第7章　工具栏和状态栏

应用程序向导自动地为应用程序创建一个常用的工具栏，应用程序底部的状态栏也有个标准问题，只有Shift和Num键的状态在此处更新。本章中的例子讨论怎样定制这些栏。

例22　使用工具栏编辑器　使用工具栏编辑器(ToolBar Editor)修改工具栏，以及创建新的工具栏。

例23　启用和禁用工具栏按钮　根据应用程序的状态，灰化和正常化工具栏按钮。

例24　为工具栏按钮添加字，在工具栏的按钮上添加文字，这样可表明按钮作用。

例25　非标准工具栏大小　讨论改变工具栏按钮大小的方法。

例26　保持工具栏按钮按下　讨论保持工具栏按钮按下的方法，以指示该功能正在使用中。

例27　保持工具栏按钮组中的一个按钮按下　与例26类似，讨论在任一给定时刻，只允许工具栏按钮组中的一个按钮按下。

例28　为工具栏添加非按钮控件　添加一个组合框到工具栏。

例29　修改应用程序的状态栏　用字符串表编辑器(String Table Editor)修改应用程序的状态栏。

例30　更新状态栏窗格　使用ClassWizard创建一个类，更新显示状态窗格上的文本消息。

例31　为状态栏添加其他控件　添加一个按钮和进度指示控件到一个状态栏。

7.1　例22　使用工具栏编辑器

目标

使用工具栏编辑器编辑应用程序的状态栏，或添加附加的工具栏到应用程序中。

注意　给应用程序添加的任何新的工具栏按钮，初始时显示灰色并且是无效的，直到添加一个处理该按钮的命令处理函数为止。例 13显示了怎样添加一个菜单命令处理函数；添加工具栏命令处理函数的方法与之相同。

步骤

1. 应用Developer Studio创建一个新的工具栏

要创建一个新的工具栏，单击 Developer Studio的Insert/Resource菜单命令，打开Insert/Resource对话框，然后从列表中选取ToolBar，并单击New。

2. 应用工具栏编辑器编辑一个工具栏

1) 编辑一个已有的工具栏，在应用程序资源的工具栏文件夹中找到它的 ID，并双击它，这将打开ToolBar Editor。

2) 用工具栏编辑器添加一个新的按钮到工具栏，先用鼠标拖动工具栏末尾的空白工具栏

按钮，放到新按钮的位置，然后用位图编辑器添加一个图像到该按钮上。要打开该按钮的属性对话框，双击它在窗口上方的图像即可 (见图7-1)。

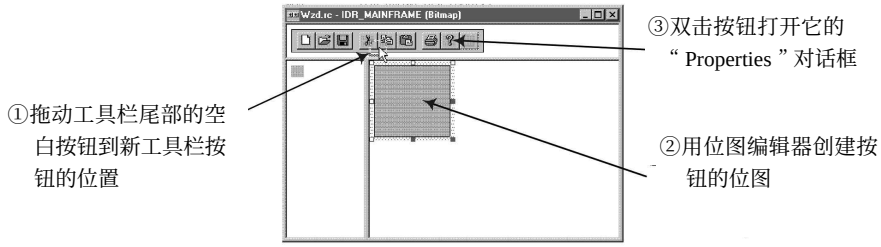

图7-1 用工具栏编辑器添加按钮

3) 添加一个间隔(spacer)到该工具栏(按钮间的空白区)。首先，确定当前哪个按钮正好在放置间隔的位置处，并用鼠标抓住它；然后稍微往右拖动该按钮，并放开它，此时应该出现一个空白间隔。要删除一个间隔，稍微拖动当前在它右边的按钮，并覆盖它。要删除一个按钮，只要把它拖离工具栏即可。如果出现误操作，则用 Developer Studio的undo命令取消误操作。

4) 每个按钮的属性对话框显示与菜单项的属性框相同的 ID和Prompt编辑框，关于它们的意义参见例12。尺寸编辑框(size edit box)不仅允许改变该按钮的大小，而且可以改变每个按钮的大小，参见例25关于创建非标准工具栏按钮。

5) 为工具栏按钮添加一个命令处理函数，与添加一个菜单命令处理函数相同 (参见例13)。实际上，工具栏按钮是一个菜单命令的图形表示，没有相应菜单的工具栏按钮几乎不存在，尽管这是传统意义上的规定。技术上可以添加一个没有相应菜单的工具栏按钮。

3. 添加一个新的工具栏到应用程序

1) 使用Developer Studio和工具栏编辑器创建一个新的工具栏后，必须添加下面的代码到CMainFrame类的Create( )函数中，正好在添加原始工具栏的代码后面。

```
if (!m_wndToolBar1.Create(this) ||
    !m_wndToolBar1.LoadToolBar(IDR_TOOLBAR1))
{
    TRACE0("Failed to create toolbar 1\n");
    return -1;      // fail to create
}
m_wndToolBar1.SetBarStyle(m_wndToolBar1.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
m_wndToolBar1.EnableDocking(CBRS_ALIGN_ANY);
```

2) 新的工具栏一排接一排地垂直添加到应用程序的窗口中，如果使工具栏一个接一个水平地添加到窗口中，可以用下面的代码：

```
DockControlBarLeftOf(&m_wndToolBar1,&m_wndToolBar);
```

参见7.1.5节DockControlBarLeftOf( )的清单。

3) 浮动工具栏可以从应用程序窗口中拖下来，“ 漂浮 ”在它自己的窗口里 (事实上，是由MFC的CMiniFrameWnd类控制一个微型框架窗口)。如果用户按下浮动工具栏上的关闭按钮，则工具栏不像其他窗口一样被销毁，而只是被隐藏起来。要允许用户把工具栏拖回到原处，需要在应用程序的视图命令中，为每个新工具栏添加一个附加菜单命令，每个这样的命令只是重显一个隐藏的工具栏，如下面所示。有关怎样添加一个菜单命令参见例 13。

```
void CMainFrame::OnToolDecision()
{
   ShowControlBar(&m_wndToolBar1,m_wndToolBar1.GetStyle() &
      WS_VISIBLE) == 0, FALSE);
}
```

4) 还可以在当前可视的工具栏菜单项边上，放置一个复选标志。参见例 16中有关怎样添加一个消息处理函数，以添加复选标志到菜单命令中。然后，可以用下面的代码填充该处理函数。

```
void CMainFrame::OnUpdateToolBar1(CCmdUI* pCmdUI)
{
   pCmdUI->SetCheck((m_wndToolBar1.GetStyle() & WS_VISIBLE) != 0);
}
```

说明

- 除了用DockControlBarLeftOf( )之外，还可以在CToolBar的Create( )的调用变量中，指定工具栏的确切位置。例如，下面的代码把一个工具栏放置在初始位置（ x，y）为（100,0）的地方。然而， CMainFrame可能把它移到别的地方。

```
if (!m_wndToolBar1.Create(this, 100, 0) ||
   :  :  :
```

- 最后一步显示了在开始时，怎样把一个工具栏放置在另一个的左边。但是在与应用程序会话期间保存用户放置一个工具栏的确切位置，请参见例 5。
- 添加一个工具栏命令处理函数到应用程序，与添加一个菜单命令处理函数的方法相同(见例13)。虽然可以用m_bAutoMenuEnable启用当前没有命令处理函数的菜单命令，但它不能启用一个没有处理函数的工具栏按钮。

CD说明

在CD上没有本例的相应工程。

清单——DockControlBarLeftOf( )

```
void CMainFrame::DockControlBarLeftOf(CToolBar* Bar1,
   CToolBar* Bar2)
{
   CRect rect;
   RecalcLayout();
   Bar2->GetWindowRect(&rect);
   rect.OffsetRect(1,0);
   DockControlBar(Bar1,AFX_IDW_DOCKBAR_LEFT,&rect);
}
```

7.2 例23 启用和禁用工具栏按钮

目标

启用或禁用一个工具栏按钮 (禁用的按钮显示灰色)。

策略

与菜单命令一样，在用ClassWizard添加一个命令处理函数到工具栏按钮之前，它将显示禁用状态(见例13)。否则，通过在类中添加一个用户界面处理函数，根据应用程序的某些条件，可以有条件地启用和禁用一个工具栏按钮。

步骤

添加一个用户界面处理函数

1) 按照例13关于添加一个菜单命令处理函数所示的步骤，但是这时选用 UPDATE_COMMAND_UI，而不是COMMAND。

2) 添加下面的代码到新的处理函数中，如果要启用菜单项，则 m_bWzd为TRUE。

```
void CWzdView::OnUpdateWzdType(CCmdUI* pCmdUI)
{
   pCmdUI->Enable(m_bWzd);
}
```

说明

- 通过ID与工具栏按钮相关联的菜单项，随按钮一起被启用或禁用。
- 工具栏按钮的状态不被更新，并且，该过程只在应用程序空闲时被调用。如果不愿等待那么长时间，可以取得一个指向该工具栏类的指针 (可能在CMainFrame里)，并调用它的UpdateWindow( )成员函数。
- 有关MFC怎样更新用户界面的详细内容，参见第3章及后面的例子。

CD说明

在CD上执行该工程时，将看到一个新工具栏按钮，它已被启用。

7.3 例24 为工具栏按钮添加字

目标

创建一个在每个按钮上包括一个或两个字的工具栏(见图7-2)。

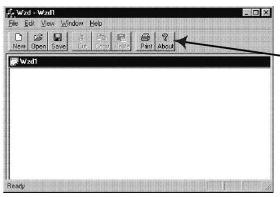

图7-2 添加一组文本串，以创建带标题的工具栏按钮

策略

使用CToolBarCtrl::AddStrings( )给工具栏添加一组文本串 (text strings)；然后用CToolBarCtrl::InsertButton( )使每一个按钮与一个串相关联；继而用工具栏编辑器创建和编辑应用程序的工具栏，动态地使一个工具栏与它的串相关联。本例中，用 ClassWizard把该功能封装到工具栏类中。

步骤

1. 创建一组工具栏标题

1) 用String Editor添加一组新串到应用程序中，这些串是显示在工具栏按钮下的短标题，每个按钮对应一个串。

2) 在CMainFrame类中定义一个数组，用它使一个工具栏的命令ID与上一步创建的串标题的ID相关联。本例中我们将创建一个叫做TOOLTEXT的结构，用它进行关联，该结构只包含两个变量：一个为命令ID，另一个为串ID。

```
static TOOLTEXT tooltext[] =
{
   ID_FILE_NEW , IDS_FILE_NEW,
   ID_FILE_OPEN, IDS_FILE_OPEN,
   ID_FILE_SAVE, IDS_FILE_SAVE,
   ID_EDIT_CUT , IDS_FILE_CUT,
   ID_EDIT_COPY, IDS_FILE_COPY,
   ID_EDIT_PASTE, IDS_FILE_PASTE,
   ID_FILE_PRINT, IDS_FILE_PRINT,
   ID_APP_ABOUT , IDS_APP_ABOUT,
};
#define TOOLTEXT_NUM (sizeof(tooltext)/sizeof(TOOLTEXT))
```

现在，创建一个新的工具栏类，在该类中添加一个叫做LoadToolBarEx( )的函数，它将装入一个工具栏资源，然后动态地为每个按钮添加一个标题。

2. 创建一个新的工具栏类

1) 用ClassWizard创建一个派生于CToolBarCtrl的新工具栏类。实际上，该类应派生于CToolBar，但是ClassWizard不支持CToolBar，因此，在新类中，必须手工改变对CToolBarCtrl的引用为对CToolBar的引用。

2) 在类中添加一个叫做LoadToolBarEx( )的新成员函数，把前面的文本串表传递给它。在该函数开始处，用CToolBar::LoadToolBar( )装入一个工具栏资源。

```
BOOL CWzdToolBar::LoadToolBarEx(UINT nID,
   TOOLTEXT *pToolText,int nCnt)
{
   BOOL bRet;
   if (bRet=LoadToolBar(nID))
   {
```

3) 工具栏资源一旦装入，将在前面创建的标题表中循环。在那里，找到它所指向的工具栏命令ID；对于已经找到的，从串资源中装入该工具栏按钮的串。

```
for (int i=0;i<nCnt;i++)
{
   // find button
   TBBUTTON tb;
   int inx=CommandToIndex(pToolText[i].idCommand);
   GetToolBarCtrl().GetButton(inx,&tb);
   // get text for button
   CString str;
   str.LoadString(pToolText[i].idString);
```

4) 接着，添加装入的串到工具栏控件。

```
// add a second NULL to string for AddStrings()
   int nLen = str.GetLength() + 1;
```

```
    TCHAR * pStr = str.GetBufferSetLength(nLen);
    pStr[nLen] = 0;

    // add new button using AddStrings
    tb.iString=GetToolBarCtrl().AddStrings(pStr);
    str.ReleaseBuffer();

    // (no ModifyButton() function)
    GetToolBarCtrl().DeleteButton(inx);
    GetToolBarCtrl().InsertButton(inx,&tb);
}
```

工具栏控件返回一个指向串的索引，并可以用该按钮存储。注意到 AddStrings( )在最后的串后面，还需要两个零(null)字符，这使我们必须添加一个额外的零字符。

还应注意，因为没有ModifyButton( )函数，我们必须删除和重新插入每个按钮。

5) 用CToolBar::SetSizes( )使该工具栏中的按钮变大，以便它们能够适应新的标题。SetSizes( )的第一个变量表示每个按钮要求多少位图图像，第二个变量表示实际上使该按钮多大。

```
// make buttons larger to handle added text
CSize sizeImage(16,15);
CSize sizeButton(35,35);
SetSizes(sizeButton, sizeImage);
```

6) 要查看工具栏类的详细清单，参见本节的“ 清单 —— 工具栏类 ”。

3. 执行新的工具栏类

在CMainFrame中，用该新的工具栏类替换应用程序的初始工具栏类。

```
// call the new CWzdToolBar::LoadToolBarEx() function
if (!m_wndToolBar.Create(this) ||
    !m_wndToolBar.LoadToolBarEx(IDR_MAINFRAME,
    (TOOLTEXT*)&tooltext,TOOLTEXT_NUM))
{
    TRACE0("Failed to create toolbar\n");
    return -1;                          // fail to create
}
```

说明

当鼠标在一个工具栏按钮上停留时间超过半分钟时，本例中添加的标题将添加到显示的气泡帮助标题中。

CD说明

在CD上执行该工程时，将看到工具栏按钮已被放大，并为每个按钮添加了一个描述词。

清单 —— 工具栏类

```
#if
!defined(AFX_WZDTOOLBAR_H__3A5CD903_E412_11D1_9B7D_00AA003D8695__INCLUDED_)
#define AFX_WZDTOOLBAR_H__3A5CD903_E412_11D1_9B7D_00AA003D8695__INCLUDED_

#if _MSC_VER >= 1000
```

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

// WzdToolBar.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar window

typedef struct t_TOOLTEXT {
    UINT idCommand;
    UINT idString;
} TOOLTEXT;

class CWzdToolBar : public CToolBar
{

// Construction
public:
    CWzdToolBar();

// Attributes
public:

// Operations
public:
    BOOL LoadToolBarEx(UINT nID,TOOLTEXT *pToolText,int nCnt);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdToolBar)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdToolBar();

    // Generated message map functions
protected:
    //{{AFX_MSG(CWzdToolBar)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
```

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

#endif
// !defined(AFX_WZDTOOLBAR_H__3A5CD903_E412_11D1_9B7D_00AA003D8695__INCLUDED_)

// WzdToolBar.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdToolBar.h"

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

//////////////////////////////////////////////////////////////////////
// CWzdToolBar

CWzdToolBar::CWzdToolBar()
{
}

CWzdToolBar::~CWzdToolBar()
{
}


BEGIN_MESSAGE_MAP(CWzdToolBar, CToolBar)
   //{{AFX_MSG_MAP(CWzdToolBar)
      // NOTE - the ClassWizard will add and remove mapping macros here.
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////
// CWzdToolBar message handlers

BOOL CWzdToolBar::LoadToolBarEx(UINT nID,TOOLTEXT *pToolText,int nCnt)
{
   BOOL bRet;
   if (bRet=LoadToolBar(nID))
   {
      // loop through tooltext adding text to buttons
      for (int i=0;i<nCnt;i++)
      {
         // find button
```

```
            TBBUTTON tb;
            int inx=CommandToIndex(pToolText[i].idCommand);
            GetToolBarCtrl().GetButton(inx,&tb);

            // get text for button
            CString str;
            str.LoadString(pToolText[i].idString);

            // add a second NULL to string for AddStrings()
            int nLen = str.GetLength() + 1;
            TCHAR * pStr = str.GetBufferSetLength(nLen);
            pStr[nLen] = 0;

            // add new button using AddStrings
            tb.iString=GetToolBarCtrl().AddStrings(pStr);
            str.ReleaseBuffer();

            // (no ModifyButton() function)
            GetToolBarCtrl().DeleteButton(inx);
            GetToolBarCtrl().InsertButton(inx,&tb);
        }

        // make buttons larger to handle added text
        CSize sizeImage(16,15);
        CSize sizeButton(35,35);
        SetSizes(sizeButton, sizeImage);
    }
    return bRet;
}
```

7.4 例25 非标准工具栏大小

目标

给工具栏添加更大的按钮(见图7-3)。

策略

用工具栏编辑器增大工具栏按钮的大小，然后，在每个按钮表面重画一幅图像，或从别处剪切并粘贴一幅新图像。接着，讨论CToolBar::SetSizes( )，它允许用程序设计方法改变工具栏按钮的大小。

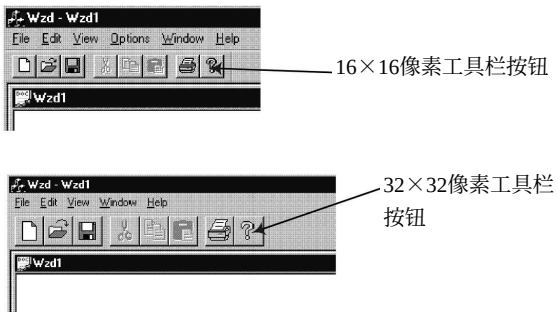

图7-3 用工具栏编辑器放大工具栏按钮

步骤

1. 用工具栏编辑器改变工具栏按钮的大小

1) 在应用程序资源的ToolBar文件夹中，找到要编辑的工具栏 ID，双击它以打开ToolBar Editor，选择Developer Studio的Image/Grid Settings…菜单命令以打开Grid Setting(网格设置)对话框，在那里可以设置新工具栏按钮的位图部分的大小，本例中为 25×25像素。

2) 现在，拖动工具栏按钮右下角的移动框，直到它大小合适为止。

3) 用工具栏编辑器把一个已有的按钮图像放大到新的大小。首先，用剪切工具选取图像；接着，拖动移动框，直到图像足够大；然后，修饰由此产生的图像。只是放大图像往往不能得到好的效果，因为没有足够像素自动创建一幅平滑图像，但可以从别的资源那里，剪切并粘贴一幅更大的图像到该按钮上。

4) 放大图像如上所述；而对于工具栏的放大，工具栏资源会自动告诉应用程序，使工具栏按钮变大以适应该新图像的大小。

2. 用CToolbar::SetSizes( )改变工具栏按钮大小

有时，可能需要在运行时放大按钮，例 24就是这样的一个例子。它要求放大工具栏按钮，以适应一个标题。要在运行时放大按钮，可以用下面的代码：

```
// set toolbar size to 32 by 32 pixels
SIZE sizeButton, sizeImage;
sizeImage.cx = 25;
sizeImage.cy = 25;
sizeButton.cx = sizeImage.cx + 7; //allow for spacing around image
sizeButton.cy = sizeImage.cy + 7;
m_wndToolBar.SetSizes(sizeButton, sizeImage);
```

说明

为了方便，CD中包括了一个标准工具栏的放大版工具栏。要使用应用程序中的该图像，在另一个Developer Studio中打开该例子的工程，然后用剪切和粘贴的方法，把那些图像装到新的应用程序中。

CD说明

在CD上执行该工程时，可以看到工具栏按钮和图标都已被放大。

7.5 例26 保持工具栏按钮按下

目标

在工具栏按钮被单击后，使它保持按下状态 (见图7-4)。

图7-4 添加一个用户界面处理函数以保持按钮按下

策略

用ClassWizard添加一个用户界面处理函数，它允许保持一个按钮按下。

步骤

添加一个用户界面处理函数

按照例24中的步骤，给应用程序添加一个用户界面处理函数，然后把下面的代码添加到该新处理函数中。

```
void CWzdView::OnUpdateWzdButton(CCmdUI* pCmdUI)
{
   // this same command checks any menu items too
   pCmdUI->SetCheck(m_bWzd);
}
```

说明

- 实际上，一个按下的按钮是对等于选中的菜单项的工具栏的，当选中一个菜单项时，便得到了一个按下的工具栏按钮。 CCmdUI类对象被重载，以便当 MFC更新一个工具栏按钮，而不是一个菜单项时，不同的 SetCheck( )函数被每个菜单项和工具栏按钮调用。
- 工具栏按钮的状态在应用程序空闲时更新 (因此该例程被调用)。如果不愿等待那么长时间，可以获得一个指向工具栏类的指针 (可能在 CMainFrame中)，并调用它的 UpdateWindow( )成员函数。
- 有关MFC更新用户界面的详细讨论参见第 3章。

CD说明

在CD上执行该工程时，可以看到一个永久按下的新按钮。

7.6 例27 保持工具栏按钮组中一个按钮按下

目标

保持工具栏按钮组中只有一个按钮按下，以表示应用程序处于一个特定的模式 (见图7-5)。

策略

使用ClassWizard，给工具栏按钮组中的每一个按钮添加一个界面消息处理函数，然后调用CCmdUI::SetRadio( )，以按下合适的按钮。

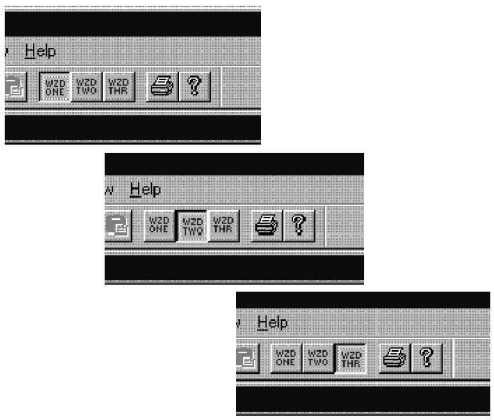

图7-5 给一组工具栏中的每一个工具栏添加一个界面消息处理函数，以保持该组中的一个按钮按下

步骤

1. 为组中的每个成员添加一个菜单命令处理函数

使用ClassWizard为组中每个按钮添加一个命令处理函数 (见例13)，用这些处理函数为应用程序设置合适的模式。

```
void CWzdView::OnWzd1Button()
{
   m_nWzdMode=CWzdView::mode1;
```

```
}

void CWzdView::OnWzd2Button()
{
   m_nWzdMode=CWzdView::mode2;
}

void CWzdView::OnWzd3Button()
{
   m_nWzdMode=CWzdView::mode3;
}
```

2. 为组中的每个成员添加一个用户界面处理函数

用ClassWizard为组中每个按钮添加一个用户界面处理函数 (见例15)。在每个处理函数中，根据当前模式，用SetRadio( )告知一个按钮已被按下。

```
void CWzdView::OnUpdateWzd1Button(CCmdUI* pCmdUI)
{
   pCmdUI->SetRadio(m_nWzdMode==CWzdView::mode1);
}

void CWzdView::OnUpdateWzd2Button(CCmdUI* pCmdUI)
{
   pCmdUI->SetRadio(m_nWzdMode==CWzdView::mode2);
}

void CWzdView::OnUpdateWzd3Button(CCmdUI* pCmdUI)
{
   pCmdUI->SetRadio(m_nWzdMode==CWzdView::mode3);
}
```

说明

对于工具栏按钮来讲，使用SetCheck( )或SetRadio( )没有区别，两者都能保持一个按钮按下；但是，如果这些按钮用来与一组菜单项关联，则应使用 SetRadio( )，以便在菜单项边上显示一个点。

有关MFC更新用户界面的详细内容参见第3章。

CD说明

在CD上执行该工程时，单击工具栏中每一个 Wzd按钮，按下一个按钮的同时，所有其他的按钮保持不按下。

7.7 例28 为工具栏添加非按钮控件

目标

添加一个组合框或其他控件窗口到工具栏 (见图7-6)。

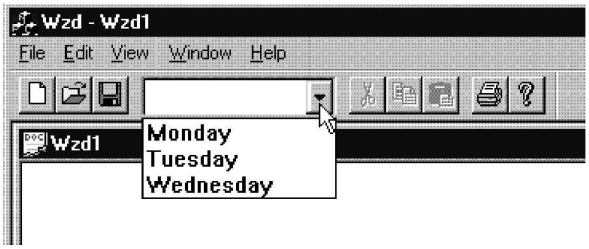

图7-6 添加一个组合框到工具栏中

策略

要放置别的控件窗口或任何子窗口到一个工具栏中，首先必须通知工具栏，在要放置窗口的地方放置一个足够大的间隔，然后在该位置手工创建一个控件窗口；要保持使用 ToolBar Editor编辑该工具栏的能力，必须创建一个特殊的工具栏按钮 ID，当它被一个工具栏按钮使用时，可以用软件改变成一个组合框。该功能被封装到工具栏类中。

步骤

1. 创建一个新的工具栏类

1) 用ClassWizard创建一个从 CToolBarCtrl派生的工具栏类，然后用文本编辑器（ Text Editor）在创建的.cpp和.h文件中用CToolBar替换CToolBarCtrl。

2) 添加一个叫做LoadToolBarEx( )的新函数到该类。

```
BOOL CwzdToolBar::LoadToolBarEx (UINT id)
```

3) 在LoadToolBarEx( )中，开始处用CToolBar::LoadToolBar( )装入一个工具栏资源：

```
// load toolbar info
BOOL bRet;
bRet=CToolBar::LoadToolBar(id);
```

4) 接着，寻找一个特殊的按钮命令 ID，本例中称为IDC_WZD_COMBO。一个使用工具栏编辑器创建的具有该 ID的工具栏按钮，将被该新的工具栏类转变为一个组合框。在工具栏中寻找该ID，使用下面代码：

```
// find where our combo box will go
int pos=CommandToIndex(IDC_WZD_COMBO);
```

5) 然后，用下面的代码把该工具栏按钮转变为一个非常宽的间隔。

```
// covert button in toolbar into a spacer for our combo box
SetButtonInfo(pos,IDC_WZD_COMBO,TBBS_SEPARATOR,COMBOLEN);
```

这里的COMBOLEN是间隔以像素为单位的大小。

6) 在LoadToolBarEx( )中的最后一步是，在创建间隔的地方创建一个控件窗口。

```
// create combo box
CRect rect;
GetItemRect(pos,&rect);
rect.bottom+=COMBODROP;    //how far will combo drop down?
m_ctrlWzdCombo.Create(WS_CHILD|WS_VISIBLE|CBS_DROPDOWNLIST,
   rect, this, IDC_WZD_COMBO);
```

7) 还应在该工具栏类中处理所有来自该组合框的控件通知，但是必须手工添加它们。

本节的“清单 —— 工具栏类”显示了怎样完成该步工作。

2. 执行新的工具栏类

1) 在CMainFrame中，用该新的工具栏类替换任何一个当前使用的工具栏类；在本例中，把所有对 CToolBar的引用变为 CWzdToolBar，然后在 CMainFrame( )的OnCreate( )中把LoadToolBar( )改为LoadToolBarEx( )。

2) 使用Toolbar Editor在想要插入一个组合框的位置为工具栏添加一个新按钮，并赋予该按钮前面指定的特殊的命令 ID，本例中是IDC_WZD_COMBO。应用程序中的多个工具栏可以共享该相同的特殊按钮ID。

说明

- 还可以用该方法添加其他控件到一个工具栏中，只要它们的尺寸能放进工具栏中，包括按钮(下压、复选和单选)、进度指示控件、编辑框、静态控件和新的日期 /时间控件。
- 一个工具栏最初看起来好象是一个具有一些子窗口按钮的父窗口，事实上它是一个自身绘制了许多“按钮”的大的控件窗口，并为这些按钮处理所有的鼠标动作。

CD说明

在CD上执行该工程时，将在工具栏中看到一个新的组合框。

清单 —— 工具栏类

```
#if
 !defined(AFX_WZDTOOLBAR_H__27649E31_C807_11D1_9B5D_00AA003D8695__INCLUDED_)
#define AFX_WZDTOOLBAR_H__27649E31_C807_11D1_9B5D_00AA003D8695__INCLUDED_

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

// WzdToolBar.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdToolBar window

class CWzdToolBar : public CToolBar
{

// Construction
public:
    CWzdToolBar();

    BOOL LoadToolBarEx(UINT id);

// Attributes
public:

// Operations
```

```
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdToolBar)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdToolBar();

    // Generated message map functions
protected:
    //{{AFX_MSG(CWzdToolBar)
    //}}AFX_MSG
    afx_msg void OnDropdownCombo();
    afx_msg void OnCloseupCombo();

    DECLARE_MESSAGE_MAP()
private:
    CString        m_sSelection;
    CComboBox        m_ctrlWzdCombo;
};

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

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

#endif
// !defined(AFX_WZDTOOLBAR_H__27649E31_C807_11D1_9B5D_00AA003D8695__INCLUDED_)

// WzdToolBar.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdToolBar.h"

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

#define COMBOPOS 3                // position of combo box in toolbar
#define COMBOLEN 120              // length of combo box in pixels
#define COMBODROP 100             // length of drop of combo box in pixels
```

```
//////////////////////////////////////////////////////////////////////////
// CWzdToolBar

CWzdToolBar::CWzdToolBar()
{
    m_sSelection=_T("");
}

CWzdToolBar::~CWzdToolBar()
{
}


BEGIN_MESSAGE_MAP(CWzdToolBar, CToolBar)
    //{{AFX_MSG_MAP(CWzdToolBar)
    //}}AFX_MSG_MAP
    ON_CBN_CLOSEUP(IDC_WZD_COMBO, OnCloseupCombo)
    ON_CBN_DROPDOWN(IDC_WZD_COMBO, OnDropdownCombo)
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
// CWzdToolBar message handlers

BOOL CWzdToolBar::LoadToolBarEx(UINT id)
{
    // load toolbar info
    BOOL bRet;
    bRet=CToolBar::LoadToolBar(id);

    // find where our combo box will go
    int pos=CommandToIndex(IDC_WZD_COMBO);

    // covert button in toolbar into a spacer for our combo box
    SetButtonInfo(pos,IDC_WZD_COMBO,TBBS_SEPARATOR,COMBOLEN);

    // create combo box
    CRect rect;
    GetItemRect(pos,&rect);
    rect.bottom+=COMBODROP; //how far will combo drop down?
    m_ctrlWzdCombo.Create(WS_CHILD|WS_VISIBLE|CBS_DROPDOWNLIST, rect, this,
        IDC_WZD_COMBO);

    return bRet;
}


void CWzdToolBar::OnDropdownCombo()
{
```

```
    m_ctrlWzdCombo.ResetContent();

    m_ctrlWzdCombo.AddString("Monday");
    m_ctrlWzdCombo.AddString("Tuesday");
    m_ctrlWzdCombo.AddString("Wednesday");
      m_ctrlWzdCombo.SelectString(-1,m_sSelection);
}

void CWzdToolBar::OnCloseupCombo()
{
    int i;
    if ((i=m_ctrlWzdCombo.GetCurSel())!=CB_ERR)
    {
        m_ctrlWzdCombo.GetLBText(i, m_sSelection);
    }
    else
    {
        m_ctrlWzdCombo.AddString(m_sSelection);
        m_ctrlWzdCombo.SelectString(-1,m_sSelection);
    }
}
```

7.8 例29 修改应用程序的状态栏

目标

给应用程序的状态栏添加附加的指示器 (见图7-7)。

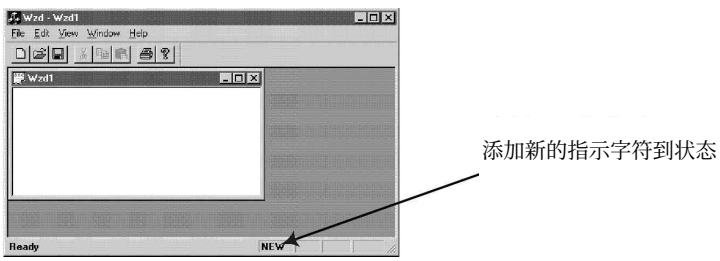

图7-7 用字符串表编辑器和文本编辑器添加一个窗格到状态栏中

注意 用本例的方法添加的任何状态栏指示器，在添加一个用户界面处理函数以更新它之前，都将保持空白。下一个例子将讲述怎样添加这样的一个处理函数。

策略

目前没有字符串表编辑器，若要添加指示器到一个状态栏中，首先必须用字符串表编辑器保存状态栏中有条件显示的文本，然后用文本编辑器，添加一行代码到 CMainFrame类中，告诉CStatusBar类创建另一个指示器 “ 窗格 ”。

步骤

1. 用字符串表编辑器添加一个状态字符串

1) 单击应用程序的ResourceView标签，并在StringTable文件夹中找到字符串表，双击字符串表的ID，以产生String Table Editor。

2) 要添加一个新的指示字符串到该表中，找到 ID_INDICATOR_REC串，并单击Studio的Insert/New String菜单命令。这样，既插入一个新字符串，同时还打开它的属性对话框 (见图7-8)。

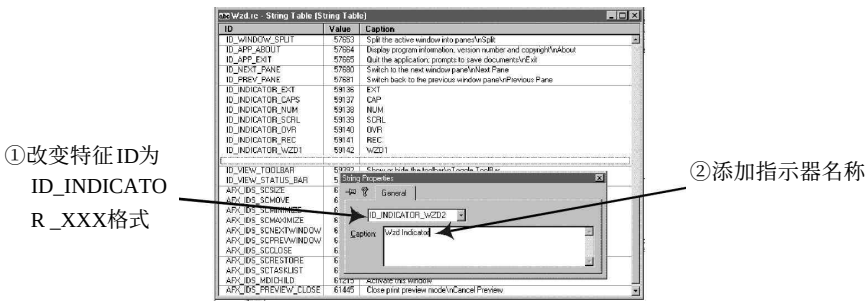

图7-8 用字符串编辑器添加一个状态字符串

3) 通常，还应赋予指示器ID一个类似于ID_INDICATOR_XXX的名称(这里的XXX描述指示器)，然后在Caption栏输入一个文本字符串。当状态栏窗格有效时，该字符串将在状态栏中显示，该字符串的长度决定在状态栏中显示指示器的窗格的长度。如果在该标题的前后插入空格，则窗格将显得比文本字符串大。单击属性对话框的关闭按钮，这一改变将被保存到表里。

2. 在MainFrame类中使用该新字符串

1) 把该指示器的ID添加到CMainFrame的indicators[ ]数组中，在MainFrm.cpp中找到该数组，并把新的ID加到数组中，如下所示。添加该ID的位置决定在状态栏上显示指示器的位置。

```
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_WZD1,      //<<<<<< new >>>>>>>
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

2) 继续下一个例子，以弄清楚怎样使指示器中的字符有条件地显示。

说明

字符串编辑器中的水平线表示字符串编号的间隔，例如： ID_INDICATOR_REC值为59141，而ID_VIEW_TOOLBAR则跳到59392，因此，在它们之间有一条水平线。为清楚起见，可以在这些水平线之一插入一个新字符串，即使已经选择 ID_INDICATOR_NUM作为想要添加新字符串的位置， String Table Editor也会在ID_INDICATOR_REC后默认设置一个位置。

CD说明

在CD上没有该例子的工程。

7.9 例30 更新状态栏窗格

目标

在应用程序的状态栏中启用并改变一个窗格 (见图7-9)。

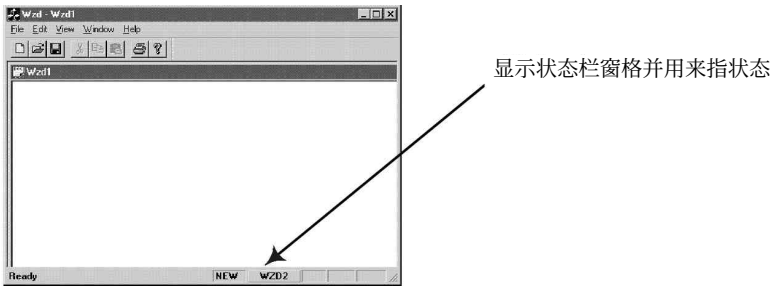

图7-9 通过系统盒的图标启动状态栏的窗格，以指示应用程序状态

策略

允许应用程序更新工具栏按钮状态的机理，同样也可以用来更新状态栏窗格的状态，但效果却不同。一个被无效化的状态栏窗格，不是显示灰色文本，而是根本不显示任何东西；被选中的窗格将从屏幕上“升起”。

步骤

1. 手工为每个状态栏窗格添加用户界面处理函数

1) 因为目前ClassWizard不能处理状态栏指示器，因此，必须手工为每个需要启用或禁用的状态栏窗格，添加一个用户界面处理函数。

```
BEGIN_MESSAGE_MAP(CWzdView, CView)
   //{{AFX_MSG_MAP(CWzdView)
   //}}AFX_MSG_MAP
   ON_UPDATE_COMMAND_UI(ID_INDICATOR_WZD1,
      OnUpdateIndicatorWzd1)
   ON_UPDATE_COMMAND_UI(ID_INDICATOR_WZD2,
      OnUpdateIndicatorWzd2)
END_MESSAGE_MAP()
```

确保把这些宏放在ClassWizard使用的({{ }})括弧之外，这里所用的ID是窗格的ID。

2) 在.h文件中定义指示器处理函数。

```
// Generated message map functions
protected:
   //{{AFX_MSG(CWzdView)
   //}}AFX_MSG
```

```
    afx_msg void OnUpdateIndicatorWzd1(CCmdUI *pCmdUI);
    afx_msg void OnUpdateIndicatorWzd2(CCmdUI *pCmdUI);
    DECLARE_MESSAGE_MAP()
```

2. 更新状态栏窗格

1) 打开一个状态指示器，可以用：

```
void CWzdView::OnUpdateIndicatorWzd1(CCmdUI *pCCmdUI)
{
    pCCmdUI->Enable(TRUE);
}
```

2) 打开一个状态指示器，并把它的名称改为NEW或任何其他名称，可以用：

```
void CWzdView::OnUpdateIndicatorWzd1(CCmdUI *pCCmdUI)
{
    pCCmdUI->Enable(TRUE);
    pCCmdUI->SetText("NEW");
}
```

3) 打开一个指示器并选取它(使它显得从屏幕中升起)，可以用：

```
void CWzdView::OnUpdateIndicatorWzd2(CCmdUI *pCCmdUI)
{
    pCCmdUI->Enable(TRUE);
    pCCmdUI->SetCheck();
}
```

说明

- 同工具栏按钮一样，状态栏窗格在应用程序空闲时更新。若要使它更早些时候更新，可以用UpdateWindow( )。

```
m_statusbar.UpdateWindow( )；
```

- 在这里调用的SetCheck( )函数，与更新菜单和工具栏界面时调用的 SetCheck( )函数一样。事实上，这些SetCheck( )的实例(和CCmdUI的其他成员函数)除了名字以外都是不同的。它们是CCmdUI基类的重载成员变量，还有其他四个从 CCmdUI基类派生的类，用来处理每种类型的条或菜单。在一个条或菜单将要被更新的时候，为每个窗格、按钮或控件创建合适的CCmdUI派生类，并用第3章中描述的OnCmdMsg( )机制，把它发送到处理函数。有关CCmdUI类和更新用户界面状态的详细内容，参见第 3章。
- 要添加其他控件到状态栏，参见下一个例子。

CD说明

在CD上执行该工程时将会看到在状态栏上有两个新的窗格，一个是按下的，另一个是升起的。

7.10 例31 为状态栏添加其他控件

目标

添加一个进度指示控件和一个按钮控件到状态栏 (见图7-10)。

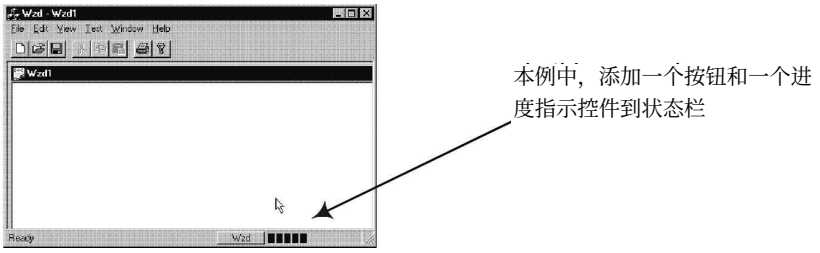

图7-10 添加控件到状态栏中

策略

用前面例子中所示的方法，添加两个新项目到状态栏中。然而这次添加的文本将是空白文本字符串，它只是作为将要动态创建的控件窗口的空间占有者；然后用 CStatusBar的GetItemRect( )成员函数，获取空间占有者的尺寸，并在它们上面手工创建控件。

步骤

1. 在状态栏中创建控件窗口占位符

1) 在字符串表中创建两个叫做 ID_INDICATOR_WZDPROGRESS和ID_INDICATOR_WZDBUTTON的新字符串，并定义它们为空白字符串。空格的数量将决定控件在状态栏中的大小。

2) 在需要它们在状态栏中显示的位置，添加这些串 ID到CMainFrame的指示器数组中。

```
// add new id's to indicators in Mainfrm.cpp
static UINT indicators[] =
{
   ID_SEPARATOR,              // status line indicator
   ID_INDICATOR_WZDBUTTON,    <<<<<<<<<<
   ID_INDICATOR_WZDPROGRESS,  <<<<<<<<<<<<<<<<
   ID_INDICATOR_CAPS,
   ID_INDICATOR_NUM,
   ID_INDICATOR_SCRL,
};
```

3) 如果现在编译该工程应该可以看到在状态栏上有两块空白区域。

2. 创建一个新的状态栏类

1) 使用ClassWizard创建从CStatusBar派生的CWzdStatusBar。

2) 把新的控件插入该类中。

```
CButton          m_WzdButton;
CProgressCtrl    m_WzdProgressCtrl;
```

3) 使用ClassWizard给该类添加一个WM_CREATE消息处理函数，在该函数中创建这些插入的控件。

```
int CWzdStatusBar::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CStatusBar::OnCreate(lpCreateStruct) == -1)
      return -1;
```

```
    CRect rect(0,0,0,0);
    m_WzdButton.Create( " Wzd " ,WS_CHILD,rect,this,
      IDC_WZD_BUTTON);
    CFont *pFont=CFont::FromHandle((HFONT)::
      GetStockObject(ANSI_VAR_FONT));
    m_WzdButton.SetFont(pFont);

    m_WzdProgressCtrl.Create(WS_CHILD|WS_VISIBLE,rect,this,
      IDC_WZD_PROGRESS);

    return 0;
}
```

可以看到，这些控件窗口创建时的初始大小为 0×0，因此，无论如何必须在创建后立即修改它们；还可以使用不同的按钮文本字体，因为默认的字体是黑体。

4) 用ClassWizard添加一个WM_SIZE消息处理函数到该类中，在那里为新的控件指定位置和大小。

```
void CWzdStatusBar::OnSize(UINT nType, int cx, int cy)
{
    CStatusBar::OnSize(nType, cx, cy);

    UINT inx;
    CRect rect;

    inx=CommandToIndex(ID_INDICATOR_WZDBUTTON);
    GetItemRect(inx,&rect);
    m_WzdButton.MoveWindow(rect);

    inx=CommandToIndex(ID_INDICATOR_WZDPROGRESS);
    GetItemRect(inx,&rect);
    m_WzdProgressCtrl.MoveWindow(rect);

}
```

注意，首先用CStatusBar::CommandToIndex( )确定新控件将要占据的空间的索引，然后用CStatusBar::GetItemRect( )获得那空间的尺寸，并用CWnd::MoveWindow( )把控件窗口移到那个空间中。

5) 要查看新状态栏类的详细的清单参见本节的 " 清单 —— 状态栏类 "。

3. 使用新的状态栏类

1) 在CMainFrame中，用新的状态栏类替换当前的状态栏类。

```
protected:  // control bar embedded members
    CWzdStatusBar       m_wndStatusBar;    <<<
    CToolBar            m_wndToolBar;
```

2) 创建应用程序时，它将包括这两个新控件。

3) 可以通过状态栏成员变量访问这些控件。

```
m_wndStatusBar.m_wndButton().ShowWindow(SW_SHOW);
m_wndStatusBar.m_wndProgressCtrl().SetPos(m_nInc);
```

4) 按钮控件把它的命令消息发送到CMainFrame，就像一个菜单命令，因此它可以在命令

消息链中被任何类处理。

5) 最后一个与按钮控件有关的问题是：当它被单击时，接收输入焦点并成为默认的按钮，这时一个较小尺寸的矩形绘制在按钮表面，并且按钮本身的边框变粗。要消除这些效果，可以添加下面的代码到处理该按钮的消息处理函数中。

```
void CMainFrame::OnWzdButton()
{
   // these commands keep the button from being highlighted
   SetFocus();
   m_wndStatusBar.GetButton().SetButtonStyle(BS_PUSHBUTTON);
   :  :  :
}
```

说明

用占位符而不用计算控件的尺寸，可以更好地控制按钮的定位，以及控件的处理。

CD说明

在CD上执行该工程时，单击 Text/Progress菜单命令，一个按钮和进度指示控件将出现在状态栏中。接着按Progress菜单项，将使进度指示控件变大；按下状态栏上的 Wzd按钮，将显示一个窗口以指示该按钮被按下。

清单 —— 状态栏类

```
#if !defined(AFX_WZDSTATUSBAR_H__DDA34AC3_D491_11D1_9B68_00AA003D8695__INCLUDED_)
#define AFX_WZDSTATUSBAR_H__DDA34AC3_D491_11D1_9B68_00AA003D8695__INCLUDED_

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

// WzdStatusBar.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdStatusBar window

class CWzdStatusBar : public CStatusBar
{
// Construction
public:
   CWzdStatusBar();

// Attributes
public:
   CButton &GetButton(){return m_WzdButton;};
   CProgressCtrl &GetProgressCtrl(){return m_WzdProgressCtrl;};
```

```
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdStatusBar)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdStatusBar();

    // Generated message map functions
protected:
    //{{AFX_MSG(CWzdStatusBar)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
private:
    CButton        m_WzdButton;
    CProgressCtrl m_WzdProgressCtrl;
};

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

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

#endif // !defined(
    AFX_WZDSTATUSBAR_H__DDA34AC3_D491_11D1_9B68_00AA003D8695__INCLUDED_)

// WzdStatusBar.cpp : implementation file
//

#include “ stdafx.h ”
#include “ wzd.h ”
#include “ WzdStatusBar.h ”

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

/////////////////////////////////////////////////////////////////////////////
```

```
// CWzdStatusBar

CWzdStatusBar::CWzdStatusBar()
{
}

CWzdStatusBar::~CWzdStatusBar()
{
}


BEGIN_MESSAGE_MAP(CWzdStatusBar, CStatusBar)
    //{{AFX_MSG_MAP(CWzdStatusBar)
    ON_WM_CREATE()
    ON_WM_SIZE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdStatusBar message handlers

int CWzdStatusBar::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CStatusBar::OnCreate(lpCreateStruct) == -1)
        return -1;

    CRect rect(0,0,0,0);
    m_WzdButton.Create( " Wzd " ,WS_CHILD,rect,this,IDC_WZD_BUTTON);
    CFont *pFont=CFont::FromHandle((HFONT)::GetStockObject(ANSI_VAR_FONT));
    m_WzdButton.SetFont(pFont);

    m_WzdProgressCtrl.Create(WS_CHILD|WS_VISIBLE,rect,this,IDC_WZD_PROGRESS);

    return 0;
}

void CWzdStatusBar::OnSize(UINT nType, int cx, int cy)
{
    CStatusBar::OnSize(nType, cx, cy);

    UINT inx;
    CRect rect;

    inx=CommandToIndex(ID_INDICATOR_WZDBUTTON);
    GetItemRect(inx,&rect);
    m_WzdButton.MoveWindow(rect);

    inx=CommandToIndex(ID_INDICATOR_WZDPROGRESS);
    GetItemRect(inx,&rect);
    m_WzdProgressCtrl.MoveWindow(rect);
```