泊等。由于控制条通常都是其父框架窗口的一个子窗口，因此，它是视图或 MDI 子窗口的“同胞”。控制条对象使用其父框架窗口的客户区域的信息来定位其自身，因此，其父框架窗口的客户区域将随着控制条的停泊位置的改变而改变。

8.1.2 工具条

工具条是经常使用的一种控制条，通常包含一系列的位图按钮，点击按钮等价与从菜单中选择相应的命令。工具条通常都停泊在框架窗口的顶部，但实际上，MFC 的工具条可以停泊在框架窗口的任何靠边的位置，也可

以浮动于自己的小框架窗口中，甚至可以浮动于应用程序的窗口中，可以改变其大小，可以用鼠标拖动到任何位置。图 8-2 和图 8-3 分别显示了工具条停泊在窗口顶部和以浮动窗口显示时的情形。

图 8-2 工具条停泊在窗口顶部　　图 8-3 工具条以浮动窗口显示

用户通过 CToolBar 类管理自己的工具条，但是从 MFC4.0 开始，CToolBar 类被重新定义为使用工具条控件（CToolBarCtrl 类）来建立。工具条控件被 Windows 95 和 Windows NT 3.51 或其更高的版本所支持。由于利用了操作系统级的支持，这一重新定义使得 MFC 可以用较少的代码来管理工具条，同时也增强了工具条的功能。用户可以 CToolBar 类的成员函数来操作工具条，或者通过获得一个 CToolBarCtrl 的指针而直接对工具条进行操作。

8.1.3 状态条

状态条是应用程序中经常使用的另外一种控制条，通常包含多个面板，用做文本输出或指示器。状态条一

般都停泊在框架窗口的底部，而且也无须改变其位置。图 8-4 是利用 AppWizard 生成的应用程序中的缺省的状态条。

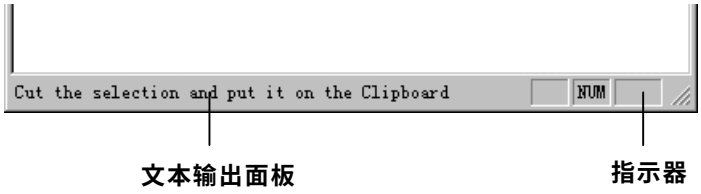

图 8-4 状态条停泊在窗口的底部

与工具条类似，状态条（CStatusBar）也被重新定义为使用状态条控件（CStatusBarCtrl）来建立。

8.1.4 对话框条

对话框条是一种基于对话框资源的控制条，其作用类似于一个无模式对话框，在对话框条中可以包含任何 Windows 控件，甚至可以设置这些控件的制表顺序。图 8-5 显示了一个典型的对话框条，这就是 Visual C++集成开发环境的 WizardBar。

图 8-5 典型的对话框条

8.1.5 集合条

集合条是 Visual C++ 6.0 中新增的，也是一种控制条。集合条中可以包含多种子窗口类型，通常都是一些控件，如按钮、文本编辑框等。集合条能将其子窗口在特定的位图背景上显示出来。使用集合条可以创建漂亮的程序界面。图 8-6 中显示了一个典型的集合条，这就

是 Internet Explorer 4.0 的界面。

图 8-6 典型的集合条

集合条(CReBar)也使用了集合条控件(CReBarCtrl)来建立。

在使用控制条的时候，很少直接使用 CControlBar 类或直接从 CControlBar 类派生子类，更多的是使用其子类或从其子类派生自己的类。

以上简要介绍了控制条类极其子类的一些基本的知识，从下一节开始具体介绍如何使用这些控制条。对于对话框条，我们仅介绍其创建方法，其使用方法可以参照无模式对话框；重点介绍的是工具条、状态条和集合条。

8.2 工具条和状态条

在“消息与命令”一章中，介绍了如何初步地使用工具条，仅仅介绍了如何修改工具条资源，如何将工具条按钮与菜单命令联系起来，如何改变工具条按钮的状态等等，而且使用的是 AppWizard 为用户创建的缺省的工具条。在本节中，将详细解释应用程序框架是如何建立工具条的，如何创建新的定制的工具条，如何控制工具条的显示与隐藏等知识，以及如何使用状态条，如何向状态条中添加新的面板，如何控制状态条的显示等。

8.2.1 缺省的工具条与状态条

在利用 AppWizard 生成应用程序的时候，只要在步骤 4 选中了 Docking toolbar 和 Initial status bar 选项，AppWizard 为用户生成的应用程序就拥有一个缺省的工具条和状态条。

1．建立新的项目

为了便于说明，需要建立一个例子程序 TestBars。利用 AppWizard 生成该应用程序框架。在创建过程中，按以下步骤操作：

（1）选择单文档界面，选中文档/视图结构支持，同时选择资源类型为英语。

（2）由于本程序不需要支持任何数据库技术，选择 None。

（3）由于本程序不需要支持任何形式的复合文档，选择 None；同时本程序也不需要包含 ActiveX 控件，故清除缺省对 ActiveX controls 的选中。

（4）本程序不涉及有关打印的知识，故清除 Print and print preview 选项；但需要缺省的工具条和状态条，故选中 Docking toolbar 和 Initial status bar 选项；工具条风格选择 Normal；其余接受缺省选择即可。

（5）接受缺省选择。

（6）单击 Finish 按钮完成设置，在新项目信息对话框中列出了本 TestBars 程序的有关已设定的信息，如图 8-7 所示。

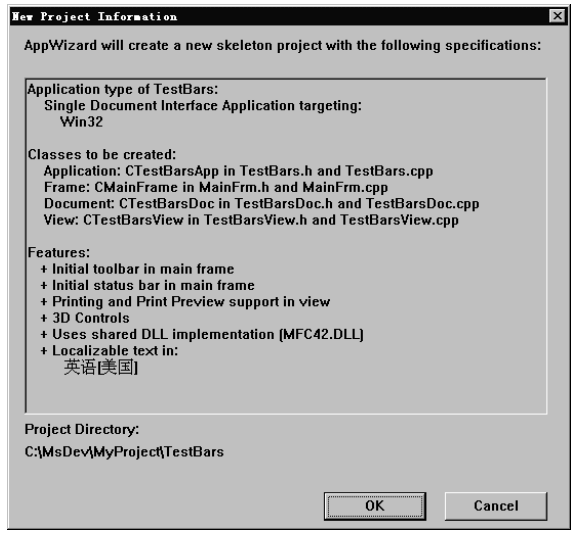

图 8-7 新项目信息

建立项目后，用户可以立即编译和运行 TestBars 应用程序，现在的程序版本就已经具有了缺省的工具条和状态条，如图 8-8 所示。

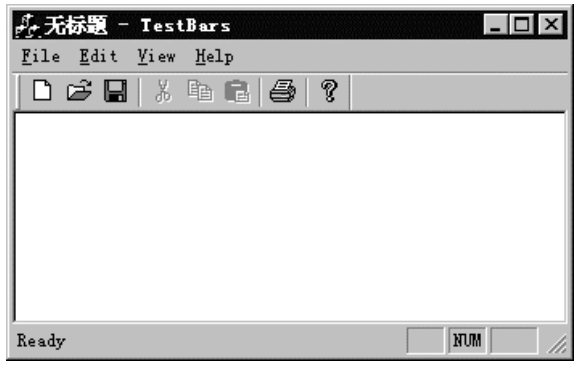

图 8-8 TestBars 程序已经拥有缺省的工具条和状态条

在“消息与命令”一章中讲述工具条的基本使用时提到，首先工具条有一个指定其按钮形式的资源，该资源实际上是一幅位图。但工具条是如何由资源创建为工具条对象并显示在窗口的顶部的呢？下面就来解释工具条的创建过程。

2．工具条的建立

每个工具条都对应着一个工具条对象，一般说来，工具条都隶属于其父窗口，这里就是主框架窗口，因此，工具条对象应该是主框架窗口的成员。状态条也是如此。在主框架窗口类的定义中，确实可以发现对工具条对象和状态条对象的定义，如程序清单 8-1 所示。

程序清单 8-1　CMainFrame 类中定义了工具条对象和状态条对象

```
class CMainFrame : public CFrameWnd
{
……
protected:    // control bar embedded members
      CStatusBar    m_wndStatusBar;
      CToolBar      m_wndToolBar;
……
};
```

在 CMainFrame 类中，定义了一个 CStatusBar 类对象 m_wndStatusBar，一个 CToolBar 类对象 m_wndToolBar。仅仅定义了工具条对象是不够的，还需要初始化该对象。由于工具条是主框架窗口的内嵌对象，因此 MFC 在创建主框架窗口的时候就创建了工具条对象，同样地，在销毁主框架窗口的时候就销毁了工具条。状态条的情况是类似的。

工具条的创建在 CMainFrame 类的 OnCreate()函数中进行。程序清单 8-2 中列出了 OnCreate()函数的代码。

程序清单 8-2　OnCreate()函数

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
WS_VISIBLE | CBRS_TOP|CBRS_GRIPPER| CBRS_TOOLTIPS | CBRS_FLYBY
|                CBRS_SIZE_DYNAMIC)                ||
    !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;          // fail to create
    }
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;          // fail to create
    }
    // TODO: Delete these three lines if you don't want the toolbar to
    //    be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
    return 0;
}
```

创建工具条对象的代码作为一个 if 语句的条件，以检查创建过程是否正确进行。如果不进行正确性检测的

话，可以将创建过程分解为调用两个函数，如下所示。

```
m_wndToolBar.CreateEx(this,TBSTYLE_FLAT,WS_CHILD|WS_VISIBLE | CBRS_TOP| CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ；
m_wndToolBar.LoadToolBar(IDR_MAINFRAME)；
```

首先调用 CToolBar 的 CreateEx()函数创建一个 Windows 工具条并将其与 m_wndToolBar 对象关联在一起。CreateEx()函数的原型如下：

```
BOOL CreateEx(CWnd* pParentWnd, DWORD dwCtrlStyle = TBSTYLE_FLAT, DWORD dwStyle = WS_CHILD | WS_VISIBLE | CBRS_ALIGN_TOP, CRect rcBorders = CRect(0, 0, 0, 0), UINT nID = AFX_IDW_TOOLBAR);
```

注意 CreateEx()函数的参数。pParentWnd 是一个指向父窗口的指针，这里使用了关键字 this，即将当前对象，也就是主框架窗口作为本工具条的父窗口。

参数 dwCtrlStyle 指定在建立内嵌 CToolBarCtrl 对象时所用的风格。缺省的风格是 TBSTYLE_FLAT。

参数 dwStyle 指定工具条的风格。dwCtrlStyle 参数与 dwStyle 参数可取的值比较多，而且相互关联，因此本书中只对使用到的风格略作说明，对于其他可选风格的详细信息，用户在使用时请查阅 Visual C++的联机手册。

参数 rcBorders 指定工具条窗口边框的宽度。缺省值为 CRect(0,0,0,0)，即工具条窗口没有边框。

参数 nID 指定工具条的标识符。

AppWizard 创建的缺省工具条使用的风格的说明列于表 8-1 中。

表 8-1 缺省工具条使用到的风格

所用风格	说 明
TBSTYLE_FLAT	创建一个平面工具条
WS_CHILD	创建一个子窗口
WS_VISIBLE	创建的工具条初始即可见
CBRS_TOP	工具条位于父窗口的顶部
CBRS_GRIPPER	工具条具有控制棒
CBRS_TOOLTIPS	工具条按钮具有提示信息（鼠标经过时显示在一个小窗口中）
CBRS_FLYBY	工具条按钮具有提示信息（显示在状态条的左部）
CBRS_SIZE_DYNAMIC	工具条的大小动态可调

除了 CreateEx()函数之外，CToolBar 类还有另外的一个函数 Create()也可以创建工具条对象，其原型如下：

```
BOOL Create( CWnd* pParentWnd, DWORD dwStyle = WS_CHILD |
WS_VISIBLE | CBRS_TOP, UINT nID = AFX_IDW_TOOLBAR );
```

其参数的意义大致同 CreateEx()函数。

在创建了工具条对象后，就要将其与相应的工具条资源联系起来，这项工作由 CToolBar 类的 LoadToobar()函数完成。LoadToobar()函数的原型如下：

```
BOOL LoadToolBar( UINT nIDResource );
```

参数 nIDResource 指定所用资源的标识符。

在显示工具条之前还有一项工作，就是确定工具条的停泊位置。与此有关的代码是：

```
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);
```

确定工具条的停泊位置需要三个步骤：

（1）确定工具条可以停泊的位置，也就是说，确定

工具条本身希望停泊在主框架窗口的位置。至于能否停泊在指定的位置，还需要主框架窗口的支持。由CToolBar类的EnableDocking()函数进行设置，这里使用CBRS_ALIGN_ANY作为参数说明工具条期望停泊的位置可以是窗口的任一边。

（2）确定主框架窗口中可以停泊工具条的位置。由CWnd类的EnableDocking()函数指定，这里使用CBRS_ALIGN_ANY作为参数说明主框架窗口允许工具条停泊在其窗口中的任一边。

（3）停泊工具条在指定位置，由CWnd类的DockControlBar()函数完成。

至此已经完成了工具条的对象创建、资源关联和位置确定等所有的准备工作，这样在程序显示主框架窗口的时候，工具条也就被显示出来。

3．状态条的建立

状态条的建立过程与工具条是类似的。在缺省的情况下，状态条的建立由以下代码完成：

```
m_wndStatusBar.Create(this);
m_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));
```

首先是建立状态条对象。CStatus类的Create()函数的原型如下：

```
BOOL Create( CWnd* pParentWnd, DWORD dwStyle = WS_CHILD | WS_VISIBLE | CBRS_BOTTOM, UINT nID = AFX_IDW_STATUS_BAR );
```

其参数的意义与CToolBar类的Create()函数类似。需要注意的是建立状态条对象后如何设置状态条指示器，这由CStatusBar类的SetIndicators()函数完成，该函数的原型如下：

```
BOOL SetIndicators( const UINT* lpIDArray, int nIDCount );
```

参数 lpIDArray 是一个指向指示器标识符数组的指针，参数 nIDCount 是数组元素的数目。在缺省状态下，AppWizard 建立的标识符数组具有四个元素，在 CMainFrame 类的源文件的头部可以找到如下定义：

```
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

数组 indicators 包含的四个元素都是预先定义好的标识符。

由于状态条一般都停泊在主框架窗口的底部，因此对状态条而言，没有确定停泊位置的问题。

现在用户应该清楚，缺省的工具条和状态条是如何被创建出来的。在下面的部分，将通过建立自己的工具条与向状态条中添加指示器，更深入地介绍工具条与状态条的操作。

8.2.2 创建自己的工具条

在比较大的 Windows 应用程序中，往往都使用了多个工具条，因此，在这样的情况下，仅仅使用缺省的工具条是不够的。其实，创建自己的工具条也不是一项困难的工作，只要用户已经真正理解缺省工具条的创建过程，实现自己的工具条就是很自然的过程。

我们计划在 TestBars 程序中加入一个新的绘图工具

条，包含一些基本的绘图功能。

1．创建工具条资源

首要的任务是创建一个工具条资源。在开发环境的 Insert 菜单中选择 Resource 命令，弹出 Insert Resource 对话框，如图 8-9 所示。

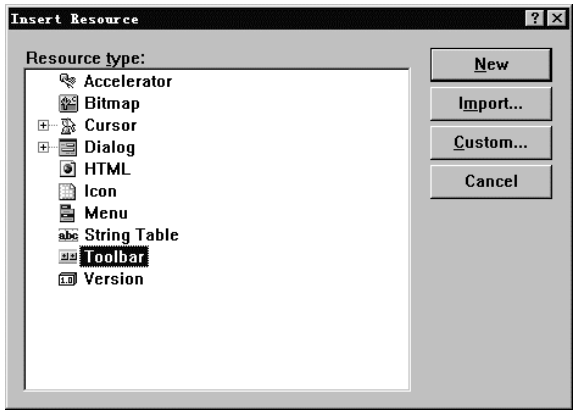

图 8-9 Insert Resource 对话框

在对话框中选择 Toolbar 类型，并单击 New 按钮建立新的工具条资源。使用工具条编辑器建立工具条按钮，完成后的工具条如图 8-10 所示。

图 8-10 完成的新工具条

打开工具条的属性对话框，修改工具条的标识符为 IDR_DRAWTOOLS，如图 8-11 所示。