第10章　帮　　助

联机帮助可以极大地减少用户花费在学习应用程序上的时间。用户可以直接查询每个控件本身，而不用在用户手册中寻找合适的命令。联机帮助还可以使应用程序立即对整个联机手册中的内容进行查询，而不像普通手册那样只能按照索引去寻找用户感兴趣的东西。同时也大大节省了打印手册的费用。

然而，只有当用户对应用程序所要做的有一个总体的了解时，联机帮助才会有用。若用户是一名完完全全的新手，它能提供的帮助好比是用一本英文字典去教一个人讲英文。当需要的只是一些最基本常识的时候，从使用手册或指导书中一些常用的部分即可获得帮助。

实例35　添加帮助菜单项，在本例中将向应用程序中的 Help菜单中添加Contents和Search菜单项。

实例36　添加上下文相关帮助，在本例中将向应用程序添加相关帮助。

实例37　添加气泡帮助，在本例中将向应用程序添加气泡帮助。

联机帮助的三种类型

联机帮助有以下三种类型：

1) 菜单帮助(Menu help)是用户通过检索Help菜单所能得到的帮助。虽然 AppWizard在这个菜单中仅仅加入了一个 About命令按钮，但用户自己还可以加入 Index和Content命令按钮。这样便允许用户无缝地通过 Windows的WinHelp应用程序打开帮助文件。如何在应用程序中添加菜单帮助请参考实例35。

2) 上下文相关帮助(Context Sentitive help)允许用户激活当前正在进行人机交互的任何菜单项或是对话框控件的帮助。实际上通过上下文相关帮助允许用户转到联机帮助文件中包含相应控件或菜单项帮助内容的页面。在上下文相关帮助的目录下有以下两种版本的帮助：

■ F1帮助允许用户按F1键以得到有关当前选中菜单项或者被激活对话框的帮助。

■ “what’s this(这是什么)”帮助允许用户通过单击控件或希望得到帮助的屏幕区域来获得帮助，这种方式使用户拥有更大的选择范围。当处于 What’this模式时，鼠标光标将变成一个问号箭头。

所有这些上下文相关帮助的例子均可在实例 36中找到。

3) 气泡(Bubble)帮助允许用户查询控件或视的某一区域，用户所做的仅仅是将鼠标光标移过要获得帮助的区域。一个小窗口将打开并描述该区域。气泡帮助不像上下文相关帮助所叙述的那么详细，但它的速度很快。气泡帮助的实例请参见实例 37。

10.1　实例35：添加帮助菜单项

1. 目标

将标准的Contents和Search菜单项加入到应用程序的Help菜单中，如图10-1所示。

2. 策略

事实上，在一些MFC类中已包含了许多用于提供菜单帮助的功能。因此，本例中要完成

的工作是有选择地“激活”这些功能。

3. 步骤

启用菜单帮助

利用菜单编辑器在帮助菜单中加入如下的命令：

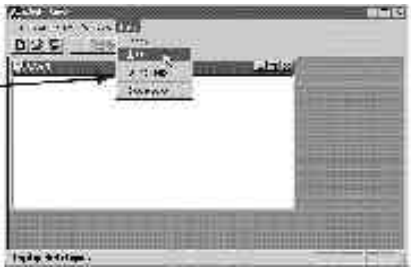

图10-1 菜单帮助

Name: &index
ID: ID_HELP_INDEX
Comment: “Display Help Index\nHelp Index”

Name: &Topics
ID: ID_HELP_FINDER
Comment: “Display Help topics\nHelp Topics”

Separator

Name: &Using Help
ID: ID_HELP_USING
Comment: “Display instructions about how to use help\nHelp”

Separator

注意 最后一个菜单命令“使用帮助”是微软的WinHelp程序提供的，它允许用户查找如何使用帮助。

使用文本编辑器，在ClassWizard的{{}}标识之外，加入下列消息宏到CMainFrame的消息映射中:

```
ON_COMMAND( ID_HELP_INDEX, CMDIFrameWnd::OnHelpIndex )
ON_COMMAND( ID_HELP_USING, CMDIFrameWnd::OnHelpUsing )
ON_COMMAND( ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder )
```

这样就可以了。现在MFC类库就可以激活Windows的WinHelp应用程序，并在用户单击菜单帮助的时候，将应用程序的帮助文件 (.hlp和.cnt)加入到其中。MFC的缺省情况是假定这些文件在应用程序的可执行文件的目录下。但如果想将这些文件放在其他位置，则要在应用程序类的InitInstance()函数中加入如下代码：

```
m_pszHelpFilePath = _tcsdup( _T(
   “d:\\somedir\\myhelp.hlp”          // the directory and name of
                                      //    your help file
   ) );
```

4. 注意

如果在第四步选择Context Help选项，AppWizard将会自动为这些菜单项添加一定的上下文相关帮助功能，同时还添加 .hlp和.cnt文件。然而在大多数公司，经常使用一些帮助文件的

编写系统诸如RoboHELP来创建.hlp和.cnt文件，这样在一定程度上会减少代码编写人员和产品经理的工作量。因而，最好亲自动手去添加这些菜单项，然后将这些固定的东西加入到工程中去。

5. 使用光盘时注意

当执行附带光盘中的工程时，将会发现帮助文件已经扩展，其中包含了三个新的帮助命令。

10.2 实例36：添加上下文相关帮助

1. 目标

在应用程序中添加上下文相关帮助(即点即显帮助)，如图10-2所示。

2. 策略

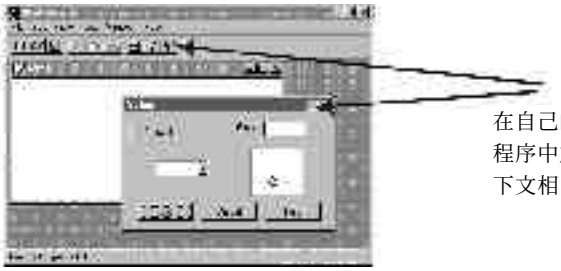

图10-2 上下文相关帮助

本例将再次激活MFC类库中的某些功能来为应用程序提供上下文相关帮助，同时还要在应用程序中加入 F1 Help和What's This Help。对于已被选的菜单项，用户可以通过按 F1键来获得F1 Help；对于被打开的对话框，用户可以通过按对话框中的 Help按钮或F1键来获得F1 Help。用户还可以通过按Shift+F1键来获得What ’s This Help，或者也可以通过单击What ’s this菜单项、工具条按钮，或是在对话框中标题栏中的问号按钮三者之一来获得 What ’s This Help。在用户单击一个主题之前， What ’s This Help将使鼠标光标显示为问号箭头。

3. 步骤

1) 在主应用程序中添加上下文相关帮助

使用文本编辑器在 CMainFrame类的消息映射中加入下列代码，代码应添加在ClassWizard{{}}之后：

```
    ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
```

如果帮助文件没有安装在应用程序的可执行文件的目录下，则在应用程序类的InitInstance()函数中加入如下代码：

```
m_pszHelpFilePath = _tcsdup( _T( ]
   “d:\\somedir\\myhelp.hlp”          // the directory and name of
                                      //     your help file
   ) );
```

使用Menu Editor(菜单编辑器)在帮助菜单中添加What ’s this命令按钮：

```
Name:              &What ’s this
ID:                ID_CONTEXT_HELP
Comment:   "Display help for clicked on buttons,menus and windows\nHelp"
```

也可以选择使用Toolbar Editor(工具条编辑器)在工具条中加入What's this按钮。设置其ID

号为ID_CONTEXT_HELP。使用AppWizard创建一个临时工程并且在第四步指定 Context help，可以为这个按钮获得一个位图 (一个带有箭头的问号)。然后从这个新工程的工具条中剪切下What ’s this位图并且将其粘贴到当前工程中。

利用Accelerator Editor(加速键编辑器)实现F1和Shift＋F1快捷键，并使用下列ID：

VK_F1	ID_CONTEXT_HELP	VIRTKEY,SHIFT
VK_F1	ID_HELP	VIRTKEY

使用String Table Editor(字符串编辑器)在字符串中加入两条空闲消息。其中一条提示用户可以按F1键以得到帮助，另一条则提示用户可以单击某一主题以得到帮助：

- 改变AFX_IDS_IDLEMESSAGE的消息为For Help,press F1。
- 添加AFX_IDS_HELPMODEMESSAGE的消息为: Select an object on which to get Help。

2) 在对话框中添加上下文相关帮助

使用Dialog Editor，打开模板自身的属性框，并且指定 Context help风格，这将在创建标题条时产生一个“ ？ ”按钮。对于每一个对话框模板重复该操作。

同样可以在每一个模板中添加 Help按钮。设置该按钮的ID号为ID_HELP。

对于对话框模板中的每一个控件，选择 Help ID选择项。这样就为该控件加入了一个帮助标识号，当调用WinHelp时通过帮助标识号来访问该控件。该标识号通过 Dialog Editor还将自动加入到一个resource.hm文件中。如果resource.hm文件不存在， Dialog Editor将自动创建一个。以后要编写帮助文件时将使用这个 .hm文件。

注意 帮助标识号是在控件的IDC号的基础上由对话框编辑器自动创建的。换句话说，如果控件的标识号是：

```
#define IDC_BUTTON1          1000
```

帮助标识号将是：

```
#define HIDC_BUTTON1         0x808203e8
```

这里标识号结尾处的“ 3e8 ”是IDC_BUTTON1中的1 000在十六进制时的等效值。这意味着如果要在另一个对话框中创建标识号同样为1 000的另一控件时，帮助系统将会激活原有的相同的帮助信息。因而，注意跟踪控件的标识号并保证标识号不重复是非常重要的，有时候应当手工分配控件标识号。这样，两个不同控件的帮助才不会互相冲突。

在对话框中支持上下文相关帮助需要在此对话框中处理两个消息： WM_HELPINFO和WM_HELPHITTEST。在What's this模式中，当用户单击对话框中的某一项时，系统发出第一条消息。第二条和第一条大致相同，不同之处仅在对话框当前没有激活，处于无模式对话框或对话条两种状态之一。在这两种情况下，需要自行确定用户单击的内容是什么，并且如果对话框具有帮助标识号时或者返回该标识号或者自行调用帮助系统。

在每个对话框类中加入 WM_HELPINFO和WM_HELPHITTEST消息处理函数(WM_HELPHITTEST必须手工添加, WM_HELPINFO可用ClassWizard添加):

```
BEGIN_MESSAGE_MAP( CWzdDialog, CDialog )
   // {{AFX_MSG_MAP( CWzdDialog )
   ON_WM_HELPINFO()
   // }}AFX_MSG_MAP
   ON_MESSAGE( WM_HELPHITTEST,OnHelpHitTest )
```

```
END_MESSAGE_MAP()
```

对于OnHelpINfo()处理函数，确定哪一种控件被选择并且查询其标识号。然后使用CWinHelp::WinHelp()建立合适的上下文相关帮助：

```
BOOL CWzdDialog::OnHelpInfo( HELPINFO* pHelpInfo )
{
   if ( pHelpInfo -> iContextType == HELPINFO_WINDOW )
   {
      DWORD helpId = -1;
      CWnd* pWnd = GetDlgItem( pHelpInfo -> iCtrlId );
      if ( pWnd )
      {
            helpId = pWnd -> GetWindowContextHelpId();
      }
      AfxGetApp() -> WinHelp( helpId, HELP_CONTEXTPOPUP );
   }

   // return CDialog::OnHelpInfo( pHelpInfo );
   return TRUE;
}
```

使用OnHelpHitTest()时，在可以查询其帮助标识号之前，必须确定用户单击了哪一个控件。不过这时可以返回该标识号到MFC类，由它激活帮助系统：

```
LRESULT CWzdDialog::OnHelpHitTest( WPARAM wParam, LPARAM lParam )
{
   DWORD helpId = -1;
   // find the window that was clicked
   CWnd* pWnd =  ChildWindowFromPoint( CPoint( LOWORD( lParam ),
      HIWORD( lParam ) ), CWP_SKIPINVISIBLE|CWP_SKIPTRANSPARENT );
   if ( pWnd )
   {
      helpId = pWnd -> GetWindowContextHelpId();
   }
   return helpId;
}
```

4. 注意

AppWizar也将自动添加这些功能中的一部分。请参考前面实例中的“注意”部分。

5. 使用光盘时注意

当执行附带光盘中的工程时，将会发现帮助菜单已经扩展，包含 What’s this项，工具条中也有一个附加的What’s this按钮。单击Test/Wzd菜单命令打开对话框，其中包括一个 Help按钮，标题栏中有一个“？”按钮。

10.3 实例37：添加气泡帮助

1. 目标

在应用程序中加入气泡帮助，如图 10-3所示。

2. 策略

本例使用CToolTipCtrl类来提供气泡帮助。这个类所做的工作仅仅是当用户将鼠标移过某

一控件或是视中的某一区域时，能打开一个气泡似的小窗口。在这个窗口中将显示出一段短小的文本信息，该文本信息在 CToolTipCtrl中定义。使用CToolTipCtrl的方法是将鼠标消息截获给它，由于它所要监控的 WM_MOUSEMOVE鼠标消息实际上是进入鼠标所在窗口的窗口过程(Window process)。例如，当鼠标移过一个按钮控件时，该按钮控件将获取所有这些鼠标消息。所幸的是在CToolTipCtrl封装的工具提示(tool tip)控件中提供了一个子类化功能，它允许截取这些消息。但在另一方面，因为某些原因， CToolTipCtrl封装的工具提示控件的功能实现起来并不是十分简单。因而将在 CToolTipCtrl派生类中添加自己的函数以直接访问工具提示控件。

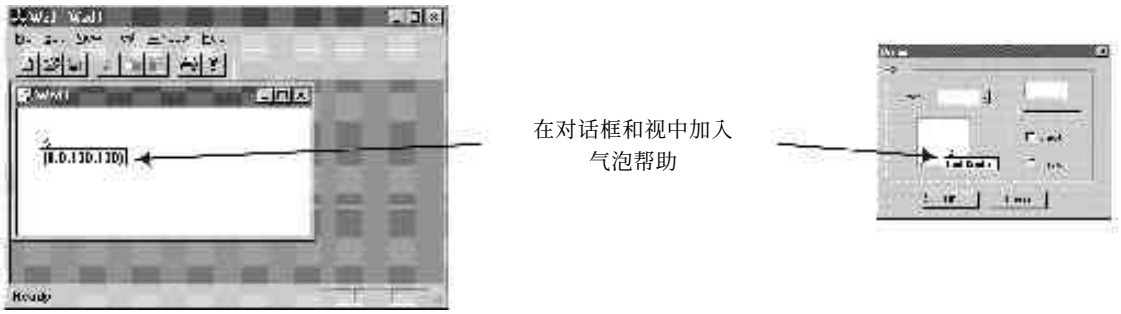

图10-3 气泡帮助

3. 步骤

1) 派生一个新的CToolTipCtrl类

使用ClassWizard创建一个CToolTipCtrl子类。

使用文本编辑器添加一个新函数 AddTool()。这个函数允许在对话框类中嵌入一个单个的工具提示控件，使用它可支持对话框中的所有控件。首先可以创建一个空的 TOOLINFO结构，以发送到工具提示控件:

```
BOOL CWzdToolTipCtrl::AddTool( UINT nID, LPCTSTR lpszText  )
{
   TOOLINFO ti;
   memset( &ti, 0, sizeof( TOOLINFO ) );
   ti.cbSize = sizeof( TOOLINFO );
```

接下来保存用户传送的文本信息:

```
   ti.lpszText = ( LPSTR )lpszText;
```

接下来要求工具提示控件对来自指定控件标识号的信息进行分类:

```
   ti.uFlags = TTF_IDISHWND|TTF_SUBCLASS;
   ti.uId = ( UINT )GetParent() -> GetDlgItem( nID ) -> m_hWnd;
```

接下来确定这个工具提示控件的属主:

```
   ti.hwnd = GetOwner() -> GetSafeHwnd();
```

最后将下面的消息发送给控件:

```
   return ( BOOL )SendMessage( TTM_ADDTOOL, 0, ( LPARAM )&ti );
}
```

2) 在对话框中实现新的CToolTipCtrl类

在对话框中嵌入下面一个新类：

```
CWzdToolTipCtrl m_tooltips;
```

如果希望在某些静态控件上面显示气泡帮助，则需要使用对话框编辑器将它们的风格修改为Parent notify。在此不需要对除了静态控件以外的任何控件进行任何修改，因为所有这些控件的风格将自动通知它们的父窗口。

使用ClassWizard添加一个WM_INITDIALOG消息处理函数到这个对话框类中，并且创建气泡帮助窗口：

```
m_tooltips.Create( this );
```

注意 尽管创建了一个窗口，但该窗口在工具提示控件为某一控件显示气泡帮助前一直是不可见的。同时注意在这个对话框中，该窗口将为所有这些控件显示气泡帮助。

在WM_INITDIALOG处理函数中，使用前面添加的 AddTool()函数，为需要显示气泡帮助的所有控件指定各自的帮助信息：

```
m_tooltips.AddTool(
   IDC_RADIO1,          // control id
   “Radio Control”      // bubble help message
   );
```

最后，激活工具提示：

```
m_tooltips.Activate( TRUE );
```

如果要在其他地方修改控件的帮助文本信息，使用以下代码：

```
m_tooltips.UpdateTipText(
   “Checked Box”,              // new text
   GetDlgItem(
      IDC_CHECK1               // the control’s id
      )
   );
```

以上便为对话框中的控件加入了气泡帮助。当在一个父窗口中有许多子窗口而又不想为每一个子窗口实现CToolTipCtrl控件时，同样可以使用这种解决办法。接下来将创建一个新的CToolTipCtrl派生类，它仅在其他窗口的特定区域特别是视中显示气泡帮助。

3) 添加一个新的CToolTipCtrl派生类

使用文本编辑器，在新工具提示类中加入一个新的函数 AddArea()。同样可以先创建一个空的TOOLINFO结构，以发送到工具提示控件:

```
BOOL CWzdToolTipCtrl::AddArea( UINT nID, LPRECT lpRect,
   LPCTSTR lpszText )
{
   TOOLINFO ti;
   memset( &ti, 0, sizeof( TOOLINFO ) );
   ti.cbSize = sizeof( TOOLINFO );
```

指定要显示的文本：

```
   ti.lpszText = ( LPSTR )lpszText;
```

告诉工具提示控件应监视的窗口 (例如视)和区域：

```
   ti.hwnd = GetOwner() -> GetSafeHwnd();
```

```
    memcpy( &ti.rect, lpRect, sizeof( RECT ) );
```

同时告诉工具提示控件为这个窗口分类，这样控件可以截获自己鼠标消息：

```
    ti.uFlags = TTF_SUBCLASS;
```

接下来指定一个自行定义的标识号，它将允许在以后自行修改该工具提示，该标识号可以是任何未分配的整型数：

```
    ti.uId = nID;
```

现在将该定义发送给工具条，然后返回：

```
    return ( BOOL )SendMessage( TTM_ADDTOOL, 0, ( LPARAM )&ti );
}
```

4) 在视中实现一个新的CToolTipCtrl类

在视类中嵌入新的工具提示类：

```
CWzdToolTipCtrl m_tooltips;
```

在视的WM_CREATE消息处理函数中创建它：

```
m_tooltips.Create( this );
```

在该消息处理函数或其他位置，定义希望显示气泡帮助信息的视中的区域：

```
m_tooltips.AddArea(
    1,                  // user defined id
    &rect1,             // a CRect value defining the area in view
                        //    coordinates
    “message”,          // the bubble help message
    );
```

在退出该消息处理函数之前，用下面的代码激活工具提示控件：

```
m_tooltips.Activate( TRUE );
```

为更新帮助文本信息，使用以下代码：

```
m_tooltips.UpdateTipText(
    “new text”,     // the new text message
    this,           // pointer to view class
    1               // the id defined above
    );
```

为改变这个工具提示所指向的区域，可以使用以下代码：

```
m_tooltips.SetToolRect(
    this,           // pointer to view class
    1,              // the id defined above
    &rect           // a new CRect value defining
                    //    the area in view coordinates
    );
```

参考本实例结尾的程序清单 —— 工具提示类来查看工具提示类的完整代码列表。

4. 注意

■ 另一个解决方案是为每一个对话框控件分别加入各自的工具提示控件。然而，这将涉及到为每一个控件创建自己的派生类，只有这样才可以在其中嵌入工具提示类。

■ 工具提示类显示单行的文本。对于特别长的信息或是某些特殊类型的信息，也许想为

文本进行多行显示，但仅仅在文本中加入一个 \n换行符是不行的。必须在工具提示控件的派生类中加入 WM_PAINT和WM_NCPAINT消息处理函数并且自行绘制窗口。使用函数GetWindowText()可获取要显示的帮助文本信息。

5. 使用光盘时注意

■ 当执行附带光盘中的工程时，将会发现鼠标在视的左上角移动时，气泡帮助窗口将被打开。然后单击 Test/Wzd菜单命令打开对话框，这允许光标移过该对话框的任何控件时，将显示气泡帮助信息。

■ 该工程文件中，同时提供了 WzdTT2.cpp和.h文件，它们是创建用户自己的工具提示帮助的基础。

6. 程序清单 —— 工具提示类

```
#if !defined( AFX_WZDTLTIP_H__1E222DA3_97EF_11D2_A18D_C32FFDBA4686__INCLUDED_ )
#define AFX_WZDTLTIP_H__1E222DA3_97EF_11D2_A18D_C32FFDBA4686__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif    // _MSC_VER > 1000
// WzdTlTip.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdToolTipCtrl window

class CWzdToolTipCtrl : public CToolTipCtrl
{
// Construction
public:
    CWzdToolTipCtrl();

// Attributes
public:

// Operations
public:
    BOOL AddTool( UINT nID, LPCTSTR lpszText = LPSTR_TEXTCALLBACK  );
    BOOL AddTool( UINT nID, UINT nIDText  );
    BOOL AddArea( UINT nID, LPRECT lpRect,
        LPCTSTR lpszText = LPSTR_TEXTCALLBACK  );
    BOOL AddArea( UINT nID, LPRECT lpRect, UINT nIDText );

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

// Implementation
public:
    virtual ~CWzdToolTipCtrl();

    // Generated message map functions
protected:
```

```
    // {{AFX_MSG( CWzdToolTipCtrl )
    // NOTE - the ClassWizard will add and remove member functions here.
    // }}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

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

// {{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
//     the previous line.

#endif
    // !defined( AFX_WZDTLTIP_H__1E222DA3_97EF_11D2_A18D_C32FFDBA4686__INCLUDED_ )
// WzdTlTip.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdTlTip.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdToolTipCtrl

CWzdToolTipCtrl::CWzdToolTipCtrl()
{
}

CWzdToolTipCtrl::~CWzdToolTipCtrl()
{
}


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

/////////////////////////////////////////////////////////////////////////////
// CWzdToolTipCtrl message handlers

BOOL CWzdToolTipCtrl::AddTool( UINT nID, LPCTSTR lpszText )
{
    TOOLINFO ti;
```

```
    memset( &ti, 0, sizeof( TOOLINFO ) );
    ti.cbSize = sizeof( TOOLINFO );
    ti.hwnd = GetOwner() -> GetSafeHwnd();
    ti.uFlags = TTF_IDISHWND|TTF_SUBCLASS;
    ti.uId = ( UINT )GetParent() -> GetDlgItem( nID ) -> m_hWnd;
    ti.lpszText = ( LPSTR )lpszText;
    return ( BOOL )SendMessage( TTM_ADDTOOL, 0, ( LPARAM )&ti );
}

BOOL CWzdToolTipCtrl::AddTool( UINT nID, UINT nIDText )
{
    CString str;
    VERIFY( str.LoadString( nIDText ) );
    return AddTool( nID,str );
}

BOOL CWzdToolTipCtrl::AddArea( UINT nID, LPRECT lpRect, LPCTSTR lpszText )
{
    TOOLINFO ti;
    memset( &ti, 0, sizeof( TOOLINFO ) );
    ti.cbSize = sizeof( TOOLINFO );
    ti.hwnd = GetOwner() -> GetSafeHwnd();
    ti.uFlags = TTF_SUBCLASS;
    ti.uId = nID;
    memcpy( &ti.rect, lpRect, sizeof( RECT ) );
    ti.lpszText = ( LPSTR )lpszText;
    return ( BOOL )SendMessage( TTM_ADDTOOL, 0, ( LPARAM )&ti );
}

BOOL CWzdToolTipCtrl::AddArea( UINT nID, LPRECT lpRect, UINT nIDText )
{
    CString str;
    VERIFY( str.LoadString( nIDText ) );
    return AddArea( nID,lpRect,str );
}
```