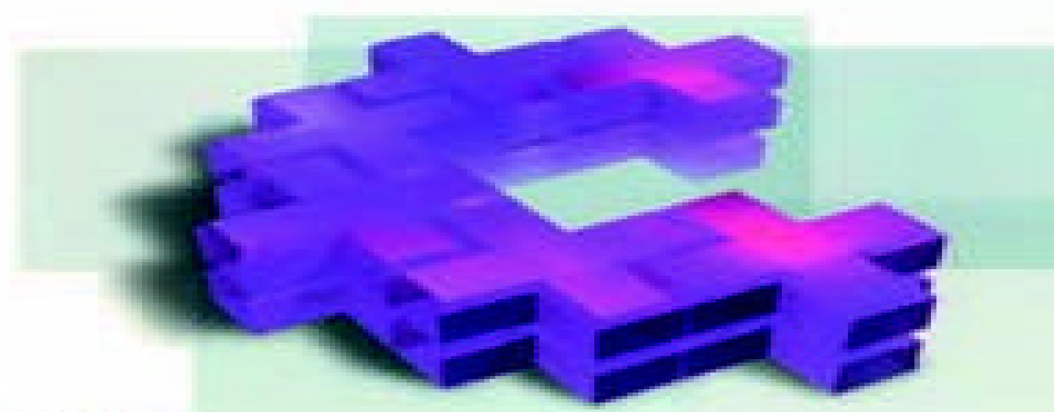
2



Microsoft Visual C++ 6.0

MFC Library Reference Part 2

类库参考手册（下）

北京希望电脑公司

Microsoft Press

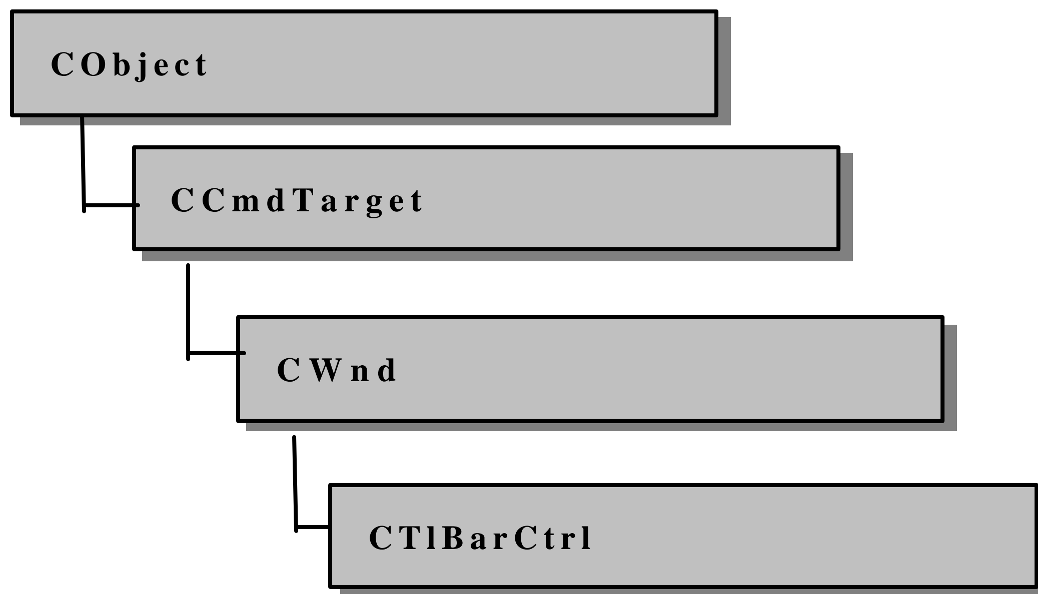
返 回

目 录

CToolBarCtrl
CToolTipCtrl
CTreeCtrl
CTreeView
CTypedPtrArray
CTypedPtrList
CTypedPtrMap
CUIntArray
CUserException
CView
CWaitCursor
CWinApp
CWindowDC
CWinThread
CWnd
CwordArray
MFC 中的宏和全局函数、变量

宏、全局函数和全局变量
结构、风格、回调函数和消息映射

CToolBarCtrl



CToolBarCtrl 类提供了 Windows 工具条通用控件的性能。这个控件（也就是 CToolBarCtrl 类）只对运行在 Windows 95 和 Windows NT 3.51 或更高版本下的程序来说才是可用的。

一个 Windows 工具条通用控件是一个矩形子窗口，它包含一个或多个按钮。这些按钮可以显示一个位图图像，一个字符串，或两者都有。当用户选择一个按

钮时，它向该工具条的属主窗口发送一条命令消息。通常，工具条中的按钮对应于应用程序的菜单中的项；这些按钮为用户访问一个应用程序的命令提供了更为直接的方法。

CToolBarCtrl 对象包含几个重要的内部数据结构：一个按钮图像列表或一个图像列表，一个按钮标签字符串列表和一个 TBBUTTON 结构的列表，该结构使一个图像和/或字符串与按钮的位置，风格，主题和命令 ID 相关联。这些数据结构的每一个都由一个从零开始的索引来引用。在你可以使用一个 CToolBarCtrl 对象之前，你必须设置这些数据结构。字符串列表只能被用作按钮标签；不能从按钮中检取字符串。

要使用一个 CToolBarCtrl 对象，通常你应该遵循下面的这些步骤：

1. 构造该 CToolBarCtrl 对象。
2. 调用 Create 来创建 Windows 工具条通用控件并将它与该 CToolBarCtrl 对象连接。通过使用风格来指定工具条的风格，如对一个透明的工具条使用 TBSTYLE_ TRANSPARENT，或对一个支持风格按钮的工具条使用 TBSTYLE_DROPDOWN。
3. 指明你希望按钮在工具条上如何显示：

- 给按钮使用位图图像，通过调用 AddBitmap 将按钮位图添加到工具条中。
- 给按钮使用来自一个图像列表的图像，通过调用 SetImageList，SetHotImageList，或 SetDisabledImageList 来指定图像列表。
- 给按钮使用字符串标签，通过调用 AddString 和/或 AddStrings 来将字符串添加到工具条中。

4. 通过调用 AddButtons 将按钮结构添加到工具条中。
5. 如果你希望在一个不是 CFrameWnd 的属主窗口中的工具条具有工具提示，则你必须在工具条的属主窗口中处理 TTN_NEEDTEXT 消息，就像在 CToolBarCtrl：处理工具提示通知中描述的一样。如果工具条的父窗口是由 CFrameWnd 派生而来的，则你不用作任何额外的努力就可以显示工具提示，因为 CFrameWnd 提供了一个缺省的处理函数。
6. 如果你希望能够让用户定制工具条，则在属主窗口中处理定制通知消息，就像在 CToolBarCtrl：处理定制通知中描述的一样。

你可以使用 SaveState 来将一个工具条控件的当前状态保存在注册表中，用 RestoreState 来根据注册表中先前保存的信息恢复工具条的状态。除了在应用程序的使用之间保存工具条的状态，通常在用户开始定制该工具条之前应用程序会保存工具条的状态，以防用户后来想将工具条恢复到它的最初的状态。

为 Internet Explorer 4.0 或更新版提供的支持

要支持在 Internet Explorer 4.0 或更新版之后引入的性能，MFC 提供了图像列表支持，为工具条提供了透明和平坦风格。

一个透明的工具条允许在工具条下的客户被透过工具条显示出来。要创建一个透明的工具条，要同时使用 TBSTYLE_FLAT 和 TBSTYLE_TRANSPARENT 风格。透明的工具条具有热点跟踪的特色；就是说，当鼠标指针移动到工具条的一个热点按钮上时，按钮的外观改变。只用 TBSTYLE_FLAT 风格创建的工具条将包含不透明的按钮。

图像列表支持使控件的缺省行为具有更大的灵活性，并支持热点图像和无效的图像。对透明的工具条使用 GetImageList，GetHotImageList 和 GetDisabledImageList 可以根据它的状态来操纵图像。

有关使用 CToolBarCtrl 的更多信息，参见“Visual C++程序员指南”中的“控件主题”和“使用 CToolBarCtrl”。

#include <afxcmn.h>

请参阅　CToolBar

CToolBarCtrl 类成员

Construction	
CToolBarCtrl	构造一个 CToolBarCtrl 对象
Create	创建一个工具条控件并将它与一个 CToolBarCtrl 对象连接

Attributes	
IsButtonEnabled	指示一个工具条控件中的指定按钮是否是有效的
IsButtonChecked	指示一个工具条控件中的指定按钮是否被核选了
IsButtonPressed	指示一个工具条控件中的指定按钮是否被压住
IsButtonHidden	指示一个工具条控件中的指定按钮是否被隐藏

续表

IsButtonIndeterminate	指示一个工具条控件中的指定按钮的状态是否是不确定的（灰的）
SetState	设置一个工具条控件中的指定按钮的状态
GetState	获取关于一个工具条控件中的指定按钮的状态的信息，比如它是有效的、被压住的，或是被核选的
GetButton	获取一个工具条控件中的指定按钮的信息
GetButtonCount	获取当前在该工具条控件中的按钮的数目
GetItemRect	获取一个工具条控制中的按钮的边界矩形
GetRect	获取一个特定工具条按钮的边界矩形
SetButtonStructSize	指定 TBBUTTON 结构的大小
GetButtonSize	获取的像素表示的工具条按钮当前宽度和高度
SetButtonSize	设置要被添加到一个工具条控件中去的按钮的尺寸
SetBitmapSize	设置要被添加到一个工具条控件中去的位图图像的尺寸
GetToolTips	获取与此工具条控件相关联的工具提示控件（如果有的话）的句柄
SetToolTips	将一个工具提示控件与该工具条控件关联
SetOwner	设置接收来自工具条控件的通知消息的窗口
SetRows	设置显示在工具条中的按钮的行数

续表

GetRows	获取当前显示在工具条中的按钮的行数
SetCmdID	设置当指定按钮被按下时，要发送到属主窗口的命令标识符
GetBitmapFlags	获取与工具条位图相关联的标志
GetDisabledImageList	获取被一个工具条控件用来显示无效按钮的图像列表
GetHotImageList	获取被一个工具条控件用来显示“热点”按钮的图像列表。当鼠标指针在一个热点控件上时，该控件被加亮显示
GetImageList	获取被一个工具条控件用来显示缺省状态按钮的图像列表
GetStyle	获取一个工具条控件当前使用的风格
GetMaxTextRows	获取显示在一个工具条按钮上的文本行的最大行数
IsButtonHighlighted	检验工具条控件的加亮状态
SetButtonWidth	设置工具条控件中的按钮宽度的最大和最小值
SetDisabledImageList	设置将要被工具条控件用来显示无效按钮的图像列表
SetHotImageLise	设置将要被工具条控件用来显示“热点”按钮的图像列表

续表

SetImageList	设置将要被工具条控件用来显示缺省状态按钮的图像列表
GetDropTarget	获取一个工具条控件的 IDropTarget 接口
SetIndent	设置一个工具条控件中的第一个按钮的缩排
SetMaxTextRows	设置一个工具条按钮所显示的最大文本行数
SetStyle	设置一个工具条控件的风格
GetAnchorHighlight	获取一个工具条的固定的加亮设置
SetAnchorHighlight	设置一个工具条的固定的加亮设置
GetHotItem	获取一个工具条中的热点项的索引
SetHotItem	设置一个工具条中的热点项的索引
GetInsertMark	获取工具条的当前插入标记
SetInsertMark	设置工具条的当前插入标记
GetMaxSize	获取工具条中的所有可视按钮和分隔线的总的尺寸
InsertMarkHitTest	获取一个工具条中的某个点的插入标记信息
GetExtendedStyle	获取一个工具条控件的扩展风格

续表

SetExtendedStyle	设置一个工具条控件的扩展风格
GetInsertMarkColor	获取用来绘制工具条的插入标记的颜色
SetInsertMarkColor	设置用来绘制工具条的插入标记的颜色
MapAccelerator	将一个加速键映射到一个工具条按钮
MoveButton	将一个按钮从一个索引移动到另一个索引
HitTest	确定一个点位于一个工具条控件的什么地方
Operations	
EnableButton	使一个工具条控件中的按钮有效或无效
CheckButton	核选或清除一个工具条控件中的指定按钮
PressButton	按下或释放一个工具条控件中的指定按钮
GetButtonInfo	获取工具条中的一个按钮的信息
SetButtonInfo	设置工具条中的一个按钮的信息
SetDrawTextFlags	设置 Win 32 中 DrawText 功能的标志，该功能通常根据标志设置的指定矩形、格式设置文本
HideButton	隐藏或显示一个工具条控件中的指定按钮
Indeterminate	设置或清除一个工具条控件中的指定按钮的不确定（灰的）状态

续表

AddBitmap	将一个或更多个位图按钮图像添加到一个工具条控件可用的按钮图像列表中
AddButtons	将一个或多个按钮添加到一个工具条按钮中
InsertButton	在一个工具条控件中插入一个按钮
DeleteButton	从该工具条控件中删除一个按钮
CommandToIndex	获取与指定的命令标识符相关联的按钮的从零开始的索引
RestoreState	恢复该工具条控件的状态
MarkButton	设置在一个工具条控件中给定的高亮状态
LoadImages	将位图定位到工具条控件的图像列表中
SaveState	保存该工具条控件的状态
Customize	显示 CustomizeToolbar 对话框
AddString	将作为一个资源 ID 传递的新字符串添加到工具条的内部字符串列表中
AddStrings	将一个或多个新字符串添加到工具条的内部字符串列表中，是通过传递指向这些用.空字符分隔的字符串的缓冲区的指针来传递这些字符串的
AutoSize	调整一个工具条控件的尺寸

CToolBarCtrl：处理工具提示通知

当你指定工具条具有 TBSTYLE_TOOLTIPS 风格时，该工具条创建并管理一个工具提示控件。一个工具提示是一个小的弹出窗口，该窗口包含了一行用于描述一个工具条按钮的文本。通常该工具提示是被隐藏的，只有当用户将光标放在一个工具条按钮上并停留大概半秒时间时它才显示出来。工具提示显示在光标的附近。

在工具提示被显示之前，TTN_NEEDTEXT 通知消息被发送给该工具条的属主窗口，以获取对应于该按钮的描述文本。如果该工具条的属主窗口是一个 CFrameWnd 窗口，则不需要任何额外的工作就会显示工具提示，因为 CFrameWnd 有一个缺省的 TTN_NEEDTEXT 通知处理函数。如果该工具条的属主窗口不是由 CFrameWnd 派生而来的，比如是一个对话框或格式视，则你必须在你的属主窗口的消息映射中添加一项，并在消息映射中提供一个通知处理函数。下面就是要被添加到你的属主窗口的消息映射：

```
ON_NOTIFY_EX( TTN_NEEDTEXT, 0, memberFxn )
```

memberFxn

当这个按钮的文本需要显示时将被调用的成员函数。

注意，一个工具提示的 id 总是 0。

除了 TTN_NEEDTEXT 通知，一个工具提示控件可以向一个工具条控件发送下列通知：

通知消息	含义
TTN_NEEDTEXTA	需要ASCII文本的工具提示控件（只在Win95下）
TTN_NEEDTEXTW	需要UNICODE文本的工具提示控件（只在Windows NT下）
TBN_HOTITEMCHANGE	表示热点（被加亮的）项已经改变了
NM_RCLICK	表示用户用鼠标右键单击一个按钮
TBN_DRAGOUT	表示用户点击按钮并拖动指针离开了按钮。它支持应用程序实现对一个工具条按钮的拖放。但接收到这个通知时，应用程序将开始拖放操作
TBN_DROPDOWN	表示用户已经点击了一个使用TBSTYLE_DROPDOWN风格的按钮
TBN_GETOBJECT	表示用户已经将指针移动到了一个使用TBSTYLE_DROPDOWN风格的按钮上

处理函数的例子和有关使能工具提示的更多信息，参见“Visual C++程序员指南”中的“工具提示”。

CToolBarCtrl：处理定制通知

一个Windows工具条通用控件具有内在的定制特征，包括一个系统定义的定制

对话框，用来让用户插入、删除，或重新安排工具条按钮。应用程序决定定制是否有效，并控制用户对该工具条定制的限度。

你可以通过给予工具条 CCS_ADJUSTABLE 风格来使这些定制特征对于用户来说是可用的。此定制特征允许用户将一个按钮拖动到一个新的位置，或通过拖动按钮离开工具条来删除这个按钮。另外，用户可以双击工具条来显示 Customize Toolbar 对话框，让用户添加、删除，或重新安排工具条按钮。应用程序可以通过使用 Custoimize 成员函数来显示对话框。

在定制过程中的每一步，工具条控件都向父窗口发送通知消息。如果用户按住 SHIFT 键并开始拖动一个按钮，则工具条自动处理这个拖动操作。工具条发送 TBN_QUERYDELETE 通知消息给父窗口，以确定是否应该删除该按钮。如果父窗口返回 FALSE，则这个拖动操作结束。否则，工具条捕捉鼠标输入并等待用户释放鼠标按钮。

当用户释放鼠标按钮时，工具条控件确定鼠标光标所在的位置。如果该光标的位置是在工具条之外，则按钮被删除。如果光标位于另一个工具条按钮上，则工具条向其父窗口发送 TBN_QUERYINSERT 通知消息，以确定是否要将一个按钮插入在给定按钮的左边。如果父窗口返回 TRUE，则该按钮被插入；否则，不插入。工具条发送 TBN_TOOLBARCHANGE 通知来表示拖动操作结束。
如果用户在没有按下 SHIFT 键的情况下开始一次拖动操作，则工具条控件发送 TBN_BEGINDRAG 通知消息给属主窗口。一个实现了自己的按钮拖动代码的应用程序可以使用这个消息作为开始一次拖动操作的信号。工具条发送 TBN_ENDDRAG 通知消息来表示拖动操作结束。

当用户通过使用 Customize Toolbar 对话框来定制一个工具条时，该工具条控件发送通知消息。在用户双击工具条之后，但在对话框被创建之前，该工具条发送 TBN_BEGINADJUST 通知消息。然后，工具条开始发送一系列 TBN_QUERYINSERT 消息，以确定该工具条是否允许插入按钮。当父窗口返回 TRUE 时，该工具条停止发送 TBN_QUERYINSERT 通知消息。如果父窗口没有对任何按钮返回 TRUE，则工具条销毁该对话框。

接着,工具条控件通过为工具条中的每一个按钮发送一个 TBN_QUERYDELETE 通知消息来确定是否可以从工具条中删除某个按钮。父窗口返回 TRUE 则表示可以删除该按钮；否则，父窗口返回 FALSE。工具条将所有的按钮都添加到对话框中，但是将那些不能删除的按钮变灰。

不管什么时候，当工具条控件需要有关 Customize Toolbar 对话框中的某个控件的信息时，它就发送 TBN_GETBUTTONINFO 通知信息，并指定它想获取其信息的按钮的索引，以及一个 TBNOTIFY 结构的地址。父窗口必须用相关的信息填充该结构。

Customize Toolbar 对话框还包括一个 Help 按钮和一个 Reset 按钮。当用户选择 Help 按钮时，工具条控件发送 TBN_CUSTHELP 通知消息。父窗口应该通过显示帮助信息来作出响应。当用户选择 Reset 按钮时，对话框发送 TBN_RESET 消息。这个消息表示工具条要再次初始化这个对话框。

这些消息都是 WM_NOTIFY 消息，通过向你的属主窗口的消息映射中添加下列形式的消息映射项，你可以在你的属主窗口中处理它们。

ON_NOTIFY（*wNotifyCode, idControl,memberFxn*）

wNotifyCode

通知消息标识符代码，比如 TBN_BEGINADJUST。

idControl

发送通知的控件的标识符。

memberFxn

当这个通知被接收时要被调用的成员函数。

你的成员函数应该用下面的原形来声明：

afx_msg void *memberFxn*(NMHDR **pNotifyStruct*, LRESULT **result*);

如果通知消息的处理函数返回一个值，则处理函数应该将这个值存放在由 result 指向的 LRESULT 中。

对每一个消息，pNotifyStruct 指向一个 NMHDR 结构或一个 TBNOTIFY 结构。这些结构被描述如下：

NMHDR 结构包含了下列成员：

```
typedef struct tagNMHDR {
   HWND hwndFrom;    // 发送消息的控件的句柄
   UINT idFrom;      // 发送消息的控件的标识符
   UINT code;        // 通知代码，参见下面
} NMHDR;
```

hwndFrom

正发送通知的控件的窗口句柄。要将这个句柄转换为一个 CWnd 指针，可以使用 CWnd::FromHandle。

idFrom

发送通知的控件的标识符。

code

通知代码。这个成员可以是一个特定于一个控件类型的值，比如 TBN_BEGINADJUST 或 TTN_NEEDTEXT，或者它可以是下面列出的普通通知值中的一个：

- NM_CLICK 用户在控件中单击了鼠标的左按钮。
- NM_DBLCLK 用户在控件中双击了鼠标的左按钮。
- NM_KILLFOCUS 控件失去了输入焦点。
- NM_OUTOFMEMORY 由于没有足够的可用内存，控件无法完成一次操作。
- NM_RCLICK 用户在控件中单击了鼠标的右按钮。
- NM_RDBLCLK 用户在控件中双击了鼠标的右按钮。
- NM_RETURN 控件拥有输入焦点，并且用户按下了 ENTER 键。
- NM_SETFOCUS 控件已经接收了输入焦点。

TBNOTIFY 结构包含下列成员：

```
typedef struct {
  NMHDR hdr;                    // 对所有 WM_NOTIFY 消息都通用的信息
```

```
    int iItem;                    // 与通知相关联的控件的的索引
    TBBUTTON tbButton;            // 与通知相关联的控件的信息
    int cchText;                  // 按钮文本中的字符数
    LPSTR lpszText;               // 按钮文本的地址
} TBNOTIFY, FAR* LPTBNOTIFY;
```

hdr

对所有 WM_NOTIFY 消息都通用的信息。

iItem

与通知相关联的控件的的索引。

tbButton

包含与通知相关联的工具条控件信息的 BBUTTON 结构。

cchText

按钮文本中的字符数。

lpszText

指向按钮文本的指针。

工具条发送下列信息：

- TBN_BEGINADJUST 当用户开始定制一个工具条控件时发送此通知。指针指向的 NMHDR 结构包含了有关这个通知的信息。处理程序不需要返回任何值。
- TBN_BEGINDRAG 当用户开始拖动一个工具条控件中的按钮时发送此通

知。指针指向一个 TBNOTIFY 结构。iItem 成员包含了被拖动的按钮的从零开始的索引。处理程序不需要返回任何特定的值。

- TBN_CUSTHELP 当用户在 Customize Toolbar 对话框中选择了 Help 按钮时发送此通知。没有返回值。指针指向一个 NMHDR 结构，该结构包含了有关这个通知消息的信息。处理程序不需要返回任何特定的值。
- TBN_ENDADJUST 当用户停止定制有关工具条时发送此通知。指针指向一个 NMHDR 结构，该结构包含了有关此通知消息的信息。处理程序不需要返回任何特定的值。
- TBN_ENDDRAG 当用户停止拖动一个工具条控件中的按钮时发送此通知。指针指向一个 TBNOTIFY 结构。iItem 成员包含了被拖动的按钮的从零开始的索引。处理程序不需要返回任何特定的值。
- TBN_GETBUTTONINFO 当用户正在定制一个工具条控件时发送此通知。工具条使用这个通知消息来获取 Customize Toolbar 对话框需要的信息。指针指向一个 TBNOTIFY 结构。iItem 成员指定了一个按钮的从零开始的索引。lpszText 和 cchText 成员指定当前按钮文本的地址和以字符数表示的长度。应用程序应该用有关这个按钮的信息来填充该结构。如果按钮的信息被拷贝到了结构中，则返回 TRUE；否则就返回 FALSE。
- TBN_QUERYDELETE 当用户正在定制一个工具条时发送此消息，以确定一个按钮是否可以被从工具条控件中删除。指针指向一个 TBNOTIFY 结构。iItem 成员包含了要被删除的按钮的从零开始的索引。如果允许该按钮被删除则返回 TRUE；否则返回 FALSE 来禁止该按钮被删除。
- TBN_QUERYINSERT 当用户正在定制一个工具条控件时发送此通知，以

确定一个按钮是否可以被插入到指定按钮的左边。指针指向一个 TBNOTIFY 结构。iItem 成员包含了要被插入的按钮的从零开始的索引。如果允许一个按钮被插入到给定按钮的前面，则返回 TRUE；否则返回 FALSE 来禁止该按钮被插入。

- TBN_RESET 当用户重新设置 Customize Toolbar 对话框时发送此通知。指针指向一个 NMHDR 结构，该结构包含了有关这个通知消息的信息。处理程序不需要返回任何特定的值。
- TBN_TOOLBARCHANGE 在用户定制完一个工具条控件之后发送此通知。指针指向一个 NMHDR 结构，该结构包含了有关这个通知消息的信息。处理程序不需要返回任何特定的值。

成员函数

CToolBarCtrl::AddBitmap

int AddBitmap(int *nNumButtons*, UINT *nBitmapID*);
int AddBitmap(int *nNumButtons*, CBitmap* *pBitmap*);

返回值

如果成功则返回第一个新图像的从零开始的索引；否则返回-1。

参数

nNumButtons
  位图中的按钮图像的数目。

nBitmapID
  包含按钮图像或要增加图像的位图的资源标识符。

pBitmap
  指向包含按钮图像或要增加图像的 CBitmap 对象的指针。

说明

此成员函数用来将一个或多个按钮图像添加到保存在该工具条控件中的按钮图像列表中。在向工具条添加位图之前，你可以使用 Windows API CreateMappedBitmap 来映射颜色。

如果你传递的是一个指向 CBitMap 对象的指针，则你必须确保直到工具条被销毁的时候该位图才会被销毁。

请参阅 CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton, CToolBarCtrl::AddString, CToolBarCtrl::AddStrings

CToolBarCtrl::AddButtons

BOOL AddButtons(int *nNumButtons*, LPTBBUTTON *lpButtons*);

返回值

如果成功则返回非零值；否则返回零。

参数

nNumButtons

要增加的按钮数。

lpButtons

一个 TBBUTTON 结构数组的地址，该结构包含关于要添加的按钮的信息。该数组的元素数目必须与 *nNumButtons* 指定的按钮数一样。

说明

此成员函数用来向一个工具条控件中添加一个或多个按钮。

lpButtons **指针指向一个 TBBUTTON 结构数组。每一个与被添加的按钮相关联的 TBBUTTON 结构包含了按钮的风格，图像和/或字符串，命令 ID，状态和用户定义的数据：**

typedef struct _TBBUTTON {

```
    int iBitmap;           // zero-based index of button image
    int idCommand;         // command to be sent when button pressed
    BYTE fsState;          // button state--see below
    BYTE fsStyle;          // button style--see below
    DWORD dwData;          // application-defined value
    int iString;           // zero-based index of button label string
} TBBUTTON;
```

其成员描述如下：

iBitmap

按钮图像的从零开始的索引。如果这个按钮没有图像，则这个成员是NULL。

idCommand

与此按钮相关联的命令标识符。当按钮被选择时，这个标识符在一个WM_COMMAND消息中被发送。如果fsStyle成员的值为TBSTYLE_SEP，则这个成员必须是零。

fsState

按钮的状态标志。它可以是下面列出的值的一个组合：

- TBSTATE_CHECKED 该按钮具有TBSTYLE_CHECKED风格并且被按下。
- TBSTATE_ENABLED 按钮接收用户输入。一个不具有这个状态的按

钮是不接收用户输入的，并且变灰。

- TBSTATE_HIDDEN 按钮不可见，并且不能接收用户输入。
- TBSTATE_INDETERMINATE 按钮是变灰的。
- TBSTATE_PRESSED 按钮被按下。
- TBSTATE_WRAP 按钮之后是一个分隔线。此按钮还必须具有TBSTATE_ ENABLED 状态。

fsStyle

按钮风格。它可以是下列值的一个组合：

- TBSTYLE_BUTTON 创建一个标准的按钮。
- TBSTYLE_CHECK 创建一个每次用户点击时可以在按下和弹起状态间切换的按钮。该按钮则处于按下状态时有一种不同的背景颜色。
- TBSTYLE_CHECKGROUP 创建一个核选按钮，它被选择后一直处于按下状态，直到同组中的另一个按钮被按下时它才弹起。
- TBSTYLE_GROUP 创建一个被选择后一直处于按下状态，直到同组中的另一个按钮被按下时它才弹起的按钮。
- TBSTYLE_SEP 创建一个分隔线，为按钮组之间提供一个小的间距。具有这个风格的按钮是不接收用户输入的。

dwData

用户定义的数据。

iString

要用来作为按钮的标签的字符串的从零开始的索引。如果这个按钮没有

字符串则这个值为 NULL。

你提供的图像和/或字符串的索引必须是已经通过调用 AddBitmap，AddString，和/或 AddStrings 被添加到工具条控件的列表中的值。

请参阅 CToolBarCtrl::InsertButton, CToolBarCtrl::DeleteButton, CToolBarCtrl::AddBitmap, CToolBarCtrl::AddString, CToolBarCtrl::AddStrings

CToolBarCtrl::AddString

int AddString(UINT *nStringID*);

返回值

如果成功则返回被添加的第一个新字符串的从零开始的索引；否则返回-1。

参数

nStringID
要添加到工具条控件的字符串列表中去的字符串资源的资源标识符。

说明

此成员函数用来将一个被作为资源 ID 传递的新字符串添加到工具条的内部字符串列表中。

请参阅 CToolBarCtrl::AddStrings, CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton, CToolBarCtrl::AddBitmap

CToolBarCtrl::AddStrings

int AddStrings(LPCTSTR *lpszStrings*);

返回值

如果成功则返回被添加的第一个新字符串的从零开始的索引；否则返回-1。

参数

lpszStrings

一个包含要添加到工具条的字符串列表中去的一个或多个以空字符结尾的字符串的缓冲区的地址。最后一个字符串必须以两个空字符结尾。

说明

此成员函数用来将一个或多个新字符串添加到一个工具条控件的可用的字符串列表中。缓冲区中的字符串必须以空字符来分隔。

你必须确保最后一个字符串有两个空结尾符。为了准确地格式化一个常量字符串，你应该按如下的方式来写：

//　**自动添加一个空字符。**

```
lpszStrings = "Only one string to add\0";
```

或

```
// 一次调用添加三个字符串
lpszStrings = "String 1\0String 2\0String 3\0";
```

你不能给这个函数传递一个 CString **对象，因为在一个** CStrin **中不可能有多于一个的空字符。**

请参阅 CToolBarCtrl::AddString, CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton, CToolBarCtrl::AddBitmap

CToolBarCtrl::AutoSize

```
void AutoSize( );
```

说明

此成员函数用来调整整个工具条控件的尺寸。当父窗口的尺寸改变或当工具条的尺寸改变（比如当你设置按钮或位图尺寸或添加字符串时）时，你应该调用这个函数。

请参阅 CToolBarCtrl::SetBitmapSize, CToolBarCtrl::SetButtonSize, CToolBarCtrl::AddString, CToolBarCtrl::AddStrings

CToolBarCtrl::CheckButton

BOOL CheckButton(int *nID*, BOOL *bCheck* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

nID
要被核选或清除的按钮的命令标识符。

bCheck
如果是 TRUE 则核选该按钮，如果是 FALSE 则清除该按钮。

说明

此成员函数用来核选或清除工具条控件中的一个给定按钮。当一个按钮被核选时，它看起来就像被按下一样。如果你想改变多于一个的按钮状态，可以考虑调用 SetState 来代替。

请参阅 CToolBarCtrl::IsButtonChecked, CToolBarCtrl::EnableButton, CToolBarCtrl::PressButton, CToolBarCtrl::HideButton, CToolBarCtrl::Indeterminate,

CToolBarCtrl::GetState, CToolBarCtrl::SetState

CToolBarCtrl::CommandToIndex

UINT CommandToIndex(UINT *nID*) const;

返回值

返回与此命令 ID **相关联的按钮的从零开始的索引。**

参数

nID
  你想要查找其按钮索引的命令 ID。

说明

此成员函数用来获取与指定的命令标识符相关联的按钮的从零开始的索引。

请参阅 CToolBarCtrl::SetCmdID, CToolBarCtrl::GetButton, CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton

CToolBarCtrl::Creat

BOOL Create(DWORD *dwStyle*, const RECT& *rect*, CWnd* *pParentWnd*, UINT nID);

返回值

如果成功则返回非零值；否则返回零。

参数

dwStyle

指定工具条控件的风格。工具条必须总是具有 WS_CHILD **风格。另外，你可以为工具条指定在说明部分描述的工具条风格和窗口风格的任意组合。**

rect

随意指定该工具条控件的尺寸和位置。它可以是一个 CRect **对象或一个** RECT **结构。**

pParentWnd

指定工具条控件的父窗口。它不能是 NULL**。**

nID

指定工具条控件的 ID**。**

说明

构造一个 CToolBarCtrl 对象分两步。首先调用构造函数，然后调用 Create 来创建该工具条控件并将它与该 CToolBarCtrl 对象连接。

工具条控件自动设置工具条窗口的尺寸和位置。其高度依赖于工具条中的按钮的高度。其宽度与父窗口的客户区的宽度一样。CCS_TOP 和 CCS_BOTTOM 风格决定了工具条是位于客户区的顶部还是底部。缺省的，一个工具条具有 CCS_TOP 风格。

对一个工具条可以使用下面的窗口风格。

- WS_CHILD 总是使用
- WS_VISIBLE 经常使用
- WS_DISABLED 很少使用

接着，你还可以使用一种或多种通用控件风格：

- CCS_ADJUSTABLE 允许用户定制工具条。如果使用了这个风格，则工具条的属主窗口必须处理由工具条发送的定制调整消息，就像在 CToolBarCtrl：处理定制通知中描述的一样。
- CCS_BOTTOM 使控件将自己定位在父窗口客户区的底部，并将自己的宽度设置得与父窗口的宽度一样。
- CCS_NODIVIDER 禁止在控件的顶部绘制两个像素的加亮。
- CCS_NOHILITE 禁止在控件的顶部绘制一个像素的加亮。
- CCS_NOMOVEY 使控件水平地，而不是垂直地调整自己的尺寸和移动自

己，作为对一个 WM_SIZE 消息的响应。如果使用了 CCS_NORESIZE 风格，则这个风格就不适用了。

- CCS_NOPARENTALIGN 禁止控件自动移动到父窗口的顶部或底部。控件保持它在父窗口中的位置，不管父亲窗口的尺寸是怎么改变的。如果还使用了 CCS_TOP 或 CCS_BOTTOM 风格，则高度被调整为缺省值，但位置和宽度仍然保持不变。
- CCS_NORESIZE 当设置控件的产生尺寸或新尺寸时，禁止控件使用缺省的宽度和高度。控件使用在创建或改变大小时指定的宽度和高度。
- CCS_TOP 使控件将自己定位在父窗口的客户区的顶部，并将宽度设置为与父窗口的宽度一样。控件缺省的就具有这个风格。

最后，可以对控件或按钮本身使用工具条风格的一个组合。在“Platform SDK”中的“工具条控件和按钮风格”中描述了这些风格。

请参阅 CToolBarCtrl::CToolBarCtrl, CToolBarCtrl::SetButtonStructSize

CToolBarCtrl::CToolBarCtrl

CToolBarCtrl( );

说明

此成员函数用来构造一个 CToolBarCtrl 对象。你必须调用 Create 来使工具条可用。

请参阅 CToolBarCtrl::Create

CToolBarCtrl::Customize

void Customize( );

说明

此成员函数用来显示 Customize Toolbar 对话框。该对话框允许用户通过增加或删除按钮来定制工具条。

要支持定制，你的工具条的父窗口必须对定制通知消息作出处理，就象在 CToolBarCtrl::处理定制通知中所描述的那样。你的工具条还必须使用 CCS_ADJUSTABLE 风格创建，就象在 CToolBarCtrl::Create 中描述的那样。

请参阅 CToolBarCtrl::Handling Customization Notifications

CToolBarCtrl::DeleteButton

BOOL DeleteButton(int *nIndex*);

返回值

如果成功则返回非零值；否则返回零。

参数

nIndex
　　要删除的按钮的从零开始的索引。

说明

此成员函数用来从工具条控件中删除一个按钮。

请参阅 CToolBarCtrl::AddButtons, CToolBarCtrl::AutoSize, CToolBarCtrl::InsertButton

CToolBarCtrl::EnableButton

BOOL EnableButton(int *nID*, BOOL *bEnable* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

nID
　　要使其无效或有效的按钮的标识符。

bEnable

如果是 TRUE 则表示要使按钮有效；是 FALSE 表示要使按钮无效。

说明

此成员函数用来使一个工具条控件中的指定按钮有效或无效。当一个按钮已经是有效的时，它就可以被按下或核选。如果你想改变多于一个的按钮状态，可以考虑调用 SetState。

请参阅 CToolBarCtrl::IsButtonEnabled, CToolBarCtrl::CheckButton, CToolBarCtrl::PressButton, CToolBarCtrl::HideButton, CToolBarCtrl::Indeterminate, CToolBarCtrl::GetState, CToolBarCtrl::SetState

CToolBarCtrl::GetAnchorHighlight

UINT GetBitmapFlags( ) const;

返回值

如果返回值是非零值，则可以进行定位加亮。如果返回的是零，则不能进行定位加亮。

说明

此成员函数用来实现 Win32 消息 TB_GETANCHORHIGHLIGHT，就像在“Platform SDK”中描述的一样。

请参阅 CToolBarCtrl::SetAnchorHighlight

CToolBarCtrl::GetBitmapFlags

UINT GetBitmapFlags( ) const;

返回值

如果显示可以支持大工具条位图，则返回具有 TBBF_LARGE 标志集的 UINT，否则就清除。

说明

此成员函数用来从工具条中获取位图标志。应该在创建工具条之后，但在向工具条添加位图之前调用这个函数。

返回值说明了显示是否支持大位图。如果显示支持大位图，并且你选择了使用它们，则在使用 AddBitmap 添加你的大位图之前，调用 SetBitmapSize 和 SetButtonSize。

请参阅 CToolBarCtrl::AddBitMap, CToolBarCtrl::SetBitmapSize, CToolBarCtrl::SetButtonSize

CToolBarCtrl::GetButton

BOOL GetButton(int *nIndex*, LPTBBUTTON *lpButton*) const;

返回值

如果成功则返回非零值；否则返回零。

参数

nIndex
要获取其信息的按钮的从零开始的索引。

lpButton
用来接收按钮信息的拷贝的 TBBUTTON 结构的地址。参见 CToolBarCtrl::AddButtons 可以获得有关 TBBUTTON 结构的信息。

说明

此成员函数用来获取一个工具条控件中的指定按钮的信息。

请参阅 CToolBarCtrl::GetState, CToolBarCtrl::SetState,

CToolBarCtrl::GetButtonCount,CToolBarCtrl::GetItemRect,CToolBarCtrl::CommandToIndex,CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton

CToolBarCtrl::GetButtonCount

int GetButtonCount( ) const;

返回值

返回按钮的数目。

说明

此成员函数用来获取工具条控件中的当前按钮的数目。

请参阅 CToolBarCtrl::GetButton, CToolBarCtrl::GetState, CToolBarCtrl::GetItemRect, CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton, CToolBarCtrl::DeleteButton

CToolBarCtrl::GetButtonInfo

BOOL GetButtonInfo(int *nID* , TBBUTTONINFO* *ptbbi*) const;

返回值

如果成功则返回非零值；否则返回零。

参数

nID
    按钮的标识符。

pbbi
    一个指向 TBBUTTONINFO 结构的指针，该结构用来接收按钮的信息。

说明

此成员函数用来实现 Win32 消息 TB_GETBUTTONINFO，就像在“Platform SDK”中描述的一样。

请参阅 CToolBarCtrl::SetButtonInfo

CToolBarCtrl::GetButtonSize

DWORD GetButtonSize( ) const;

返回值

返回一个 DWORD **值，其** LOWORD **和** HIWORD **中分别包含了宽度和高度值。**

说明

此成员函数用来获取一个工具条按钮的尺寸。

请参阅 CToolBarCtrl::GetButtonInfo

CToolBarCtrl::GetDisabledImageList

CImageList* GetDisabledImageList( ) const;

返回值

返回一个指向 CImageList **对象的指针。如果没有被设置为无效的图像列表则返回** NULL**。**

说明

此成员函数用来实现 Win32 **消息** TB_GETDISABLEDIMAGELIST**，就像在**"Platform SDK"**中描述的一样。**

GetDisabledImageList **的** MFC **实现使用包含了工具条控件的按钮图像的**

CImageList对象，而不是使用一个图像列表的句柄。

请参阅 CToolBarCtrl::SetDisabledImageList, CToolBarCtrl::GetHotImageList, CToolBarCtrl::GetImageList

CToolBarCtrl::GetDropTarget

HRESULT GetDropTarget(IDropTarget** *ppDropTarget*) const;

返回值

返回一个HRESULT值，指明操作的成功或失败。

参数

ppDropTarget
一个指向 IDropTarget 接口指针的指针。如果发生了一个错误，则一个NULL指针将被放置在这个地址中。

说明

此成员函数用来实现 Win32 消息 TB_GETOBJECT，就像在“Platform SDK”中描述的一样。

CToolBarCtrl::GetExtendedStyle

DWORD GetExtendedStyle( ) const;

返回值

返回一个代表此工具条控件中当前使用的扩展风格的结构。参见“Platform SDK”中的“工具条扩展风格”可以获得风格的列表。

说明

此成员函数用来实现Win32消息TB_GETEXTENDEDSTYLE，就像在“Platform SDK”中描述的一样。

请参阅 CToolBarCtrl::SetExtendedStyle

CToolBarCtrl::GetHotImageList

CImageList* GetHotImageList( ) const;

返回值

返回一个指向CImageList对象的指针，如果没有被设置为无效的图像列表则返回NULL。

说明

此成员函数用来实现 Win32 消息 TB_GETHOTIMAGELIST，就像在“Platform SDK”中描述的一样。

当鼠标指针位于一个热点按钮上时，该按钮被加亮显示。

请参阅 CToolBarCtrl::GetDisabledImageList, CToolBarCtrl::GetImageList

CToolBarCtrl::GetHotItem

```
int GetHotItem( ) const;
```

返回值

返回一个工具条中的热点项的从零开始的索引。

说明

此成员函数用来实现 Win32 消息 TB_GETHOTITEM，就像在“Platform SDK”中描述的一样。

CToolBarCtrl::GetImageList

CImageList* GetImageList ( ) const;

返回值

返回一个指向 CImageList **对象的指针，如果没有被设置为无效的图像列表则返回** NULL。

说明

此成员函数用来实现 Win32 **消息** TB_GETIMAGELIST**，就像在**“Platform SDK”**中描述的一样。**

CToolBarCtrl::GetInsertMark

void GetInserMark (TBINSERTMARK* *ptbim*) const;

参数

ptbim

一个指向 TBINSERTMARK **结构的指针，该结构用来接收插入标记。**

说明

此成员函数用来实现 Win32 **消息** TB_GETINSERTMARK**，就像在**“Platform SDK”**中描述的一样。**

请参阅 CToolBarCtrl::SetInsertMark

CToolBarCtrl::GetInsertMarkColor

```
COLORREF GetInsertMarkColor( ) const;
```

返回值

返回一个包含当前插入标记的颜色的 COLORREF **值。**

说明

此成员函数用来实现 Win32 **消息** TB_GETINSERTMARKCOLOR**，就像在**“Platform SDK”**中描述的一样。**

请参阅 CToolBarCtrl::SetInsertMarkColor

CToolBarCtrl::GetItemRect

BOOL GetItemRect(int *nIndex*, LPRECT *lpRect*) const;

返回值

如果成功则返回非零值；否则返回零。

参数

nIndex

要获取其信息的按钮的从零开始的索引。

lpRect

一个用来接收边界矩形的坐标的 RECT **结构或** CRect **对象的地址。**

说明

此成员函数用来接收工具条控件中的一个按钮的边界矩形。此函数不接收状态被设置为 TBSTATE_HIDDEN **的按钮的边界矩形。**

请参阅 CToolBarCtrl::GetButton, CToolBarCtrl::GetButtonCount, CToolBarCtrl::GetState, CToolBarCtrl::SetButtonSize, CToolBarCtrl::SetBitmapSize

CToolBarCtrl::GetMaxSize

BOOL GetMaxSize (LPSIZE *pSize*) const;

返回值

如果成功则返回非零值；否则返回零。

参数

pSize

一个指向用来接收项的尺寸的 SIZE **结构的指针。**

说明

此成员函数用来实现 Win32 **消息** TB_GETMAXSIZE**，就像在“** Platform SDK **”中描述的一样。**

CToolBarCtrl::GetMaxTextRows

int GetMaxTextRows ( ) const;

返回值

返回一个工具条按钮上显示的文本的最大行数。

说明

此成员函数用来获取一个工具条按钮上显示的文本的最大行数。

请参阅 CToolBarCtrl::SetMaxTextRows

CToolBarCtrl::GetRect

BOOL GetRect(int *nID*, LPRECT *lpRect*) const;

返回值

如果成功则返回 TRUE**；否则返回** FALSE**。**

参数

nID

　　按钮标识符。

lpRect

　　一个指向用来接收边界矩形信息的 RECT **结构的指针。**

说明

此成员函数用来实现 Win32 消息 TB_GETRECT，就像在“Platform SDK”中描述的一样。

CToolBarCtrl::GetRows

int GetRows( ) const;

返回值

返回工具条中当前显示的按钮的行数。

说明

此成员函数用来获取工具条控件中当前显示的按钮的行数。注意，除非工具条是用 TBSTYLE_WRAPABLE 风格创建的，否则行数总是 1。

请参阅 TBSTYLE_WRAPABLE in CToolBarCtrl::Create, CToolBarCtrl::SetRows

CToolBarCtrl::GetState

int GetState(int *nID*) const;

返回值

如果成功则返回按钮的状态信息；否则返回-1。按钮的状态信息可以是在CToolBarCtrl::AddButtons中所列出的值的一个组合。

参数

nID
要获取其信息的按钮的命令标识符。

说明

此成员函数用来获取有关一个工具条控件中的指定按钮的状态的信息，比如它是否是有效，被按下，或被核选。

如果你要处理多于一个的按钮状态，则这个函数是特别方便的。如果只需要获取一个状态，则可以使用下列成员函数：IsButtonEnabled，IsButtonChecked，IsButtonPressed，IsButtonHidden，或IsButtonIndeterminate。但是，GetState成员函数是检测TBSTATE_WRAP按钮状态的唯一方法。

请参阅 CToolBarCtrl::SetState, CToolBarCtrl::GetItemRect, CToolBarCtrl::IsButtonEnabled, CToolBarCtrl::IsButtonChecked,

CToolBarCtrl::IsButtonPressed, CToolBarCtrl::IsButtonHidden,

CToolBarCtrl::IsButtonIndeterminate

CToolBarCtrl::GetStyle

DWORD GetStyle( ) const;

返回值

返回一个包含了工具条控件风格的一个组合的 DWORD**，就像**“Platform SDK”**中描述的一样。**

说明

此成员函数用来获取一个工具条控件当前使用的风格。

请参阅 CToolBarCtrl::SetStyle

CToolBarCtrl::GetToolTips

CToolTipCtrl* GetToolTips( ) const;

返回值

返回指向一个与工具条相关联的 CToolTipCtrl **对象的指针。如果该工具条没有关联的工具提示控件，则返回** NULL**。**

说明

此成员函数用来获取与工具条控件相关联的工具提示控件（如果有的话）的句柄。由于工具条控件通常会创建并维护它自己的工具提示控件，所以大多数程序不需要调用这个函数。

请参阅 CToolBarCtrl::SetToolTips, CToolBarCtrl: Handling Tool Tip Notifications, CToolTipCtrl

CToolBarCtrl::HitTest

int HitTest(LPPOINT *ppt*) const;

返回值

返回一个指明了工具条上的一个点的位置的整数值。如果这个值是零或正值，则这个返回值表示的是该点所在的非分隔线项的从零开始的索引。

如果返回值是负值，则该点不在一个按钮中。该返回值的绝对值是一个分隔线或最近的非分隔线项的索引。

参数

ppt

一个指向POINT结构的指针，该结构的x成员包含了点击测试的x坐标，其y成员包含了点击测试的y坐标。这个坐标是相对于工具条的客户区的。

说明

此成员函数用来实现Win32消息TB_HITTEST的行为，就像在“Platform SDK”中描述的一样。

CToolBarCtrl::HideButton

BOOL HideButton(int *nID*, BOOL *bHide* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

nID
要被显示或隐藏的按钮的命令标识符。

bHide
如果是TRUE则隐藏此按钮，是FALSE则显示此按钮。

说明

此成员函数用来隐藏或显示一个工具条控件中的指定按钮。如果你想改变多于一个的按钮状态，则可以考虑调用 SetState **来代替。**

请参阅 CToolBarCtrl::IsButtonHidden, CToolBarCtrl::EnableButton, CToolBarCtrl::CheckButton,CToolBarCtrl::PressButton,ToolBarCtrl::Indeterminate, CToolBarCtrl::GetState, CToolBarCtrl::SetState

CToolBarCtrl::Indeterminate

BOOL Indeterminate(int *nID*, BOOL *bIndeterminate* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

nID

要清除或设置其不确定状态的按钮的命令标识符。

bIndeterminate

如果是 TRUE **则设置指定按钮的不确定状态，是** FALSE **则清除它。**

说明

此成员函数用来设置或清除一个工具条控件中的指定按钮的不确定状态。不确定的按钮被显示为灰色，例如，当一个字处理器中被选择的文本同时包含加粗的或正常的字符时，该处理器中的工具条上的加粗按钮就显示为灰色。

如果你希望改变多于一个的按钮状态，可以考虑调用 SetState **来代替。**

请参阅 Button styles in CToolBarCtrl::AddButtons,

CToolBarCtrl::IsButtonIndeterminate,

CToolBarCtrl::EnableButton,CToolBarCtrl::CheckButton,

CToolBarCtrl::PressButton,CToolBarCtrl::HideButton, CToolBarCtrl::GetState, CToolBarCtrl::SetState

CToolBarCtrl::InsertButton

BOOL InsertButton(int *nIndex*, LPTBBUTTON *lpButton*);

返回值

如果成功则返回非零值；否则返回零。

参数

nIndex

一个按钮的从零开始的索引。此函数将一个新按钮插入到这个按钮的左边。

lpButton

一个包含要被插入的按钮的信息的 TBBUTTON 结构的指针。参见 CToolBarCtrl:: AddButtons 可以获得有关 TBBUTTON 结构的描述。

说明

此成员函数用来在一个工具条控件中插入一个按钮。

你提供的图像和/或字符串的索引必须是先前通过调用 AddBitmap，AddString 和/或 AddStrings 已经添加到工具条控件的列表中去了的值。

请参阅 CToolBarCtrl::AddButtons, CToolBarCtrl::DeleteButton, CToolBarCtrl::AddBitmap, CToolBarCtrl::AddString, CToolBarCtrl::AddStrings

CToolBarCtrl::InsertMarkHitest

BOOL InsertMarkHitTest(LPPOINT *ppt*, LPTBINSERTMARK *ptbim*) const;

返回值

如果成功则返回非零值；否则返回零。

参数

ppt

一个指向包含点击测试坐标的 POINT **结构，该坐标相对于工具条的客户区。**

ptbim

一个指向用来接收插入标记信息的 TBINSERTMARK **结构的指针。**

说明

此成员函数用来实现 Win32 **消息** TB_INSERTMARKHITTEST **的行为，就像在**"Platform SDK"**中描述的一样。**

请参阅 CToolBarCtrl::GetInsertMark, CToolBarCtrl::SetInsertMark

CToolBarCtrl::IsButtonChecked

BOOL IsButtonChecked(int *nID*) const;

返回值

如果按钮被核选则返回非零值；否则返回零。

参数

nID
  工具条中的按钮的命令标识符。

说明

此成员函数用来确定一个工具条控件中的指定按钮是否被核选。如果你要获取多于一个的按钮状态，考虑调用 GetState。

请参阅 CToolBarCtrl::CheckButton, CToolBarCtrl::GetState, CToolBarCtrl::SetState, CToolBarCtrl::IsButtonEnabled, CToolBarCtrl::IsButtonPressed, CToolBarCtrl::IsButtonHidden, CToolBarCtrl::IsButtonIndeterminate

CToolBarCtrl::IsButtonEnabled

BOOL IsButtonEnabled(int *nID*) const;

返回值

如果按钮是有效的则返回非零值；否则返回零。

参数

nID
工具条中的按钮的命令标识符。

说明

此成员函数用来确定一个工具条控件中的指定按钮是否是有效的。如果你要获取多于一个的按钮状态，考虑调用 GetState。

请参阅 CToolBarCtrl::EnableButton, CToolBarCtrl::GetState, CToolBarCtrl::SetState, CToolBarCtrl::IsButtonChecked, CToolBarCtrl::IsButtonPressed, CToolBarCtrl::IsButtonHidden,CToolBarCtrl::IsButtonIndeterminate,CToolBarCtrl::IsButtonHighlighted

CToolBarCtrl::IsButtonHidden

BOOL IsButtonHidden(int *nID*) const;

返回值

如果按钮被隐藏则返回非零值；否则返回零。

参数

nID
　　工具条中的按钮的命令标识符。

说明

此成员函数用来确定一个工具条控件中的指定按钮是否是隐藏的。如果你要获取多于一个的按钮状态，考虑调用 GetState。

请参阅 CToolBarCtrl::HideButton, CToolBarCtrl::GetState, CToolBarCtrl::SetState, CToolBarCtrl::IsButtonEnabled, CToolBarCtrl::IsButtonChecked, CToolBarCtrl::IsButtonPressed,CToolBarCtrl::IsButtonIndeterminate,CToolBarCtrl::IsButtonHighlighted

CToolBarCtrl::IsButtonHighlighted

Bool IsButtonHighlighted(int *nID*) const;

返回值

如果按钮被加亮则返回非零值；否则返回 0。

参数

nID
　　工具条中的按钮的命令标识符。

说明

此成员函数用来检查一个工具条控件中的指定按钮是否处于加亮状态。

请参阅　CToolBarCtrl::IsButtonChecked, CToolBarCtrl::IsButtonIndeterminate, CToolBarCtrl::IsButtonHidden, CToolBarCtrl::IsButtonEnabled, CToolBarCtrl::IsButtonPressed, CToolBarCtrl::MarkButton

CToolBarCtrl::IsButtonIndeterminate

BOOL IsButtonIndeterminate(int *nID*) const;

返回值

如果按钮处于不确定状态则返回非零值；否则返回零。

参数

nID

　　工具条中的按钮的命令标识符。

说明

此成员函数用来确定一个工具条控件中的指定按钮是否是不确定的。不确定按钮被显示成灰色，例如，当一个字处理器中被选择的文本同时包含加粗的或正常的字符时，该处理器中的工具条上的加粗按钮就显示为灰色。

如果你要获取多于一个的按钮状态，考虑调用 GetState。

请参阅 CToolBarCtrl::Indeterminate, CToolBarCtrl::GetState, CToolBarCtrl::SetStateCToolBarCtrl::IsButtonEnabled, CToolBarCtrl::IsButtonChecked, CToolBarCtrl::IsButtonPressed,

CToolBarCtrl::IsButtonHidden, CToolBarCtrl::IsButtonHighlighted

CToolBarCtrl::IsButtonPressed

BOOL IsButtonPressed(int *nID*) const;

返回值

如果按钮被按下则返回非零值，否则返回零。

参数

nID

工具条中的按钮的命令标识符。

说明

此成员函数用来确定一个工具条控件中的指定按钮是否被按下。如果你要获取多于一个的按钮状态，考虑调用 GetState。

请参阅 CToolBarCtrl::PressButton, CToolBarCtrl::GetState, CToolBarCtrl::SetState, CToolBarCtrl::IsButtonEnabled,

CToolBarCtrl::IsButtonChecked,
CToolBarCtrl::IsButtonHidden,CToolBarCtrl::IsButtonIndeterminate,CToolBarCtrl::

IsButtonHighlighted

CToolBarCtrl::LoadImages

void LoadImages(int *iBitmapID*, HINSTANCE *hinst*);

参数

iBitmapID

包含要装入的图像的位图的 ID。要指定你自己的位图资源，可以将这个参数设置为一个位图资源的 ID，并将 *hInst* 设置为 NULL。你的位图资源将被添加到图像列表作为一个单一的图像。你可以通过将 *hinst* 设置为 HINST_COMMCTRL，将这个参数设置为下列 ID 值之一，来添加标准的，系统定义的位图：

位图 ID	描述
IDB_HIST_LARGE_COLOR	大尺寸的探测位图
IDB_HIST_SMALL_COLOR	小尺寸的探测位图
IDB_STD_LARGE_COLOR	大尺寸的标准位图
IDB_STD_SMALL_COLOR	小尺寸的标准位图
IDB_VIEW_LARGE_COLOR	大尺寸的视位图
IDB_VIEW_SMALL_COLOR	小尺寸的视位图

hinst

指向调用应用程序的程序示例句柄。这个参数这个参数可以是 HINST_COMMCTRL，用来加载一个标准的图像列表。

说明

此成员函数用来实现 Win32 **消息** TB_LOADIMAGES **的行为，就象在“** Platform SDK **”中描述的一样。**

请参阅 CToolBarCtrl::SetImageList, CToolBarCtrl::GetImageList

CToolBarCtrl::MapAccelerator

BOOL MapAccelerator (TCHAR *chAccel*, UINT* *pIDBtn*);

返回值

如果成功则返回非零值；否则返回零。

参数

chAccel

要被映射的加速键字符。这个字符与按钮文本中的加下划线的字符是一样的。

pIDBtn

一个指向用来接收某个按钮的命令标识符的 UINT **的指针。此按钮用来响应在** chAccel **中指定的加速键。**

说明

此成员函数用来实现 Win32 消息 TB_MAPACCELERATOR 的行为，就像在“Platform SDK”中描述的一样。

CToolBarCtrl::MarkButton

BOOL MardButton(int *nID*, BOOL *fHighlight* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

nID

按钮标识符。

fHighlight

指定要设置的加亮状态。缺省的，这个值是 TRUE。如果将其设置为 FALSE，则此按钮被设置为它的缺省状态。

说明

此成员函数用来实现 Win32 **消息** TB_MARKBUTTON **的行为，就像在**“Platform SDK”**中描述的一样。**

请参阅 CToolBarCtrl::IsButtonHighlighted

CToolBarCtrl::MoveButton

BOOL MoveButton(UINT *nOldPos*, UINT *nNewPos*);

返回值

如果成功则返回非零值；否则返回零。

参数

nOldPos
　　要被移动的按钮的从零开始的索引。

nNewPos
　　按钮的终点的从零开始的索引。

说明

此成员函数用来实现 Win32 **消息** TB_MOVEBUTTON **的行为，就像在“** Platform SDK **”中描述的一样。**

CToolBarCtrl::PressButton

BOOL PressButton(int *nID*, BOOL *bPress* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

nID
  要被按下或释放的按钮的命令标识符。

bPress
  如果是 TRUE **则按下指定的按钮；如果是** FLASE **则释放指定按钮。**

说明

此成员函数用来按下或释放一个工具条控件中的指定按钮。如果你要改变多于

一个的按钮状态，可以考虑调用 SetState 来代替。

请参阅 CToolBarCtrl::IsButtonPressed, CToolBarCtrl::EnableButton, CToolBarCtrl::CheckButton,CToolBarCtrl::HideButton, CToolBarCtrl::Indeterminate, CToolBarCtrl::GetState, CToolBarCtrl::SetState

CToolBarCtrl::RestoreState

void RestoreState(HKEY *hKeyRoot*, LPCTSTR *lpszSubKey*, LPCTSTR *lpszValueName*);

参数

hKeyRoot

表示注册表中的一个当前打开的主键，或是下列预定义的保留句柄值的任何一个：

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS

lpszSubKey

指向一个以空字符结尾的字符串，该字符串包含了与某个值关联的子键

的名字。这个参数可以是空或是一个指向空字符串的指针。如果这个参数是 NULL，则该值被添加到由 hKeyRoot 参数指定的主键中。

lpszValueName

指向一个包含要获取的值的名字的字符串。如果这个名字的值在主键中没有给出，则函数将它添加到该主键中。

说明

此成员函数从参数所指定的注册表中的位置恢复工具条控件的状态。

请参阅 CToolBarCtrl::SaveState

CToolBarCtrl::SaveState

void SaveState(HKEY *hKeyRoot*, LPCTSTR *lpszSubKey*, LPCTSTR *lpszValueName*);

参数

hKeyRoot

表示注册表中的一个当前打开主键，或是下列预定义的保留句柄值的任何一个：

- HKEY_CLASSES_ROOT

- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS

lpszSubKey

指向一个以空字符结尾的字符串，该字符串包含了与某个值关联的子键的名字。这个参数可以是空或是一个指向空字符串的指针。如果这个参数是 NULL，则该值被添加到由 hKeyRoot 参数指定的主键中。

lpszValueName

指向一个包含要获取的值的名字的字符串。如果这个名字的值在主键中没有给出，则函数将它添加到该主键中。

说明

此成员函数用来将工具条控件的状态保存到由参数指定的注册表中的位置。

请参阅 CToolBarCtrl::RestoreState

CToolBarCtrl::SetAnchorHighlight

BOOL SetAnchorHighlight(BOOL *fAnchor* = TRUE);

返回值

如果成功则返回非零值；否则返回零。

参数

fAnchor

指示定位加亮是否有效。如果这个值是非零值，则定位加亮将是有效的。如果这个值是零，则不允许定位加亮。

说明

此成员函数用来实现 Win32 消息 TB_SETANCHORHIGHLIGHT 的行为，就像在“Platform SDK”中描述的一样。

请参阅 CToolBarCtrl::GetAnchorHighlight

CToolBarCtrl::SetBitmapSize

BOOL SetBitmapSize(CSize *size*);

返回值

如果成功则返回非零值；否则返回零。

参数

size

位图图像的以像素表示的宽度和高度。

说明

此成员函数用来设置要被添加到一个工具条控件中的实际位图图像的尺寸。

只有在向工具条中添加位图之前才调用这个函数。如果应用程序不显式地设置位图的尺寸，则它的缺省值为16×15像素。

请参阅 CToolBarCtrl::SetButtonSize, CToolBarCtrl::GetItemRect

CToolBarCtrl::SetButtonInfo

BOOL SetButtonInfo(int *nID*, TBBUTTONINFO* *ptbbi*);

返回值

如果成功则返回非零值；否则返回零。

参数

nID

按钮标识符。

ptbbi

一个指向用来接收按钮信息的 TBBUTTONINFO 结构的指针。

说明

此成员函数用来实现 Win32 消息 TB_SETBUTTONINFO 的行为，就像在“Platform SDK”中描述的一样。

请参阅 CToolBarCtrl::GetButtonInfo

CToolBarCtrl::SetButtonSize

```
BOOL SetButtonSize( CSize size );
```

返回值

如果成功则返回非零值；否则返回零。

参数

size

按钮的以像素表示的宽度和高度。

说明

此成员函数用来设置工具条控件中的按钮的尺寸。按钮的尺寸必须总是至少与它所装入的位图的尺寸一样。

只有在向工具条中添加位图之前才调用这个函数。如果应用程序不显式地设置按钮的尺寸，则它的缺省值为 24×22 像素。

请参阅 CToolBarCtrl::SetBitmapSize, CToolBarCtrl::GetItemRect

CToolBarCtrl::SetButtonStructSize

```
void SetButtonStructSize( int nsize );
```

参数

nsize

TBBUTTON 结构的以字节数表示的尺寸。

说明

此成员函数用来指定 TBBUTTON 结构的大小。如果你想在 TBBUTTON 结构中保存额外的数，你可以从 TBBUTTON 派生一个新的结构，增加你需要的成员，或者创建一个新的结构，该结构包含一个 TBBUTTON 结构作为它的第一

个成员。然后你可以调用这个函数来向工具条控件给出此新结构的大小。

参见 CToolBarCtrl::AddButtons 可以获得有关 TBBUTTON 结构的更多信息。

请参阅 CToolBarCtrl::Create,CToolBarCtarl::AddButtons
CToolBarCtrl::InsertButton,CToolBarCtrl::GetButton

CToolBarCtrl:: SetButtonWidth

BOOL SetButtonWidth(int *cxMin*, int *cxMax*);

返回值

如果成功则返回非零值；否则返回零。

参数

cxMin
    以像素表示的最小按钮宽度。工具条按钮的宽度不能小于这个值。

cxMax
    以像素表示的最大按钮宽度。如果按钮文本太宽，则控件用省略的点来显式它。

说明

此成员函数用来实现 Win32 **消息** TB_SETBUTTONWIDTH **的行为，就像在**“Platform SDK”**中描述的一样。**

请参阅 CToolBarCtrl::SetButtonInfo

CToolBarCtrl::SetCmdID

BOOL SetCmdID(int *nIndex*, UINT *nID*);

返回值

如果成功则返回非零值；否则返回零。

参数

nIndex

要设置其命令 ID **的按钮的从零开始的索引。**

nID

要给选定的按钮设置的命令 ID**。**

说明

此成员函数用来设置指定按钮的命令标识符，当该按钮被按下时，此命令标识符将被发送给属主窗口。

请参阅 CToolBarCtrl::CommandToIndex, CToolBarCtrl::GetButton, CToolBarCtrl::AddButtons, CToolBarCtrl::InsertButton

CToolBarCtrl::SetDisabledImageList

CImageList* SetDisabledImageList(CImageList* *pImageList*);

返回值

一个指向 CImageList **对象的指针，该对象先前被工具条控件用来显示无效按钮图像。**

参数

pImageList

一个指向 CImageList **对象的指针，该对象包含了要被工具条控件用来显示无效按钮图像的图像。**

说明

此成员函数用来实现 Win32 消息 TB_SETDISABLEDIMAGELIST 的行为，就像在“Platform SDK”中描述的一样。

SetDisabledImageList 的 MFC 实现使用的是一个包含工具条控件的无效按钮图像的 CImageList 对象，而不是一个图像列表句柄。

请参阅 CToolBarCtrl::GetDisabledImageList, CToolBarCtrl::SetHotImageList, CToolBarCtrl::SetimageList

CToolBarCtrl::SetDrawTextFlages

DWORD SetDrawTextFlage(DWORD *dwMask*, DWORD *dwDTFlags*);

返回值

返回一个包含先前的文本绘制标志的 DWORD。

参数

dwMask

在 Win32 函数 DrawText 中指定的一个或多个 DT_标志的组合。该组合说明了在绘制文本时使用 *dwDTFlags* 中的哪些位。

dwDTFlags

在 Win32 函数 DrawText 中指定的一个或多个 DT_标志的组合。该组合表明如何绘制按钮文本。当按钮文本被绘制时，这个值被传递给 DrawText。

说明

此成员函数用来实现 Win32 消息 TB_SETDRAWTEXTFLAGS 的行为，就像在“Platform SDK”中描述的一样。

此成员函数用来设置 Win32 函数 DrawText 中的标志。该函数用来绘制指定矩形中的文本，根据所设置的标志来格式化字符串。

CToolBarCtrl::SetExtendedStyle

DWORD SetExtendedStyle(DWORD *dwExStyle*) const;

返回值

返回一个代表先前的扩展风格的 DWORD。参见“Platform SDK”中的“工具条扩展风格”可以获得风格的列表。

参数

dwExStyle

指定新的扩展风格的值。这个参数可以是工具条风格的一个组合。

说明

此成员函数用来实现 Win32 **消息** TB_SETEXTENDEDSTYLE **的行为，就像在**“Platform SDK”**中描述的一样。**

请参阅 CToolBarCtrl::GetExtendedStyle

CToolBarCtrl::SetHotImageList

```
CImageList* SetHotImageList( CImageList* pImageList );
```

返回值

返回一个指向 CImageList **对象的指针，该对象先前被工具条控件用来显示热点按钮图像。**

参数

pImageList

一个指向 CImageList 对象的指针，该对象包含的图像将被工具条控件用来显示热点按钮图像。

说明

此成员函数用来实现 Win32 消息 TB_SETHOTIMAGELIST 的行为，就像在“Platform SDK”中描述的一样。

SetHotImageList 的 MFC 实现使用的是一个包含工具条控件的热点按钮图像的 CImageList 对象，而不是一个图像列表句柄。

当指针位于一个热点按钮上时，该按钮将被加亮显示。

请参阅 CToolBarCtrl::GetHotImageList, CToolBarCtrl::SetDisabledImageList, CToolBarCtrl::SetImageList

CToolBarCtrl::SetHotItem

int SetHotItem(int *nHot*);

返回值

返回先前的热点项的索引，如果没有热点项则返回-1。

参数

nHot

将要被变成热点的项的从零开始的索引号。如果这个值是-1，**则没有一个项将被变成热点。**

说明

此成员函数用来实现 Win32 **消息** TB_SETHOTITEM **的行为，就像在“**Platform SDK**”中描述的一样。**

请参阅 CToolBarCtrl::GetHotItem

CToolBarCtrl::SetImageList

CImageList* SetImageList(CImageList* *pImageList*);

返回值

返回一个指向 CImageList **对象的指针，该对象先前被工具条控件用来显示处于缺省状态的按钮图像。**

参数

pImageList

一个指向 CImageList 对象的指针，该对象包含的图像将被工具条控件用来显示缺省状态按钮图像。

说明

此成员函数用来实现 Win32 消息 TB_SETIMAGELIST 的行为，就像在“Platform SDK”中描述的一样。

SetImageList 的 MFC 实现使用的是一个包含工具条控件的缺省状态按钮图像的 CImageList 对象，而不是一个图像列表句柄。

请参阅 CToolBarCtrl::GetImageList, CToolBarCtrl::SetDisabledImageList, CToolBarCtrl::SetHotImageList

CToolBarCtrl::SetIndent

BOOL SetIndent(int *iIndent*);

返回值

如果成功则返回非零值；否则返回零。

参数

iIndent

这个值指定了以像素表示的缩进。

说明

此处于函数用来设置一个工具条控件中的第一个按钮的缩进。

CToolBarCtrl::SetInsertMark

void SetInsertMark (TBINSERTMARK* *ptbim*)

参数

ptbim

一个指向包含插入标记的 TBINSERTMARK **结构的指针。**

说明

此成员函数用来实现 Win32 **消息** TB_SETINSERTMARK **的行为，就像在“** Platform SDK **”中描述的一样。**

请参阅 CToolBarCtrl::GetInsertMark

CToolBarCtrl::SetInsertMarkColor

COLORREF SetInsertMarkColor(COLORREF *clrNew*);

返回值

返回一个包含先前的插入标记颜色的 COLORREF **值。**

参数

clrNew

一个包含新的插入标记颜色的 COLORREF **值。**

说明

此成员函数用来实现 Win32 **消息** TB_SETINSERTMARKCOLOR **的行为，就像在“** Platform SDK **”中描述的一样。**

请参阅 CToolBarCtrl::GetInsertMarkColor

CToolBarCtrl::SetMaxTextRows

BOOL SetMaxTextRows(int *iMaxRows*);

返回值

如果成功则返回非零值；否则返回零。

参数

iMaxRows

要设置的最大行数。

说明

此成员函数用来设置显示在一个工具条按钮中的文本的最大行数。

请参阅 CToolBarCtrl::GetMaxTextRows

CToolBarCtrl::SetOwner

void SetOwner(CWnd* *pWnd*);

参数

pWnd

指向将要成为工具条控件的新属主窗口的 CWnd 或 CWnd 派生对象的指针。

说明

此成员函数用来为工具条控件设置属主窗口。属主窗口是接收来自该工具条的通知的窗口。

请参阅 CToolBarCtrl::Create

CToolBarCtrl::SetRows

void SetRows(int *nRows*, BOOL *bLarger*, LPRECT *lpRect*);

参数

nRows
　　需要的行数。

bLarger
　　如果工具条不能调整到需要的行数，此参数指明是使用更多的行还是使用更少的行。

lpRect
　　指向 CRect 对象或 RECT 结构，该对象或结构用来接收工具条的新的边界矩形。

说明

此成员函数用来请求工具条控件将它自己调整为需要的行数。

如果该工具条不能将自己调整为需要的行数，则它将根据 *bLarger* 的值把自己调整为下一个较大的或较小的尺寸。如果 *bLarger* 是 TRUE，则新行数将大于请求的行数。如果 *bLarger* 是 FALSE，则新行数将小于请求的行数。

如果按钮可以被安排为给定的行数，则该给定值是有效的，并且所有行都具有相同的按钮数（可能除了最后一行）。例如，一个包含四个按钮的工具条不能被调整为三行，因为这样的安排将使最后两行都较短。如果你尝试要将它调整为三行，则如果 *bLarge* 是 TRUE，则得到的是四行，如果 *bLarger* 是 FALSE，则得到的是两行。

如果工具条中有分隔线，则有关何时给定行数有效的规则就更复杂了。按钮的安排可以这样考虑：按钮组（组中的第一个按钮之前和最后一个按钮之后都有分隔线）不会被分行排列，除非它们不能在一行中安排。

如果一个组不能正好占据完一行，则下一个组将从下一个行开始，即使是这个组可以在上一个大组结束的行中完全安排。这个规则的目的就是使两个大组之间的分隔更加显而易见。这样导致的垂直分隔线被作为行计算。

还要注意，SetRows 成员函数将总是选择会获得最小的工具条尺寸的版面规划。创建一个具有 TBSTYLE_WRAPABLE 风格的工具条，然后调整这个控件将只是简单地根据给定的控件宽度应用这个方法来画出轮廓。

只有使用TBSTYLE_WRAPABLE风格创建的函数才可以调用这个函数。

请参阅 Toolbar styles in CToolBarCtrl::Create, CToolBarCtrl::GetRows

CToolBarCtrl::SetState

BOOL SetState(int *nID*, UINT *nState*);

返回值

如果成功则返回非零值；否则返回零。

参数

nID

按钮的命令标识符。

nState

状态标志。它可以是CToolBarCtrl::AddButtons中列出的按钮状态值的一个组合。

说明

此成员函数用来设置一个工具条控件中的指定按钮的状态。

如果你想要设置多于一个的按钮状态，这个函数是特别方便的。如果只要设置一个状态，可以使用下列的成员函数：EnableButton，CheckButton，HideButton，Indeterminate，或 PressButton。

请参阅 CToolBarCtrl::GetState, CToolBarCtrl::AddButtons, CToolBarCtrl::EnableButton, CToolBarCtrl::CheckButton,CToolBarCtrl::HideButton, CToolBarCtrl::Indeterminate,CToolBarCtrl::PressButton

CToolBarCtrl::SetStyle

void SetStyle(DWORD *dwStyle*);

参数

dwStyle

一个包含工具条控件风格的组合的 DWORD，就像在“Platform SDK”中描述的一样。

说明

此成员函数用来设置一个工具条控件的风格。

请参阅 CToolBarCtrl::GetStyle

CToolBarCtrl::SetToolTips

void SetToolTips(CToolTipCtrl* *pTip*);

参数

pTip

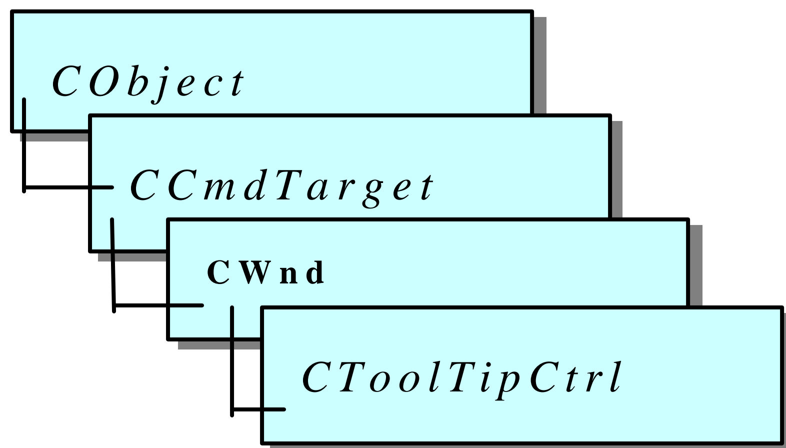
指向一个 CToolTipCtrl **对象的指针。**

说明

此成员函数用来使一个工具提示控件与一个工具条控件相关联。

请参阅 CToolBarCtrl::GetToolTips, CToolBarCtrl: Handling Tool Tip Notifications, CtoolTipCtrl

CToolTipCtrl



CToolTipCtrl 类封装了一个“工具提示控件”的性能。工具提示控件是一个小的弹出窗口，它用来显示一行描述应用程序中的一个工具的目的的文本。一个“工具”可以是一个窗口，比如说一个子窗口或控件，或者是一个窗口的客户区中的一个应用程序定义的矩形区域。一个工具提示大多数时间是隐藏的，只有在用户将光标放在一个工具上并停留大约半秒时间时，工具提示才显示出来。工具提示显示在光标的附近，当用户点击鼠标按钮或将光标从工具上离开时，工具提示消失。

CToolTipCtrl 提供了一些功能，用来控制工具提示的初始时间和持续时间，工

具提示周围的边距宽度，工具提示窗口本身的宽度，以及工具提示的背景和文本颜色。一个工具提示控件可以为多个工具提供信息。

CToolTipCtrl类提供了Windows通用工具提示控件的功能。这个控件（也就是这个CToolTipCtrl类）只对运行在Windows 95和Windows NT 3.51或更高版本下的程序来说是可用的。

有关使能CToolTipCtrl的更多信息，参见“Visual C++程序员指南”中的“控件主题”和“使用CToolTipCtrl”。

#include <afxcmn.h>

请参阅　CtoolBar

CToolTipCtrl 类成员

Construction	
CtoolTipCtrl	创建一个 CToolTipCtrl 对象
Create	创建一个工具提示控件并将它与一个 CToolTipCtrl 对象连接

Attributes	
GetText	获取一个工具提示控件为一个工具维持的文本
GetToolInfo	获取一个工具提示控件维持的关于一个工具的信息
SetToolInfo	设置一个工具提示控件为一个工具维持的文本
GetToolCount	获取由一个工具提示控件支持的工具数
GetDelayTime	获取当前为一个工具提示控件设置的初始，弹出，和再显示持续时间
SetDelayTime	为一个工具提示控件设置初始，弹出，和再显示持续时间

续表

GetMargin	获取当前为一个工具提示窗口设置的上，左，底，和右边距
SetMargin	为一个工具提示窗口设置上，左，底，和右边距
GetMaxTipWidth	获取一个工具提示窗口的最大宽度
SetMaxTipWidth	设置一个工具提示窗口的最大宽度
GetTipBkColor	获取一个工具提示窗口中的背景颜色
SetTipBkColor	设置一个工具提示窗口中的背景颜色
GetTipTextColor	获取一个工具提示窗口中的文本颜色
SetTipTextColor	设置一个工具提示窗口中的文本颜色
Operations	
Activate	激活工具提示控件或使它成为不活动的
AddTool	向一个工具提示控件注册一个工具
DelTool	从工具提示控件中删除一个工具
HitTest	测试一个点，以确定它是否位于给定工具的边界矩形之内，如果是，返回关于这个工具的信息
RelayEvent	传递一个鼠标消息给工具提示控件处理
SetToolRect	为一个工具设置一个新的边界矩形
UpdateTipText	为一个工具设置工具提示文本
Update	强制当前工具被重画

续表

Pop	从视中删除一个被显示的工具提示窗口

成员函数

CToolTipCtrl::Activate

void Activate(BOOL *bActivate*);

参数

bActivate

指示是要激活该工具提示控件，还是要使它成为不活动的。

说明

此成员函数用来激活一个工具提示控件或使它成为不活动的。如果 bActivate 是 TRUE，则控件被激活；如果是 FALSE，则将控件变为不活动的。

当一个工具提示控件被激活时，当光标在一个向工具提示注册过的工具上时就会显示工具提示信息；当工具提示控件是不活动的时，则不会显示工具提示信息，即使是光标在一个工具上。

请参阅 CToolTipCtrl::UpdateTipText, CToolTipCtrl::SetDelayTime

CToolTipCtrl::AddTool

BOOL AddTool(CWnd* *pWnd*, UINT *nIDText*, LPCRECT *lpRectTool* = NULL,
    UINT *nIDTool* = 0);
BOOL AddTool(CWnd* *pWnd*, LPCTSTR *lpszText* = LPSTR_TEXTCALLBACK,
    LPCRECT *lpRectTool* = NULL, UINT *nIDTool* = 0);

返回值

如果成功则返回非零值；否则返回 0。

参数

pWnd
    指向包含此工具的窗口的指针。

nIDText
    包含工具的文本的字符串资源的 ID。

lpRectTool
    指向一个 RECT **结构的指针，该结构包含了工具的边界矩形的坐标。此坐标是相对于由** pWnd **指定的窗口的客户区的左上角的。**

nIDTool

该工具的 ID。

lpszText

指向工具的文本的指针。如果这个参数包含的值是 LPSTR_TEXTCALLBACK,则 TTN_NEEDTEXT 通知消息被发送给 *pWnd* 指向的窗口的父窗口。

说明

一个工具提示控件可以与多于一个的工具相关联。此成员函数用来向工具提示控件注册一个工具,这样当光标位于这个工具上面时,保存在工具提示中的信息就被显示。

请参阅 CToolTipCtrl::DelTool

CToolTipCtrl::Create

BOOL Create(CWnd* *pParentWnd*, DWORD *dwStyle* = 0);

返回值

如果 CToolTipCtrl 对象被成功创建,则返回非零值;否则返回零。

参数

pParentWnd

指定工具提示控件的父窗口，通常是一个 CDialog。它不能是 NULL。

dwStyle

指定工具提示控件的风格。使用该控件所需要的风格的任意组合。

说明

构造一个 CToolTipCtrl 对象分为两步。首先调用构造函数来构造 CToolTipCtrl 对象；然后调用 Create 来创建工具提示控件并将它与 CToolTipCtrl 对象连接。

dwStyle 参数可以是任意 Windows 风格的组合。另外，一个工具提示控件还具有两种特定类风格：TTS_ALWAYSTIP 和 TTS_NOPREFIX。

风格	含义
TTS_ALWAYSTIP	指示当光标在一个工具上时显示工具提示，不管工具提示的属主窗口是否是处于活动状态。没有这个风格，则只有当工具的属主窗口是活动的时候才会显示工具提示控件，否则不显示
TTS_NOPREFIX	这个风格禁止系统将&字符从一个字符串中去掉。如果一个工具提示控件没有 TTS_NOPREFIX 风格，则系统自动去掉&字符，让应用程序用同一个字符串作为菜单项和工具提示控件中的文本

一个工具提示控件总是具有 WS_POPUP 和 WS_EX_TOOLWINDOW 风格，不管在创建它们的时候你是否指定。

请参阅 CToolTipCtrl::CToolTipCtrl

CToolTipCtrl::CToolTipCtrl

CToolTipCtrl( );

说明

此成员函数用来构造一个 CToolTipCtrl 对象。在构造了对象之后你必须调用

Create 函数。

请参阅 CToolTipCtrl::Create

CToolTipCtrl::DelTool

void DelTool(CWnd* *pWnd*, UINT *nIDTool* = 0);

参数

pWnd

指向包含工具的窗口的指针。

nIDTool

工具的 ID。

说明

此成员函数用来从一个工具提示控件支持的工具集合中删除由 *pWnd* 和 *nIDTool* 指定的工具。

请参阅 CToolTipCtrl::AddTool

CToolTipCtrl::GetDelayTime

int GetDelayTime(DWORD *dwDuration*) const;

返回值

返回用毫秒表示的指定延迟时间。

参数

dwDuration
用来指定将要获取哪一个时间持续值的标志。这个参数可以是下列值之一：

- TTDT_AUTOPOP **获取当指针固定在一个工具的边界矩形内时工具提示窗口保持持续可见的时间长度。**
- TTDT_INITIAL **获取在工具提示窗口显示之前指针必须固定在一个工具的边界矩形内的时间长度。**
- TTDT_RESHOW **获取当指针从一个工具移动到另一个工具时，后续工具提示窗口显示所需要的时间长度。**

说明

此成员函数用来实现 Win32 **消息** TTM_GETDELAYTIME **的行为，就像在**

“Platform SDK”中描述的一样。

请参阅 CToolTipCtrl::SetMargin

CToolTipCtrl::GetMargin

void GetMargin(LPRECT *lprc*) const;

参数

lprc

是一个将用来获取边距信息的 RECT 结构的地址。该 RECT 结构的成员没有定义一个边界矩形。为了实现这个消息的目的，将结构成员解释如下：

Member	Representation
Top	顶边线与工具提示文本顶部之间的以像素表示的距离
Left	左边线与提示文本最左边之间的以像素表示的距离
bottom	底边线与提示文本底部之间的以像素表示的距离
right	右边线与提示文本最右边之间的以像素表示的距离

说明

此成员函数用来实现 Win32 消息 TTM_GETMARGIN 的行为，就像在“Platform

SDK”中描述的一样。

请参阅　CToolTipCtrl::SetMargn

CToolTipCtrl::GetMaxTipWidth

int GetMaxTipWidth( ) const;

返回值

返回一个工具提示窗口的最大宽度。

说明

此成员函数用来实现 Win32 消息 TTM_GETMAXTIPWIDTH 的行为，就像在“Platform SDK”中描述的一样。

请参阅　CToolTipCtrl::SetMaxTipWidth

CToolTipCtrl::GetText

void GetText(CString& *str*, CWnd* *pWnd*, UINT *nIDTool* = 0) const;

参数

str

是对一个用来接收工具的文本的 CString 对象的引用。

pWnd

指向包含工具的窗口的指针。

nIDTool

工具的 ID。

说明

此成员函数用来获取工具提示控件为一个工具保持的文本。*pWnd* 和 *nIDTool* 参数标识工具。如果该工具先前通过调用 CToolTipCtrl::AddTool 向工具提示控件注册过，则由 *str* 参数引用的对象被赋给工具的文本。

请参阅 CToolTipCtrl::AddTool, CToolTipCtrl::DelTool

CToolTipCtrl::GetTipBkColor

COLORREF GetTipBkColor( ) const;

返回值

返回一个代表背景颜色的 COLORREF **值。**

说明

此成员函数用来实现 Win32 **消息** TTM_GETTIPBKCOLOR **的行为，就像在**"Platform SDK"**中描述的一样。**

请参阅 CToolTipCtrl::SetTipBkColor

CToolTipCtrl::GetTipTextColor

```
COLORREF GetTipTextColor( ) const;
```

返回值

返回一个代表文本颜色的 COLORREF **值。**

说明

此成员函数用来实现 Win32 **消息** TTM_GETTIPTEXTCOLOR **的行为，就像在**"Platform SDK"**中描述的一样。**

请参阅 CToolTipCtrl::SetTipTextColot

CToolTipCtrl::GetToolCount

```
int GetToolCount( ) const;
```

返回值

返回向工具提示控件注册了的工具数目。

说明

此成员函数用来获取向工具提示控件注册的工具数目。
请参阅 CToolTipCtrl::AddTool, CToolTipCtrl::DelTool

CToolTipCtrl::GetToolInfo

BOOL GetToolInfo(CToolInfo& *CToolInfo*, CWnd* *pWnd*, UINT *nIDTool* = 0) const;

返回值

如果成功则返回非零值；否则返回 0。

参数

CToolInfo

是对一个用来接收工具的文本的 TOOLINFO 对象的引用。

pWnd

指向包含工具的窗口的指针。

nIDTool

工具的 ID。

说明

此成员函数用来获取一个工具提示控件保持的一个工具的信息。由 *CToolInfo* 引用的 TOOLINFO 结构中的 hwnd 和 uId 成员标识了该工具。如果该工具先前已经通过调用 AddTool 向工具提示控件注册了，则 TOOLLINFO 结构被用该工具的信息填充。

请参阅 CToolTipCtrl::AddTool

CToolTipCtrl::HitTest

BOOL HitTest(CWnd* *pWnd*, CPoint *pt*, LPTOOLINFO *lpToolInfo*) const;

返回值

如果由点击-测试信息指定的点在工具的边界矩形之内，则返回非零值；否则返回 0。

参数

pWnd
    指向包含工具的窗口的指针。

pt
    指向一个用来包含要被测试的点的坐标的 CPoint 对象的指针。

lpToolInfo
    指向包含工具信息的 TOOLINFO 结构的指针。

说明

此成员函数用来测试一个点，以确定这个点是否位于给定工具的边界矩形之内，如果是，则获取有关该工具的信息。

如果这个函数返回一个非零值，则用该点所在的工具的信息来填充 *lpToolInfo* 指向的结构。

TTHITTESTINFO 结构按如下定义：

```
typedef struct _TT_HITTESTINFO { // tthti
```

```
    HWND hwnd;      // handle of tool or window with tool
    POINT pt;       // client coordinates of point to test
    TOOLINFO ti; // receives information about the tool
} TTHITTESTINFO, FAR * LPHITTESTINFO;
```

hwnd

指定该工具的句柄。

pt

如果一个点在工具的边界矩形内，则指定这个点的坐标。

ti

有关该工具的信息。有关 TOOLINFO **结构的更多信息，可以参见** CToolTipCtrl::Get- ToolInfo。

请参阅 CToolTipCtrl::GetToolInfo

CToolTipCtrl::Pop

```
void pop( );
```

说明

此成员函数用来将一个被显示的工具提示窗口从视中删除。

此成员函数用来实现 Win32 **消息** TTM_POP **的行为，就像在**“Platform SDK”

中描述的一样。

CToolTipCtrl::RelayEvent

void RelayEvent(LPMSG *lpMsg*);

参数

lpMsg
指向一个包含要转播的消息的 MSG 结构的指针。

说明

此成员函数用来将一个鼠标消息传递给一个工具提示控件处理。一个工具提示只处理下面的消息，这些消息是通过 RelayEvent 传递给它的：

WM_LBUTTONDOWN	WM_MOUSEMOVE
WM_LBUTTONUP	WM_RBUTTONDOWN
WM_MBUTTONDOWN	WM_RBUTTONUP
WM_MBUTTONUP	

请参阅 CWnd::PreTranslateMessage, CWinApp::PreTranslateMessage

CToolTipCtrl::SetDelayTime

void SetDelayTime(UINT *nDelay*);
void SetDelayTime(DWORD *dwDuration*, int *iTime*);

参数

nDelay
　　指定以毫秒表示的新的延迟时间。

dwDuration
　　指定要获取哪一段持续时间值的标志。参见 CToolTipCtrl::GetDelayTime **可以获得其有效值的描述。**

iTime
　　以毫秒表示的指定延迟时间。

说明

此成员函数用来为一个工具提示控件设置延迟时间。延迟时间是在工具提示窗口显示之前光标必须保持在一个工具上的时间长度。缺省的延迟时间是 500 **毫秒。**

请参阅 CToolTipCtrl::Activate, CToolTipCtrl::HitTest
CtoolTipCtrl::GetDelayTime,TTM_SETDELAYTIME

CToolTipCtrl::SetMargin

void SetMargin(LPRECT lprc);

参数

lprc

　　一个包含要被设置的边距的信息的 RECT **结构的地址。该** RECT **结构的成员没有定义一个边界矩形。参见** CToolTipCtrl::GetMargin **可以获得有关边界信息的描述。**

说明

此成员函数用来实现 Win32 **消息** TTM_SETMARGIN **的行为，就像在“** Platform SDK **”中描述的一样。**

请参阅　CToolTipCtrl::GetMargin

CToolTipCtrl::SetMaxTipWidth

int SetMaxTipWidth(int *iWidth*)

返回值

返回先前的最大提示宽度。

参数

iWidth
要设置的最大工具提示窗口的宽度。

说明

此成员函数用来实现 Win32 消息 TTM_SETMAXTIPWIDTH 的行为，就像在“Platform SDK”中描述的一样。

请参阅 CToolTipCtrl::GetMaxTipWidth

CToolTipCtrl::SetTipBkColor

```
void SetTipBkColor ( COLORREF clr );
```

参数

clr
新的背景颜色。

说明

此成员函数用来实现 Win32 消息 TTM_SETTIPBKCOLOR 的行为，就像在“Platform SDK”中描述的一样。

请参阅 CToolTipCtrl::GetTipBkColor

CToolTipCtrl::SetTipTextColor

void SetTipTextColor(COLORREF *clr*);

参数

clr

新的文本颜色。

说明

此成员函数用来实现 Win32 消息 TTM_SETTIPTEXTCOLOR 的行为，就像在“Platform SDK”中描述的一样。

请参阅 CToolTipCtrl::GetTipTextColor

CToolTipCtrl::SetToolInfo

void SetToolInfo(LPTOOLINFO *lpToolInfo*);

参数

lpToolInfo
　　指向一个用来指定要设置的信息的 TOOLINFO **结构的指针。**

说明

此成员函数用来设置一个工具提示控件为一个工具保持的信息。

请参阅　CToolTipCtrl::GetToolInfo

CToolTipCtrl::SetToolRect

void SetToolRect(CWnd* *pWnd*, UINT *nIDTool*, LPCRECT *lpRect*);

参数

pWnd
　　指向包含工具的窗口的指针。

nIDTool

工具的 ID。

lpRect

指向一个 RECT 结构的指针，该结构指定了新的边界矩形。

说明

此成员函数用来为一个工具设置新的边界矩形。

请参阅 CToolTipCtrl::GetToolInfo

CToolTipCtrl::Update

void Update( );

说明

此成员函数用来强制当前工具被重画。

请参阅 CToolTipCtrl::UpdateTipText

CToolTipCtrl::UpdateTipText

void UpdateTipText(LPCTSTR *lpszText*, CWnd* *pWnd*, UINT *nIDTool* = 0);
void UpdateTipText(UINT *nIDText*, CWnd* *pWnd*, UINT *nIDTool* = 0);

参数

lpszText
指向工具的文本的指针。

pWnd
指向包含此工具的窗口的指针。

nIDTool
工具的 ID。

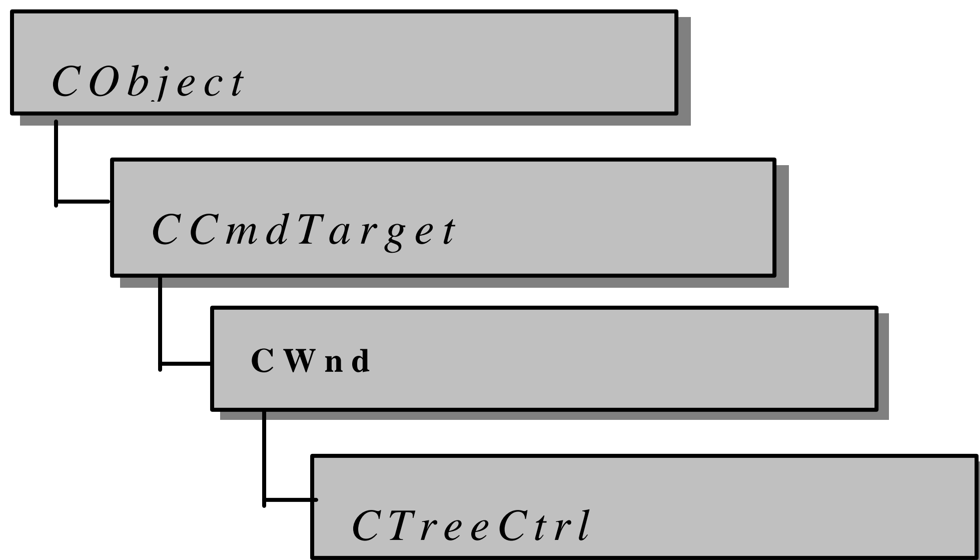
nIDText
包含工具文本的字符串资源的 ID。

说明

此成员函数用来为此控件的工具更新工具提示文本。

请参阅 CToolTipCtrl::GetToolInfo

CTreeCtrl



一个“tree view 控件”是一个用来显示项的层次列表的窗口，比如一个文档中的标题，索引中的项，或磁盘中的文件和目录。每一个项都包括一个标签和一个可选的位图图像，每一个项还有一个与其相关的子项的列表。单击一个项，用户可以展开或缩进该项的相关子项的列表。

CTreeCtrl 类提供了 Windows 通用 tree view 控件的性能。这个控件（也就是

CTreeCtrl类）只对运行在Windows 95和Windows NT 3.51或更高版本下的程序来说是可用的。

有关使用CTreeCtrl的更多信息，参见“Visual C++程序员指南”中的“控件主题”和“使用CTreeCtrl”。

#include <afxcmn.h>

请参阅 CImageList

CTreeCtrl类成员

Construction	
CTreeCtrl	构造一个CTreeCtrl对象
Create	创建一个tree view控件并将它与一个CTreeCtrl对象连接

Attributes	
GetCount	获取与一个 treeview 控件相关联的 tree 项的数目
GetIndent	获取一个 tree view 项对它的父项的偏移（以像素表示）
SetIndent	设置一个 tree view 项对它的父项的偏移（以像素表示）
GetImageList	获取与一个 tree view 控件相关联的图像列表的句柄
SetImageList	设置与一个 tree view 控件相关联的图像列表的句柄
GetNextItem	获取与指定的关系匹配的下一个 tree view 项
ItemHasChildren	如果指定项有子项则返回非零值
GetChildItem	获取一个指定 tree view 项的子项
GetNextSiblingItem	获取指定 tree view 项的下一个兄弟项
GetPrevSiblingItem	获取指定 tree view 项的前一个兄弟项
GetParentItem	获取指定 tree view 项的父项
GetFirstVisibleItem	获取指定 tree view 项的第一个可视项
GetNextVisibleItem	获取指定 tree view 项的下一个可视项
GetPrevVisibleItem	获取指定 tree view 项的前一个可视项
GetSelectedItem	获取当前被选择的 tree view 项

续表

GetDropHilightItem	获取一次拖放操作的目标
GetRootItem	获取指定 tree view 项的根
GetItem	获取一个指定 tree view 项的属性
SetItem	设置一个指定 tree view 项的属性
GetItemState	返回一个项的状态
SetItemState	设置一个项的状态
GetItemImage	获取与一个项相关联的图像
SetItemImage	设置与一个项相关联的图像
GetItemText	返回一个项的文本
SetItemText	设置一个项的文本
GetItemData	返回与一个项关联的 32 位的应用程序指定值
SetItemData	设置与一个项关联的 32 位的应用程序指定值
GetItemRect	获取一个 tree view 项的边界矩形
GetEditControl	获取用来编辑指定 tree view 项的编辑控件的句柄
GetVisibleCount	获取与一个 tree view 项关联的可视 tree 项的编号
GetToolTips	获取一个 tree view 控件使用的子 ToolTip 控件的的句柄
SetToolTips	设置一个 tree view 控件的子 ToolTip 控件的的句柄
GetBkColor	获取控件的当前背景颜色
SetBkColor	设置控件的背景颜色

续表

GetItemHeight	获取 tree view 项的当前高度
SetItemHeight	设置 tree view 项的当前高度
GetTextColor	获取控件的当前文本颜色
SetTextColor	设置控件的文本颜色
SetInsertMark	设置一个 tree view 控件的插入标记
GetCheck	获取一个 tree 控件项的核选状态
SetCheck	设置一个 tree 控件项的核选状态
GetInsertMarkColor	获取 tree view 用来绘制插入标记的颜色
SetInsertMarkColor	设置 tree view 用来绘制插入标记的颜色

Operations

InsertItem	在一个 tree view 控件中插入一个新项。
DeleteItem	从一个 tree view 控件中删除一个项
DeleteAllItems	从一个 tree view 控件中删除所有的项
Expand	展开或收缩指定 tree view 项的子项
Select	选择，在视中滚动，或重画一个指定的 tree view 项
SelectItem	选择一个指定的 tree view 项
SelectDropTarget	重画作为一次拖放操作的目标的 tree 项
SelectSetFirstVisible	选择一个指定的 tree view 项作为第一个可视项

续表

EditLabel	现场编辑一个指定的 tree view 项
HitTest	返回与 CtreeCtrl 关联的光标的当前位置
CreateDragImage	为指定的 tree view 项创建一个拖动位图
SortChildren	排序一个给定父项的子项
EnsureVisible	确保一个 tree view 项在它的 tree view 控件中是可视的
SortChildrenCB	使用一个由应用程序定义的排序函数来排列一个给定父项的子项

成员函数

CTreeCtrl::Create

BOOL Create(DWORD *dwStyle*, const RECT& *rect*, CWnd* *pParentWnd*, UINT *nID*);

返回值

如果初始化成功则返回非零值；否则返回 0。

参数

dwStyle

指定 tree view **控件的风格。可以对这个控件使用** tree view **控件风格的任意组合。**

rect

指定 tree view **控件的尺寸和位置。此参数可以是一个** CRect **对象或一个** RECT **结构。**

pParentWnd

指定 tree view **控件的父窗口，通常是一个** CDialog**。它不能是** NULL**。**

nID

指定 tree view **控件的** ID**。**

说明

构造一个 CTreeCtrl **要分两步。首先调用构造函数，然后调用** Create **来创建这个** tree view **控件并将它与该** CTeeCtrl **对象连接。**

下面的风格可以应用到一个 tree view **控件：**

- TVS_HASLINES tree view **控件的子项与它们的父项之间用线连接。**
- TVS_LINESATROOT tree view **控件用线连接子项和根项。**
- TVS_HASBUTTONS tree view **在每一个父项的左边添加一个按钮。**

- TVS_EDITLABELS tree view 控件允许用户编辑 tree view 项的标签。
- TVS_SHOWSELALWAYS 当 tree view 失去焦点时，使被选择的项仍然保持被选择。
- TVS_DISABLEDRAGDROP 该 tree view 控件被禁止发送。
- TVN_BEGINDRAG 通知消息。
- TVS_NOTOOLTIPS tree view 控件使用工具提示。
- TVS_SINGLEEXPAND 当使用这个风格时，改变在 tree view 中的选择将导致正被选择的项展开，而没有被选择的项收缩。如果用鼠标单击被选择的项，并且该项是关闭的，则该项就会展开。如果该被选择的项被单击时是打开的，则它就会收缩。

请参阅 CTreeCtrl:: CTreeCtrl, Tree View Control Window Styles in the Paltform SDK

CTreeCtrl::CreateDragImage

CImageList* CreateDragImage(HTREEITEM *hItem*);

返回值

如果成功，则返回指向图像列表的指针，拖动位图将被加入到其中去；否则返回 NULL。

参数

hItem

　　要被拖动的 tree 项的句柄。

说明

此成员函数用来为一个 tree view 控件中的给定项创建一个拖动位图，为位图创建一个图像列表，并将位图添加到该图像列表中。当项被拖动时，应用程序用图像-列表函数来显示图像。

CImageList **对象是持久的，当使用完它时你必须删除它。例如：**

```
CImageList * pImageList = MyTreeCtrl.CreateDragImage( nItem, &point );
    …
    …
    delete pImageList;
```

请参阅　CTreeCtrl::SelectDropTarget, CTreeCtrl::GetDropHilightItem, CTreeCtrl::SetImageList

CTreeCtrl::CTreeCtrl

CTreeCtrl( ) ;

说明

此函数用来构造一个 CTreeCtrl 对象。

请参阅 CTreeCtrl::Create

CTreeCtrl::DeleteAllItems

BOOL DeleteAllItems( );

返回值

如果成功则返回非零值；否则返回 0。

说明

此成员函数用来从 tree view 控件中删除所有的项。

请参阅 CTreeCtrl::DeleteItem, CTreeCtrl::InsertItem

CTreeCtrl::DeleteItem

BOOL DeleteItem(HTREEITEM *hItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

要被删除的 tree view 项的句柄。如果 *hItem* 的值是 TVI_ROOT，则所有的项都被从此 tree view 控件中删除。

说明

此成员函数用来从 tree view 控件中删除一个项。

请参阅 CTreeCtrl::DeleteAllItems, CTreeCtrl::InsertItem

CTreeCtrl::EditLabel

```
CEdir* EditLabel( HTREEITEM hItem );
```

返回值

如果成功则返回一个指向 CEdit 对象的指针，该对象被用来编辑该项的文本；否则返回 NULL。

参数

hItem

要被编辑的 tree 项的句柄。

说明

此成员函数开始现场编辑指定项的文本。通过用一个单行编辑控件所包含的文本替换该项的文本来完成编辑。

请参阅 CTreeCtrl::GetEditControl

CTreeCtrl::EnsureVisible

BOOL EnsureVisible (HTREEITEM *hItem*);

返回值

如果系统滚动 tree view 控件中的项以保证指定项可见则返回 TRUE。否则，返回值为 FALSE。

参数

hItem

要被变为可见的 tree 项的句柄。

说明

此成员函数用来保证一个 tree view 项是可见的。如果必要，函数扩展父项或滚动 tree view 扩展以使该项可见。

请参阅 CTreeCtrl::GetFirstVisibleItem, CTreeCtrl::GetVisibleCount

CTreeCtrl::Expand

BOOL Expand(HTREEEETEM *hItem*, UINT *nColor*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

要被扩展的 tree 项的句柄。

nCode

用来指示要被进行的动作的标志。这个标志可以是下列值之一：

- TVE_COLLAPSE 收缩列表。
- TVE_COLLAPSERESET 收缩列表并删除子项。
- TVE_EXPAND 展开列表。
- TVE_TOGGLE 如果列表当前是展开的则收缩列表；反之则展开列表。

说明

此成员函数用来展开或收缩给定父项的子项列表（如果有）。

请参阅 CTreeCtrl::EnsureVisible

CTreeCtrl::GetBkColor

COLORREF GetBkColor( ) const;

返回值

返回一个代表当前背景颜色的 COLORREF 值。如果这个值是-1，则控件正使用系统的颜色作为背景色。

说明

此成员函数用来实现 Win32 消息 TVM_GETBKCOLOR 的行为，就像在“PlatformSDK”中描述的一样。

请参阅 CTreeCtrl::SetBkColor

CTreeCtrl::GetCheck

BOOL GetCheck(HTREEITEM *hItem*) const;

返回值

如果该 tree 控件项是被核选的，则返回非零值；否则返回 0。

参数

hItem
要获取有关其状态信息的 HTREEITEM。

说明

此成员函数用来获取一个项的核选状态。

请参阅 CTreeCtrl::SetCheck

CTreeCtrl::GetChileItem

HTREEITEM GetChildItem(HTREEITEM *hItem*);

返回值

如果成功则返回该子项的句柄；否则返回 NULL。

参数

hItem
一个 tree **项的句柄。**

说明

此成员函数用来获取由 *hItem* **指定的项的子项。**

请参阅 CTreeCtrl::GetItem, CTreeCtrl::GetParentItem, CTreeCtrl::SortChildren

CTreeCtrl::GetCount

UINT GetCount( )

返回值

返回此 tree view **控件中的项的数目；否则返回**-1。

说明

此成员函数用来获取一个 tree view 控件中的项的数目。

请参阅 CTreeCtrl::GetVisibleCount

CTreeCtrl::GetDropHilightItem

HTREEITEM GetDropHilightItem( );

返回值

如果成功则返回项的句柄；否则返回 NULL。

说明

此成员函数用来获取一次拖放操作的目标。

请参阅 CTreeCtrl::SelectDropTarget

CTreeCtrl::GetEditControl

CEdit* GetEditControl( )

返回值

如果成功则返回一个指向用来编辑项的文本的编辑控件的指针；否则返回NULL。

说明

此成员函数用来获取被用来编辑一个 tree view **项的文本的编辑控件的句柄。**

请参阅 CTreeCtrl::EditLabel

CTreeCtrl::GetFirstVisibleItem

```
HTREEITEM GetFirstVisibleItem( );
```

返回值

如果成功则返回第一个可视项的句柄；否则返回 NULL。

说明

此成员函数用来获取该 tree view **控件中的第一个可视项的句柄。**

请参阅 CTreeCtrl::GetNextVisibleItem, CTreeCtrl::GetPrivVisibleItem, CTreeCtrl::EnsureVisible, CTreeCtrl::GetVisibleCount

CTreeCtrl::GetImageList

CImageList* GetImageList(UINT *nImage*)

返回值

如果成功则返回指向控件的图像列表的指针；否则返回 NULL。

参数

nImage

要获取的图像列表的类型。该图像列表可以是下列类型之一：

- TVSIL_NORMAL 获取常规的图像列表，它包含了该 tree view 项的被选择的和不被选择的图像。
- TVSIL_STATE 获取状态图像列表,它包含了处于用户定义状态的 tree view 项的图像。

说明

此成员函数用来获取与该 tree view 控件关联的常规或状态图像列表的句柄。一个 tree view 控件中的每一个项都有一对图像与之关联。当项被选择时就显示其中某一个图像，当项没有被选择时则显示另一个图像。例如，当一个项被选择时显示为一个打开的文件夹，而当它没有被选择时则显示为一个关闭的文件

夹。

有关图像列表的更多信息，参见 CImageList 类。

请参阅 CImageList, CTreeCtrl::SetImageList

CTreeCtrl::GetIndent

```
UINT GetIndent( );
```

返回值

返回以像素表示的缩进量。

说明

此成员函数用来获取子项相对于父项缩进的量。

请参阅 CTreeCtrl::SetIndent

CTreeCtrl::GetInsertMarkColor

```
COLORREF GetInsertMarkColor( ) const;
```

返回值

返回一个包含当前插入编辑颜色的 COLORREF。

说明

此成员函数用来实现 Win32 **消息** TVM_GETINSERTMARKCOLOR **的行为，就像在**“PlatformSDK”**中描述的一样。**

请参阅 CTreeCtrl::SetInsertMarkColor

CTreeCtrl::GetItem

BOOL GetItem(TVITEM* *pItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

pItem

是一个指向 TVITEM **结构的指针，就像在**“Platfor SDK”**中描述的一样。**

说明

此成员函数用来获取指定 tree view **项的属性。**

请参阅 CTreeCtrl::SetItem, CTreeCtrl::GetChildItem, CTreeCtrl::GetNextItem, CTreeCtrl::SelectItem

CTreeCtrl::GetItemData

DWORD GetItemData(HTREEITEM *hItem*) const;

返回值

返回一个与由 *hItem* **指定的项关联的** 32 **位的应用程序指定值。**

参数

hItem

要获取其数据的项的句柄。

说明

此成员函数用来获取与指定项关联 32 **位的应用程序指定值。**

请参阅 CTreeCtrl::SetItemData

CTreeCtrl::GetItemHeight

SHORT GetItemHeight( ) const;

返回值

返回以像素表示的项的高度。

说明

此成员函数用来实现 Win32 **消息** TVM_GETITEMHEIGHT **的行为，就像在**“PlatformSDK”**中描述的一样。**

请参阅 CTreeCtrl::SetItemHeight

CTreeCtrl::GetItemImage

BOOL GetItemImage(HTREEITEM *hItem*, int& *nImage*, int& *nSelectedImage*) const;

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

要获取其图像的项的句柄。

nImage

一个用来接收该 tree view 控件的图像列表中的该项图像的索引的整数。

nSelectedImage

一个用来接收该 tree view 控件的图像列表中的该项的被选择图像的索引的整数。

说明

在一个 tree view 控件中每一个项有一对与之关联的位图图像。图像显示在一个项的标签的左边。其中一个项是在项被选择的时候显示的，另一个项是在项没有被选择的时候显示的。例如，当一个项被选择时它可能被显示为一个打开的文件夹，而当它没有被选择时则显示为一个关闭的文件夹。

此成员函数用来接收项的图像和被选择图像在该 tree view 控件的图像列表中的索引。

请参阅 CTreeCtrl::SetItemImage, CImageList

CTreeCtrl::GetItemRect

BOOL GetItemRect(HTREEITEM *hItem*, LPRECT *lpRect*, BOOL *bTextOnly*);

返回值

如果项是可视的则返回非零值，以及包含在 *lpRect* **中的边界矩形。否则，返回 0 和没有被初始化的** *lpRect*。

参数

hItem

一个 tree view **项的句柄。**

lpRect

指向一个用来接收边界矩形的 RECT **结构的指针。其中的坐标是相对于该** tree view **控件的左上角的。**

bTextOnly

如果这个参数是非零值，则边界矩形值包括项的文本。否则，它包括该项在 tree view **控件所占据的整个一行。**

说明

此成员函数用来获取 *hItem* **的边界矩形，并确定该项是否是可视的。**

请参阅 CTreeCtrl::GetVisibleCount, CTreeCtrl::GetNextVisibleItem, CTreeCtrl::GetPrevVisibleItem, CTreeCtrl::EnsureVisible

CTreeCtrl::GetItemState

UINT GetItemState(HTREEITEM *hItem*, UINT *nStateMask*) const;

返回值

返回一个用来指定项的状态的 UINT。**可能取值的信息，参见** CTreeCtrl::GetItem。

参数

hItem
要获取其状态的项的句柄。

nStateMask
用来指定要获取哪些状态的掩码，有关 *nStateMask* **的可能取值的更多信息，参见**“Platform SDK”**中的有关** TVITEM **结构的** state **和** stateMask **成员的讨论。**

说明

此成员函数返回由 *hItem* **指定的项的状态。**

请参阅 CTreeCtrl::GetItem

CTreeCtrl::GetItemText

CString GetItemText(HTREEITEM *hItem*) const;

返回值

返回一个包含该项的文本的 CString **对象。**

参数

hItem

要获取其文本的项的句柄。

说明

此成员函数返回由 *hItem* **指定的项的文本。**

请参阅 CTreeCtrl::SetItemText

CTreeCtrl::GetNextItem

HTREEITEM GetNextItem(HTREEITEM *hItem*, UINT *nCode*);

返回值

如果成功则返回下一个项的句柄；否则返回 NULL。

参数

hItem

一个 tree **项的句柄。**

nCode

一个用来指示与 *hItem* **的关系的类型的标志。这个标志可以是下列值之一：**

- TVGN_CARET **获取当前被选择的项。**
- TVGN_CHILD **获取第一个子项。***hItem* **参数必须是** NULL。
- TVGN_DROPHILITE **获取是一次拖放操作的目标的项。**
- TVGN_FIRSTVISIBLE **获取第一个可见的项。**
- TVGN_TEXT **获取下一个兄弟项。**
- TVGN_NEXTVISIBLE **获取跟随在指定项之后的下一个可视项。**
- TVGN_PARENT **获取指定项的父项。**
- TVGN_PREVIOUS **获取前一个兄弟项。**

- TVGN_PREVIOUSVISIBLE 获取在指定项之前的第一个可视项。
- TVGN_ROOT 获取根项的第一个子项，指定项是该根项的一个部分。

说明

此成员函数用来获取与 *nItem* 具有由 *nCode* 参数指定的关系的 tree view 项。

请参阅 CTreeCtrl::SetItem, CTreeCtrl::GetChildItem, CTreeCtrl::Getitem, CTreeCtrl::SelectItem, CTreeCtrl::GetPrevSiblingItem

CTreeCtrl::GetNextSiblingItem

HTREEITEM GgeNextSiblingItem) HTREEITM *hItem*);

返回值

返回下一个兄弟项的句柄；否则返回 NULL。

参数

hItem

一个 tree 项的句柄。

说明

此成员函数用来获取下一个兄弟项。

请参阅 CTreeCtrl::GetPrevSiblingItem, CTreeCtrl::GetChildItem, CTreeCtrl::GetItem, CTreeCtrl::SelectItem, CTreeCtrl::GetParentItem

CTreeCtrl::GetNextVisibleItem

HTREEITEM GetNextVisibleItem(HTREEITEM *hItem*);

返回值

返回下一个可视项的句柄；否则返回 NULL。

参数

hItem
一个 tree 项的句柄。

说明

此成员函数用来获取 *hItem* 的下一个可视项。

请参阅 CTreeCtrl::GetPrevVisibleItem, CTreeCtrl::GetFirstVisibleItem, CTreeCtrl::EnsureVisible, CTreeCtrl::GetParentItem

CTreeCtrl::GetParentItem

HTREEITEM GetParentItem(HTREEITEM *hItem*);

返回值

返回父项的句柄；否则返回 NULL。

参数

hItem
一个 tree **项的句柄。**

说明

此成员函数用来获取 *hItem* **的父项。**

请参阅 CTreeCtrl::GetChildItem, CTreeCtrl::GetRoolItem, CTreeCtrl::GetItem, CTreeCtrl::GetPrevSiblingItem

CTreeCtrl::GetPrevSiblingItem

HTREEITEM GetPrevSiblingItem(HTREEITEM *hItem*);

返回值

返回前一个兄弟项的句柄；否则返回NULL。

参数

hItem
一个tree项的句柄。

说明

此成员函数用来获取*hItem*的前一个兄弟项。
请参阅CTreeCtrl::GetNextSiblingItem, CTreeCtrl::GetParentItem, CTreeCtrl::GetChildItem

CTreeCtrl::GetPrevVisibleItem

HTREEITEM GetPrevVisibleItem(HTREEITEM *hItem*);

返回值

返回前一个可视项的句柄；否则返回NULL。

参数

hItem
　　一个 tree 项的句柄。

说明

此成员函数用来获取 *hItem* 的前一个可视项。

请参阅　CTreeCtrl::GetNextVisibleItem, CTreeCtrl::GetFirstVisibleItem, CTreeCtrl::EnsureVisible, CTreeCtrl::GetVisibleCount

CTreeCtrl::GetRootItem

HTREEITEM GetRootItem( );

返回值

返回根项的句柄；否则返回 NULL。

说明

此成员函数用来获取 *hItem* 的根项。

请参阅　CTreeCtrl::GetItem, CTreeCtrl::GetChildItem, CTreeCtrl::GetParentItem

CTreeCtrl::GetSelectedItem

HTREEITEM GetSelectedItem( );

返回值

返回被选择项的句柄；否则返回 NULL。

说明

此成员函数用来获取此 tree view **控件中的当前被选择项。**

请参阅 CTreeCtrl::Select, CTreeCtrl::SelectDropTarget, CTreeCtrl::GetDropHilightItem

CTreeCtrl::GetTextColor

COLORREF GetTextColor( ) const;

返回值

返回一个代表当前文本颜色的 COLORREF **值。如果这个值是**-1，**则控件是使用系统颜色作为文本颜色。**

说明

此成员函数用来实现 Win32 消息 TVM_GETTEXTCOLOR 的行为，就像在“PlatformSDK”中描述的一样。

请参阅 CTreeCtrl::SetTextColor

CTreeCtrl::GetToolTips

```
CToolTipCtrl* GetToolTips( );
```

返回值

返回一个指向被 tree 控件使用的 CToolTipCtrl 对象的指针。如果 Create 成员函数使用了 TVS_NOTOOLTIPS 风格，则不会有工具提示控件被使用，此时返回值为 NULL。

说明

此成员函数用来实现 Win32 消息 TVM_GETTOOLTIPS 的行为，就像在“PlatformSDK”中描述的一样。

GetToolTips 的 MFC 实现返回一个被 tree 控件使用的 CToolTipCtrl 对象，而不是返回指向一个工具提示控件的句柄。

请参阅 CTreeCtrl::SetToolTips

CTreeCtrl::GetVisibleCount

UINT GetVisibleCount( );

返回值

返回 tree view 控件中的可见项的数目；否则返回-1。

说明

此成员函数用来获取一个 tree view 控件中的可视项的数目。

请参阅 CTreeCtrl::GetCount, CTreeCtrl::EnsureVisible

CTreeCtrl::HitTest

HTREEITEM HitTest(CPoint *pt*, UINT * *pFlags*);
HTREEITEM HitTest(TVHITTESTINFO* *pHitTestInfo*);

返回值

返回位于指定点的 tree view 项的句柄，如果没有项位于该点，则返回 NULL。

参数

pt

要测试的点的客户坐标。

pFlags

指向一个用来接收有关点击测试的信息的整数的指针。它可以是说明部分中列出的 flags 成员值中的一个或多个。

pHitTestInfo

一个包含点击测试的位置并接收测试结果的信息的 TVHITTESTINFO 结构的地址。

说明

此成员函数用来确定相对于一个 tree view 控件的客户区的指定点的定位。

当调用这个函数时，*pt* 参数指定要测试的点的坐标。此函数返回位于指定点的项的句柄，或者如果没有项位于该点则返回 NULL。另外，*pFlags* 参数包含了指明指定点的定位的值。

请参阅 CTreeCtrl::GetItemRect

CTreeCtrl::InsertItem

HTREEITEM InsertItem(LPTVINSERTSTRUCT *lpInsertStruct*);
HTREEITEM InsertItem(UINT *nMask*, LPCTSTR *lpszItem*,int *nImage*, int *nSelectedImage*,
UINT *nState*, UINT *nStateMask*, LPARAM *lParam*, HTREEITEM *hParent*,
HTREEITEM *hInsertAfter*);
HTREEITEM InsertItem(LPCTSTR *lpszItem*, HTREEITEM *hParent* = TVI_ROOT,
HTREEITEM *hInsertAfter* = TVI_LAST);
HTREEITEM InsertItem(LPCTSTR *lpszItem*, int *nImage*, int *nSelectedImage*,
HTREEITEM *hParent* = TVI_ROOT, HTREEITEM *hInsertAfter* = TVI_LAST);

返回值

如果成功则返回新项的句柄；否则返回 NULL。

参数

lpInsertStruct

一个指向用来指定要插入的 tree view **项的属性的** TVINSERTSTRUCT **的指针。**

nMask

用来指定要设置的属性的整数。

lpszItem

一个包含项的文本的字符串的地址。

nImage

项的图像在 tree view 控件的图像列表中的索引。

nSelectedImage

项的被选择图像在 tree view 控件的图像列表中的索引。

nState

为项的状态指定的值。

nStateMask

指定要设置的状态。

lParam

与此项关联的一个 32 位的应用程序指定的值。

hParent

要被插入的项的父项的句柄。

hInsertAfter

新项要被插入其后的项的句柄。

说明

此成员函数用来在一个 tree view 控件中插入一个新项。

请参阅 CTreeCtrl::DeleteItem, CTreeCtrl::HitTest, CTreeCtrl::SelectDropTarget, CTreeCtrl::GetItem

CTreeCtrl::ItemHasChildren

BOOL ItemHasChildren(HTREEITEM *hItem*);

返回值

如果由 *hItem* **指定的** tree **项有子项则返回非零值；否则返回** 0。

参数

hItem

一个 tree **项的句柄。**

说明

此成员函数用来确定由 *hItem* **指定的** tree **项是否有子项。如果有，则你可以接着调用** CTreeCtrl::GetChildItem **来获取那些子项。**

请参阅 CTreeCtrl::GetChildItem

CTreeCtrl::Select

BOOL Select(HTREEITEM *hItem*, UINT *nCode*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

一个 tree **项的句柄。**

nCode

要进行的动作的类型。这个参数可以是下列值之一：

- TVGN_CARET **设置给定项的选择。**
- TVGN_DROPHILITE **用指明一次拖放操作的目标的风格重画给定的项。**
- TVGN_FIRSTVISIBLE **垂直滚动该** tree view **以使指定的项成为第一个可见的项。**

说明

此成员函数通过滚动使该项进入视中，或用指明一次拖放操作的目标的风格重

画该项，以选择给定的 tree view 项。

如果 *nCode* 包含的值是 TVGN_CARET，则父窗口接收 TVN_SELCHANGING 和 TVN_SELCHANGED 通知消息。而且，如果指定项是一个收缩的父项的子项，则该父项的子项列表被展开以显示该指定项。在这种情况下，父窗口接收 TVN_ITEMEXPANDING 和 TVN_ITEMEXPANDED 通知消息。

请参阅 CTreeCtrl::SelectItem, CTreeCtrl::GetSelectedItem, CTreeCtrl::SelectDropTarget

CTreeCtrl::SelectDropTarget

BOOL SelectDropTarget(HTREEITEM *hItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

一个 tree 项的句柄。

说明

此成员函数用指明一次拖放操作的目标的风格来重画该项。

请参阅 CTreeCtrl::SelectItem, CTreeCtrl::GetDropHilightItem, CTreeCtrl::CreateDragImage

CTreeCtrl::SetlectItem

BOOL SelectItem(HTREEITEM *hItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

一个 tree 项的句柄。

说明

此成员函数用来选择给定的 tree view 项。如果 *hItem* 是 NULL，则此函数不选择任何项。

请参阅 CTreeCtrl::Select, CTreeCtrl::GetSelectedItem, CTreeCtrl::SelectDropTarget

CTreeCtrl::SelectSetFirstVisible

BOOL SelectSetFirstVisible(HTREEITEM *hItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem
    要被设置为第一个可视项的 tree **项的句柄。**

说明

此成员函数用来垂直滚动该 tree view **以使指定的项成为第一个可视项。此函数用** TVM_SELECTITEM **和** TVGN_FIRSTVISIBLE **消息参数向窗口发送一个消息。**

请参阅 CTreeCtrl::Select, CTreeCtrl::SelectItem, CTreeCtrl::SelectDropTarget

CTreeCtrl::SetBkColor

COLORREF SetBkColor(COLORREF *clr*);

返回值

返回一个代表当前文本颜色的 COLORREF **值。如果返回值是**-1**，则控件正使用系统颜色作为文本颜色。**

参数

clr

一个包含了新的背景颜色的 COLORREF **值。如果这个值是**-1**，则控件将改变为用系统颜色作为背景色。**

说明

此成员函数实现了 Win32 **消息** TVM_SETBKCOLOR **的行为，就像在**“Platform SDK”**中描述的一样。**

请参阅 CTreeCtrl::GetBkColor

CTreeCtrl::SetCheck

BOOL SetCheck(HTREEITEM *hItem*, BOOL *fCheck* = TRUE);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

用来接收改变的核选状态的 HTREEITEM。

fCheck

表明是否要将 tree **控件项核选。缺省的，**SetCheck **将该项设置为被核选的。**

说明

此成员函数用来设置一个 tree **控件项的核选状态。当该** tree **控件项被核选（***fCheck* **设置为** TRUE**），则该项显示一个邻近的核选标记。**

请参阅 CTreeCtrl::GetCheck

CTreeCtrl::SetImageList

CImageList* SetImageList(CImageList* *pImageList*, int *nImageListType*);

返回值

返回指向先前的图像列表的指针（如果有）；否则返回 NULL。

参数

pImageList

指向要被分配的图像列表的指针。如果 *pImageList* 是 NULL，则从 tree view 控件中删除所有的图像。

nImageListType

要设置的图像列表的类型。图像列表可以是下列值之一：

- TVSIL_NORMAL 获取常规的图像列表，它包含了该 tree view 项的被选择的和不被选择的图像。
- TVSIL_STATE 获取状态图像列表，它包含了处于用户定义状态的 tree view 项的图像。

说明

此成员函数用来设置一个 tree view 控件的常规或状态图像列表，并使用新的图

像来重画该 tree view 项的图像。

请参阅 CImageList, CTreeCtrl::GetImageList

CTreeCtrl::SetIndent

```
void SetIndent( UINT nIndent );
```

参数

nIndent

以像素表示的缩进的宽度。如果 *nIndent* **小于系统定义的最小宽度，则新的宽度被设置为系统定义的最小值。**

说明

此成员函数用来为一个 tree view 控件设置缩进的宽度，并重画控件以反映新的宽度。

请参阅 CTreeCtrl::GetIndent, CTreeCtrl::GetItemRect

CTreeCtrl::SetInsertMark

```
BOOL SetInsertMark ( HTREEITEM hItem, BOOL fAfter = TRUE );
```

返回值

如果成功则返回非零值；否则返回0。

参数

hItem

用来指定将在哪一个项处放置插入标记的HTREEEITEM。如果这个参数是NULL，则删除该插入标记。

fAfter

用来指定是将插入标记放置在指定项的前面还是后面的BOOL值。如果这个参数是非零值，则插入标记将被放置在指定项之后。如果此参数是零，则将插入标记放置在指定项之前。

说明

此成员函数实现了Win32消息TVM_SETINSERTMARK的行为，就像在“Platform SDK”中描述的一样。

CTreeCtrl::SetInsertMarkColor

COLORREF SetInsertMarkColor(COLORREF *clrNew*);

返回值

返回一个包含了先前的插入标记颜色的 COLORREF **值。**

参数

clrNew
一个包含了新的插入标记颜色的 COLORREF **值。**

说明

此成员函数实现了 Win32 **消息** TVM_SETINSERTMARKCOLOR **的行为，就像在“** Platform SDK **”中描述的一样。**

请参阅 CTreeCtrl::GetInsertMarkColor

CTreeCtrl::SetItem

BOOL SetItem(TVITEM* *pItem*);
BOOL SetItem(HTREEITEM *hItem*, UINT *nMask*, LPCTSTR *lpszItem*, int *nImage*,
int *nSelectedImage*, UINT *nState*, UINT *nStateMask*, LPARAM *lParam*);

返回值

如果成功则返回非零值；否则返回 0。

参数

pItem

一个指向包含新项属性的 TVITEM **结构的指针，就像在**“Platform SDK”**中描述的一样。**

hItem

要设置其属性的项的句柄。

nMask

指定要设置哪些属性的整数。

lpszItem

一个包含了项的文本的字符串的地址。

nImage

项的图像在 tree view **控件的图像列表中的索引。**

nSelectedImage

该项的被选择图像在 tree view **控件的图像列表中的索引。**

nState

指定项的状态值。

nStateMask

指定要设置哪些状态。

lParam

一个与该项关联的 32 位的应用程序定义值。

说明

此成员函数用来设置指定 tree view 项的属性。

在 TVITEM 结构中，*hItem* 成员标识了这个项，mask 成员指定了要设置的属性。

如果 mask 成员或 *nMask* 参数指定的是 TVIF_TEXT 值，则 pszText 成员或 *lpszItem* 就是一个以空字符结尾的字符串的地址，而 cchTextMax 成员被忽略。如果 *mask*（或 *nMask*）指定的是 TVIF_STATE 值，则 stateMask 成员或 *nStateMask* 参数指定要改变的是哪一个项状态，而 state 成员或 *nState* 参数包含了那些状态的值。

请参阅 CTreeCtrl::GetItem

CTreeCtrl::SetItemData

BOOL SetItemData(HTREEITEM *hItem*, DWORD dwData);

返回值

如果成功则返回非零值；否则返回0。

参数

hItem

  要获取其数据的项的句柄。

dwData

  一个与由 *hItem* **指定的项关联的**32**位的应用程序指定值。**

说明

此成员函数用来设置与指定项关联的32**位的应用程序指定值。**

请参阅 CTreeCtrl::GetItemData

CTreeCtrl::SetItemHeight

SHORT SetItemHeight (SHORT *cyHeight*);

返回值

返回以像素表示的项的先前的高度。

参数

cyHeight

指定 tree view 中的每一个项的新高度。如果这个参数小于图像的高度，则它将被设置为图像的高度。如果这个参数不是偶数值，则它将被设置为与它最接近的小于它的偶数值。如果这个参数是-1，则控件将变为使用它缺省的项高度。

说明

此成员函数实现了 Win32 消息 TVM_SETITEMHEIGHT 的行为，就像在“Platform SDK”中描述的一样。

请参阅 CTreeCtrl::GetItemHeight

CTreeCtrl::SetItemImage

BOOL SetItemImage(HTREEITEM *hItem*, int *nImage*, int *nSelectedImage*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

要设置其图像的项的句柄。

nImage

项的图像在 tree view 控件的图像列表中的索引。

nSelectedImage

项的被选择图像在 tree view 控件的图像列表中的索引。

说明

在一个 tree view 控件中每一个项有一对与之关联的位图图像。图像显示在一个项的标签的左边。其中一个项是在项被选择的时候显示的，另一个项是在项没有被选择的时候显示的。例如，当一个项被选择时它可能被显示为一个打开的文件夹，而当它没有被选择时则显示为一个关闭的文件夹。

此成员函数用来设置项的图像和它的被选择图像在 tree view 控件的图像列表中的索引。

有关图像的更多信息，参见 CImageList。

请参阅 CTreeCtrl::GetItemImage, CImageList

CTreeCtrl::SetItemState

BOOL SetItemState(HTREEITEM *hItem*, UINT *nState*, UINT *nStateMask*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem
要设置其状态的项的句柄。

nState
指定该项的新的状态。

nStateMask
指定要改变哪些状态。

说明

此成员函数用来设置由 *hItem* 指定的项的状态。有关状态的信息，参见 CTreeCtrl::GetItem。

请参阅 CTreeCtrl::GetItem, CTreeCtrl::GetItemState

CTreeCtrl::SetItemText

BOOL SetItemText(HTREEITEM *hItem*, LPCTSTR *lpszItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

要设置其文本的项的句柄。

lpszItem

一个包含项的新文本的字符串的地址。

说明

此成员函数用来设置由 *hItem* **指定的项的文本。**

请参阅 CTreeCtrl::GetItemText

CTreeCtrl::SetTextColor

COLORREF SetTextColor (COLORREF *clr*);

返回值

一个代表先前的文本颜色的 COLORREF 值。如果这个值是-1，则控件使用系统颜色作为文本的颜色。

参数

clr

一个包含了新的文本颜色的 COLORREF 值。如果这个参数是-1，则控件将变为使用系统颜色作为文本的颜色。

说明

此成员函数实现了 Win32 消息 TVM_SETTEXTCOLOR 的行为，就像在“Platform SDK”中描述的一样。

请参阅　CTreeCtrl::GetTextColor

CTreeCtrl::SetToolTips

void SetToolTips(CToolTipCtrl* *pWndTip*);
CToolTipCtrl* SetToolTips (CToolTipCtrl* *pWndTip*);

返回值

返回一个指向包含控件先前使用的工具提示的 CToolTipCtrl 对象，如果先前没有使用工具提示则返回 NULL。

参数

pWndTip

一个指向将要被该 tree 控件使用的 CToolTipCtrl 对象的指针。

说明

此成员函数实现了 Win32 消息 TVM_SETTOOLTIPS 的行为，就像在“Platform SDK”中描述的一样。

要使用工具提示，则需要在创建 CTreeCtrl 对象时指定 TVS_NOTOOLTIPS 风格。

请参阅 CTreeCtrl::GetToolTips, CTreeCtrl::Create

CTreeCtrl::SortChildren

BOOL SortChildren (HTREEITEM *hItem*);

返回值

如果成功则返回非零值；否则返回 0。

参数

hItem

其子项要被排序的父项的句柄。如果 *hItem* 是 NULL，则将从该 tree 的根项开始排序。

说明

此成员函数用来给一个 tree view 控件中的指定父项的子项进行排序。SorChildren 不会对整个 tree 进行递归；只对 *hItem* 的直接子项进行排序。

请参阅 CTreeCtrl::SortChildrenCB

CTreeCtrl::SortChildrenCB

BOOL SortChildrenCB(LPTVSORTCB *pSort*);

返回值

如果成功则返回非零值；否则返回 0。

参数

pSort

指向一个 TVSORTCB 结构的指针。

说明

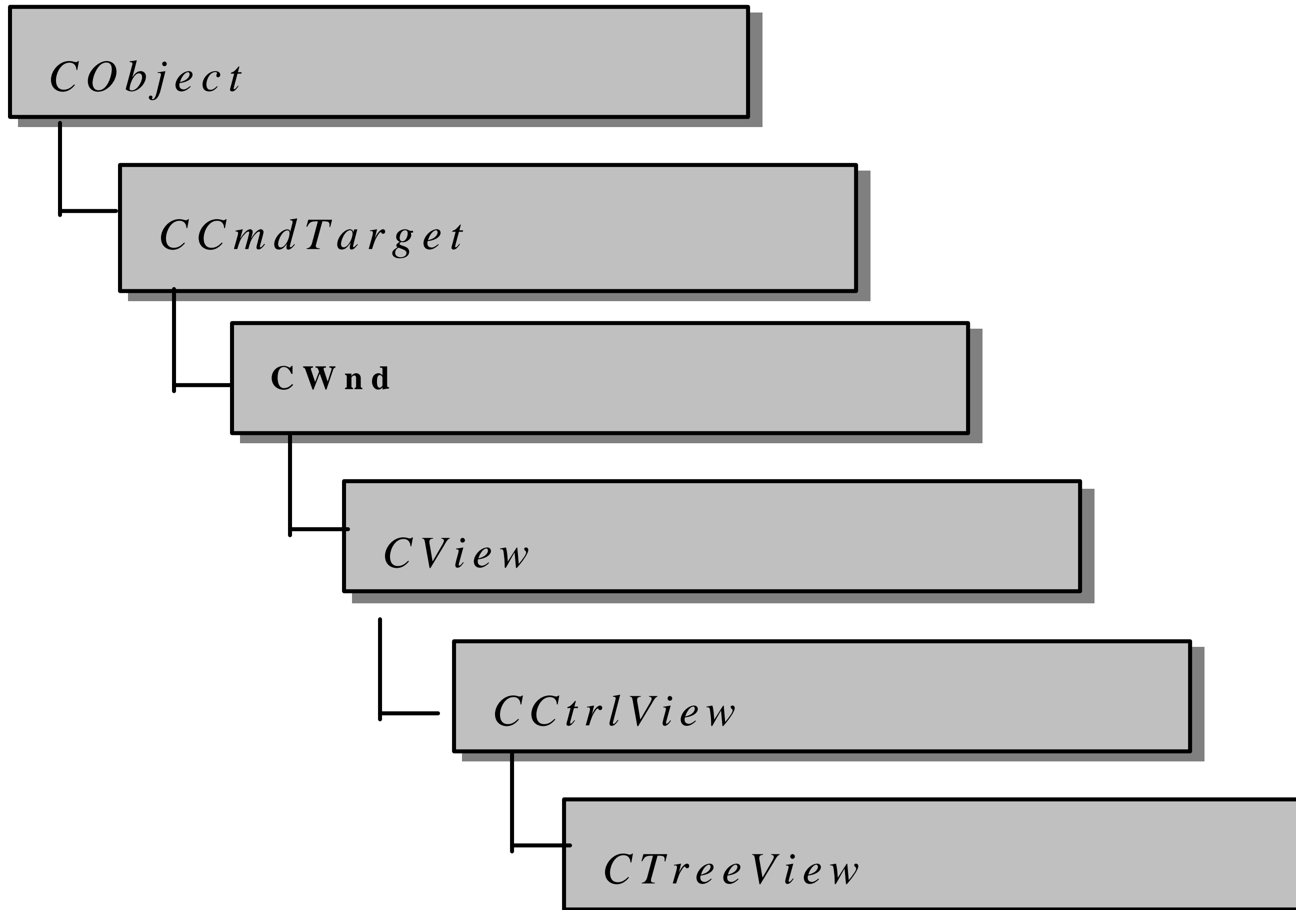
此成员函数通过使用一个应用程序定义的用来比较各项的回收函数来排序 tree view 项。

如果第一项应该在第二项的前面，则结构的比较函数 lpfnCompare 必须返回一个负值；如果第一项应该在第二项的后面则返回一个正值，如果两个项相等则返回零。

lParam1 和 *lParam2* 参数分别对应于两个被比较项的 TVITEM 结构的 lParam 成员。*lParamSort* 初始对应于 TV_SORTCB 结构的 lParam 成员。

请参阅 CTreeCtrl::SortChildren

CTreeView



CTreeView 类简化了对 tree 控件和 CTreeCtrl 类的使用，其中 CTreeCtrl 类使用 MFC 的文档-视结构封装了 tree-控件的性能，有关这种结构的更多信息，参见 CView 类的概述以及其中的交叉索引。

#include <afxcview.h>

请参阅 CView, CCtrlView, CTreeCtrl

CTreeView 类成员

Construction	
CTreeView	构造一个 CTreeView 对象

Attributes	
GetTreeCtrl	返回与视关联的 tree 控件

成员函数

CTreeView:: CTreeView

CTreeView( );

说明

此成员函数用来构造一个 CTreeView 对象。

请参阅 CTreeCtrl

CTreeView::GetTreeCtrl

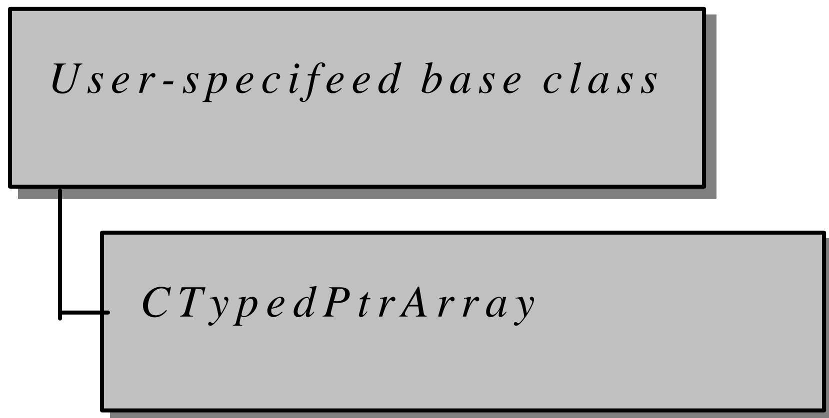
CTreeCtrl& GetTreeCtrl( ) const;

说明

此成员函数返回一个对与该视关联的 tree 控件的引用。

请参阅 CtreeCtrl

CTypedPtrArray



template <class *BASE_CLASS*, class *TYPE*> class CTypedPtrArray : public *BASE_CLASS*

参数

BASE_CLASS

类型指针数组类的基类；必须是一个数组类（CObArray **或** CPtrArray**）。**.

TYPE

保存在基类数组中的元素的类型。

说明

CTypedPtrArray 类为 CPtrArray 或 CObArray 类对象提供了一个类型-安全“包装”。当你使用 CTypedPtrArray 而不是 CPtrArray 或 CObArray 时，C++类型-检查工具帮助消除由不匹配的指针类型引发的错误。

另外，CTypedPtrArray 包装了许多在你使用 CObArray 或 CPtrArray 时必须使用的强制转换。

由于所有的 CTypedPtrArray 函数都是内联的，所以使用这个模板不会使你的代码的大小或速度受到很大的影响。

有关使用 CTypedPtrArray 的更多信息，参见“Visual C++程序员指南”中的文章“集合”和“集合：基于模板的类”。

#include <afxtempl.h>

请参阅　CPtrArray, CObArray

CTypedPtrArray 类成员

Element Access

GetAt	返回给定索引处的值
ElementAt	返回一个对数组中的元素指针的模板引用
SetAt	为一个给定索引设置值，不允许数组增长
SetAtGrow	为一个给定索引设置值，允许数组增长
Add	将一个新元素添加到设置的尾部。如果必要可以增长数组
Append	将一个设置的内容添加到另一个数组的结尾。如果必要可以增长数组
Copy	将另一个数组拷贝到该数组，如果必要可以增长该数组
InsertAt	在一个指定索引处插入一个元素（或另一个数组中的索引元素）

Operators

operator	设置或获取指定索引处的元素

成员函数

CTypedPtrArray::Add

int Add(*TYPE newElement*);

返回值

返回被添加元素的索引。

参数

TYPE

指定要添加到数组中的元素的类型的模板参数。

newElement

要添加到此数组中去的元素。

说明

此成员函数调用 BASE_CLASS::Add。**更多的细节说明，参见** CObArray::Add。

CTypedPtrArray::Append

int Append(const CTypedPtrArray<*BASE_CLASS*, *TYPE*>& *src*);

返回值

返回第一个被添加元素的索引。

参数

BASE_CLASS

类型指针数组类的基类；必须是一个数组类（CObArray **或** CPtrArray**）。**

TYPE

保存在基类数组中的元素的类型。

src

要被添加到一个数组中的元素的源。

说明

此成员函数调用 *BASE_CLASS*::Append**。更多的细节说明，参见** CObArray::Add**。**

CTypedPtrArray::Copy

void Copy(const CTypedPtrArray<*BASE_CLASS*, *TYPE*>& *src*);

参数

BASE_CLASS

类型指针数组类的基类；必须是一个数组类（CObArray 或 CPtrArray）。

TYPE

保存在基类数组中的元素的类型。

src

要被拷贝到一个数组中的元素的源。

说明

此成员函数调用 *BASE_CLASS*::Copy。更多的细节说明，参见 CObArray::Add。

CTypedPtrArray::ElementAt

TYPE& ElementAt(int *nIndex*);

返回值

返回一个对位于由 *nIndex* 指定的位置的元素的模板引用。这个元素的类型是由 *TYPE* 模板参数指定的。

参数

TYPE

指定保存在此数组的元素类型的模板参数。

nIndex

一个整数索引，它大于或等于 0，小于或等于由 *BASE_CLASS*::GetUpperBound 返回的值。

说明

此成员函数调用 *BASE_CLASS*::ElementAt。更多的细节说明，参见 CObArray::ElementAt。

请参阅 CObArray::ElementAt, CObArray::GetUpperBound

CTypedPtrArray::GetAt

TYPE GetAt(int *nIndex*) const;

返回值

返回位于 *nIndex* **指定的位置的元素的拷贝。这个元素的类型由** *TYPE* **模板参数指定。**

参数

TYPE

指定保存在此数组的元素的类型的模板参数。

nIndex

一个整数索引，它大于或等于 0**，小于或等于由** *BASE_CLASS*::GetUpperBound **返回的值。**

说明

此成员函数调用 *BASE_CLASS*::GetAt。**更多的细节说明，参见** CObArray::GetAt。

请参阅 CObArray::GetAt, CObArray::GetUpperBound

CTypedPtrArray::InsertAt

void InsertAt(int *nIndex*, *TYPE newElement*, int *nCount* = 1);
void InsertAt(int *nStartIndex*, CTypedPtrArray<*BASE_CLASS*, *TYPE*>*

pNewArray);

参数

nIndex

一个整数索引，可能大于由 CObArray::GetUpperBound **返回的值。**

TYPE

保存在基类数组中的元素的类型。

newElement

要放置在此数组中的对象指针。*newElement* **的值可以是** NULL**。**

nCount

此元素将被插入的次数（缺省为 1**）。**

nStartIndex

一个可能大于由 CObArray::GetUpperBound **返回值的整数索引。**

BASE_CLASS

类型指针数组类的基类；必须是一个数组类（CObArray **或** CPtrArray**）。**

pNewArray

另一个包含要被添加到此设置中的元素的数组。

说明

此成员函数调用 *BASE_CLASS*::InsertAt。**更多的细节说明，参见** CObArray::InsertAt。

CTypedPtrArray::SetAt

void SetAt(int *nIndex*, *TYPE ptr*);

参数

nIndex

一个整数索引，它大于或等于 0**，小于或等于由** *CObArray*::GetUpperBound **返回的值。**

TYPE

保存在基类数组中的元素类型。

ptr

一个指向要被插入此数组的指定索引处的元素的指针。可以是 NULL。

说明

此成员函数调用 *BASE_CLASS*::SetAt。**更多的细节说明，参见**

CObArray::SetrtAt。

CTypedPtrArray::SetAtGrow

void SetAtGrow(int *nIndex*, TYPE *newElement*);

参数

nIndex

　　一个整数索引，它大于或等于 0。

TYPE

　　保存在基类数组的元素类型。

newElement

　　要被添加到此数组中的对象指针。可以是 NULL。

说明

此成员函数调用 *BASE_CLASS*::SetAtGrow **。更多的细节说明，参见** CObArray::SetAtGrow。

操作符

CTypedPtrArray::operator[ ]

TYPE& operator[ ](int *nIndex*);
TYPE operator[ ](int *nIndex*) const;

参数

TYPE
指定保存在此数组的元素的类型的模板参数。

nIndex
一个整数索引，它大于或等于 0，小于或等于由 *BASE_CLASS*::GetUpperBound 返回的值。

说明

这些内联操作符调用 *BASE_CLASS*::operator[ ]。
第一个操作符，对不是 const 的数组使用，可以使用在一个赋值语句的右边（r-值）或左边（l-值）。第二个操作符，是对 const 数组使用的，只能被使用在右边。

请参阅 CObArray::operator []

CTypedPtrList

template< class ***BASE_CLASS*****, class** ***TYPE*** **>**
class CTypedPtrList : public ***BASE_CLASS***

参数

BASE_CLASS

　　类型指针列表类的基类；必须是一个指针列表类（COblist **或** CPtrList**）。**

TYPE

　　保存在基类列表中的元素的类型。

说明

CTypedPtrList 类为类 CPtrList 的对象提供了一个类型-安全的“包装”。当你使用 CTypedPtrList 而不是 CObList 或 CPtrList 的时候，C++类型-检查工具帮助消除由不匹配的指针类型引发的错误。

另外，CTypedPtrList 包装实现了许多在使用 CObList 或 CPtrList 时要实现的强制转换。

因为所有的 CTypedPtrList 函数都是内联的，所以使用这个模板不会明显地影响你的代码的大小和速度。

从 CObList 派生的列表可以是连续的，但是从 CPtrList 派生的列表却不能。

当一个 CTypedPtrList 对象被删除时，或者是当它的元素被删除时，只有指针被删除了，而它们所引用的项并没有被删除。

有关使用 CTypedPtrList 的更多信息，参见“Visual C++程序员指南”中的文章“集合”和“基于模板的类”。

#include <afxtempl.h>

请参阅　CPtrList, CObList

CTypedPtrList 类成员

Head/Tail Access	
GetHead	返回列表的头元素（不能是空的）
GetTail	返回列表的尾元素（不能是空的）

Operations	
RemoveHead	从列表的头部删除元素
RemoveTail	从列表的尾部删除元素
AddHead	将一个元素（或另一个数组中的所有元素）添加到列表的头部（产生一个新的头部）
AddTail	将一个元素（或另一个数组中的所有元素）添加到列表的尾部（产生一个新的尾部）

Iteration	
GetNext	获取用于反复的下一个元素
GetPrev	获取用于反复的前一个元素

Retrieval/Modification	
GetAt	获取在一个给定位置处的元素
SetAt	设置在一个给定位置处的元素

成员函数

CTypedPtrList::AddHead

POSITION AddHead(*TYPE newElement*);
void AddHead(CTypedPtrList<*BASE_CLASS*, *TYPE*> **pNewList*);

返回值

第一种版本返回新插入的元素的 POSITION 值。

参数

TYPE
　　保存在基类列表中的元素的类型。

newElement
　　要添加到此列表中的对象指针。可以是 NULL 值。

BASE_CLASS
　　此类型指针列表类的基类；必须是一个指针列表类（CObList 或 CPtrList）。

pNewList

一个指向另一个 CTypedPtrList 对象的指针。在 *pNewList* 中的元素将被添加到列表中。

说明

此成员函数调用 *BASE_CLASS*::AddHead。第一种版本将一个新元素添加到列表的头元素之前。第二种版本将另一个列表中的元素添加到此列表的头元素之前。

CTypedPtrList::AddTail

POSITION AddTail(*TYPE newElement*);
void AddTail(CTypedPtrList<*BASE_CLASS*, *TYPE*> **pNewList*);

返回值

第一个版本返回新插入的元素的 POSITION 值。

参数

TYPE

保存在基类列表中的元素的类型。

newElement

要添加到此列表中的对象指针。可以是 NULL 值。

BASE_CLASS

此类型指针列表类的基类；必须是一个指针列表类（CObList 或 CPtrList）。

pNewList

一个指向另一个 CTypedPtrList 对象的指针。在 pNewList 中的元素将被添加到列表中。

说明

此成员函数调用 *BASE_CLASS*::AddTail。第一种版本将一个新元素添加到列表的尾元素之后。第二种版本将另一个列表中的元素添加到此列表的尾元素之后。

CTypedPtrList::GetAt

TYPE& GetAt(POSITION *position*);
TYPE GetAt(POSITION *position*) const;

返回值

如果是通过一个指向 const CTypedPtrList 的指针访问此列表，则 GetAt 返回一个类型由模板参数 *TYPE* 指定的指针。这使此函数只能被使用在赋值语句的右

边，这样就保护了列表不被修改。

如果列表被直接访问，或通过一个指向 CTypedPtrList 的指针访问，则 GetAt 返回对一个类型由模板参数 TYPE 指定的指针的引用。这使得此函数可以使用在赋值语句的任何一边，从而允许该列表可以被修改。

参数

TYPE

指定保存在列表中的元素类型的模板参数。

position

一个由先前调用 GetHeadPosition 或 Find 成员函数返回的 POSITION 值。

说明

一个类型为 POSITION 的变量是此列表的一个关键字。它与索引是不一样的，你不能自己处理一个 POSITION 值。GetAt 获取与一个给定位置关联的 CObject 指针。

你必须确保你的 POSITION 值表示的是列表中的一个有效位置。如果它是无效的，则 Microsoft 基础类库的调试版将给出断言。

这个内联函数调用了 *BASE_CLASS*::GetAt。

请参阅 CObList::GetAt

CTypedPtrList::GetHead

TYPE& GetHead( );
TYPE GetHead( ) const;

返回值

如果是通过一个指向 const CTypedPtrList 的指针访问此列表，则 GetHead 返回一个类型由模板参数 *TYPE* 指定的指针。这使此函数只能被使用在赋值语句的右边，这样就保护了列表不被修改。

如果列表被直接访问，或通过一个指向 CTypedPtrList 的指针访问，则 GetHead 返回对一个类型由模板参数 *TYPE* 指定的指针的引用。这使得此函数可以使用在赋值语句的任何一边，从而允许该列表可以被修改。

参数

TYPE
　　指定保存在列表中的元素类型的模板参数。

说明

此成员函数用来获取代表此列表中的头元素的指针。

在调用 GetHead 之前，你必须保证该列表不是空的。如果该列表是空的，则 Microsoft 基础类库的调试版将给出断言。可以使用 IsEmpty 来检验该列表是否包含元素。

请参阅 CPtrList::IsEmpty, CTypedPtrList::GetTail, CTypedPtrList::GetNext, CTypedPtrList::GetPrev

CTypedPtrList::GetNext

TYPE& GetNext(POSITION& *rPosition*);
TYPE GetNext(POSITION& *rPosition*) const;

返回值

如果是通过一个指向 const CTypedPtrList 的指针访问此列表，则 GetNext 返回一个类型由模板参数 *TYPE* 指定的指针。这使此函数只能被使用在赋值语句的右边，这样就保护了列表不被修改。

如果列表被直接访问，或通过一个指向 CTypedPtrList 的指针访问，则 GetNext 返回对一个类型由模板参数 *TYPE* 指定的指针的引用。这使得此函数可以使用在赋值语句的任何一边，从而允许该列表可以被修改。

参数

TYPE

指定保存在列表中的元素类型的模板参数。

rPosition

一个对先前调用 GetNext,GetHeadPosition 或其他成员函数返回的 POSITION 值的引用。

说明

此成员函数用来获取由 *rPosition* 标识的列表元素，然后将 *rPosition* 设置为列表中的下一个项的 POSITION 值。如果你是通过调用 GetHeadPosition 或 CPtrList::Find 来建立初始位置的，你就可以使用 GetNext 来实现一个向前的反复循环。

你必须保证你的 POSITION 值代表的是列表中的一个有效位置。如果位置是无效的，则 Microsoft 基础类库的调试版将给出断言。

如果获取的元素是列表中的最后一个，则 *rPosition* 被设置为新值 NULL。

在一个反复中删除一个元素是有可能的。参见 CObList::RemovAt 的示例。

请参阅 CObList::Find, CObList::GetHeadPosition, CObList::GetTailPosition, CTypedPtrList::GetPrev, CTypedPtrList::GetHead, CTypedPtrList::GetTail

CTypedPtrList::GetPrev

TYPE& GetPrev(POSITION& *rPosition*);
TYPE GetPrev(POSITION& *rPosition*) const;

返回值

如果是通过一个指向 const CTypedPtrList 的指针访问此列表，则 GetPrev 返回一个类型由模板参数 *TYPE* 指定的指针。这使此函数只能被使用在赋值语句的右边，这样就保护了列表不被修改。

如果列表被直接访问，或通过一个指向 CTypedPtrList 的指针访问，则 GetPrev 返回对一个类型由模板参数 *TYPE* 指定的指针的引用。这使得此函数可以使用在赋值语句的任何一边，从而允许该列表可以被修改。

参数

TYPE
　　指定保存在列表中的元素类型的模板参数。

rPosition
　　一个对先前调用 GetPrev 或其它成员函数返回的 POSITION 值的引用。

说明

此成员函数用来获取由 *rPosition* 标识的列表元素，然后将 *rPosition* 设置为列表中的下一个项的 POSITION 值。如果你是通过调用 GetTailPosition 或 Find 来建立初始位置的，你就可以使用 GetPrev 来实现一个反向的反复循环。

你必须保证你的 POSITION 值代表的是列表中的一个有效位置。如果位置是无效的，则 Microsoft 基础类库的调试版将给出断言。

如果获取的元素是列表中的第一个，则 *rPosition* 被设置为新值 NULL。

请参阅 CPtrList::Find, CPtrList::GetTailPosition, CPtrList::GetHeadPosition, CTypedPtrList::GetNext, CTypedPtrList::GetHead, CTypedPtrList::GetTail

CTypedPtrList::GetTail

TYPE& GetTail( );
TYPE GetTail( ) const;

返回值

如果是通过一个指向 const CTypedPtrList 的指针访问此列表，则 GetTail 返回一个类型由模板参数 *TYPE* 指定的指针。这使此函数只能被使用在赋值语句的右边，这样就保护了列表不被修改。

如果列表被直接访问，或通过一个指向 CTypedPtrList 的指针访问，则 GetTail 返回对一个类型由模板参数 *TYPE* 指定的指针的引用。这使得此函数可以使用在赋值语句的任何一边，从而允许该列表可以被修改。

参数

TYPE
指定保存在列表中的元素类型的模板参数。

说明

此成员函数用来获取此列表中的头元素。

在调用 GetTail 之前，你必须保证该列表不是空的。如果列表是空的，则 Microsoft 基础类库的调试版将给出断言。使用 IsEmpty 来检验列表是否包含元素。

请参阅 CPtrList::IsEmpty, CPtrList::Find, CPtrList::GetTailPosition, CPtrList::GetHeadPosition, CTypedPtrList::GetPrev, CTypedPtrList::GetNext, CTypedPtrList::GetHead

CTypedPtrList::RemoveHead

TYPE RemoveHead( );

返回值

返回先前在列表头部的指针。这个指针的类型由模板参数 *TYPE* **指定。**

参数

TYPE

　　指定保存在列表中的元素类型的模板参数。

说明

此成员函数从列表的头部删除元素并返回这个元素。

在调用 RemoveHead **之前，你必须保证该列表不是空的。如果列表是空的，则** Microsoft **基础类库的调试版将给出断言。使用** IsEmpty **来检验列表是否包含元素。**

请参阅 CTypedPtrList::RemoveTail, CPtrList::IsEmpty, CPtrList::GetHead, CPtrList::AddHead

CTypedPtrList::RemoveTail

TYPE RemoveTail( );

返回值

返回先前在列表尾部的指针。这个指针的类型由模板参数 *TYPE* 指定。

参数

TYPE
指定保存在列表中的元素类型的模板参数。

说明

此成员函数从列表的尾部删除元素并返回这个元素。

在调用 RemoveHead 之前，你必须保证该列表不是空的。如果列表是空的，则 Microsoft 基础类库的调试版将给出断言。使用 IsEmpty 来检验列表是否包含元素。

请参阅 CTypedPtrList::RemoveHead, CPtrList::IsEmpty, CPtrList::GetTail, CPtrList::AddTail

CTypedPtrList::SetAt

void SetAt(POSITION *pos*, *TYPE newElement*);

参数

pos

要被设置的元素的 POSITION。

TYPE

保存在基类列表中的元素类型。

newElement

要被写入列表中去的对象指针。

说明

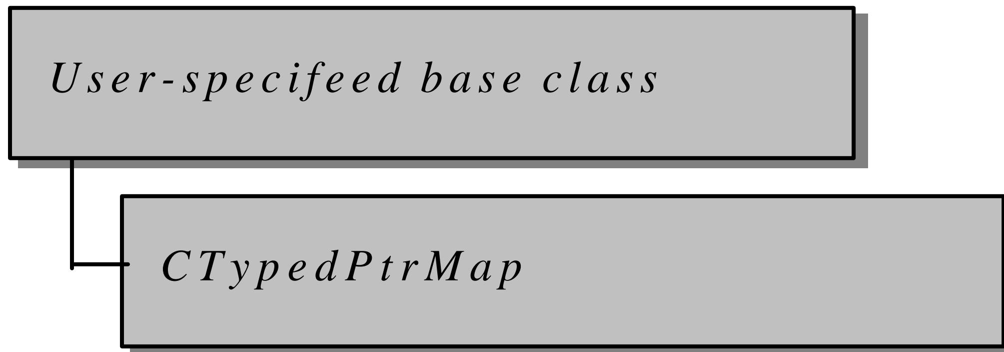
此成员函数调用 *BASE_CLASS*::SetAt。

一个类型为 POSITION 的变量是此列表的一个关键字。它与索引是不一样的，你不能自己处理一个 POSITION 值。SetAt 写对象指针到列表中指定位置。

你必须确保你的 POSITION 值表示的是列表中的一个有效位置。如果它是无效的，则 Microsoft 基础类库的调试版将给出断言。

更多的细节说明，参见 CObList::SetAt。

CTypedPtrMap



template< class *BASE_CLASS*, class *KEY*, class *VALUE* >
class CTypedPtrMap : public BASE_CLASS

参数

BASE_CLASS
此类型指针映射类的基类；它必须是一个指针映射类（CMapPtrToPtr，

CMapPtrToWord，CMapWordToPtr，或 CMapStringToPtr）。

KEY

被用来作为该映射的关键字的对象的类。

VALUE

保存在该映射中的对象的类。

说明

CTypedPtrMap 类为指针-映射类 CMapPtrToPtr，CMapPtrToWord，CMapWordToPtr 和 CMapStringToPtr 提供了安全类型的“包装”。当你使用 CTypedPtrMap 时，C++的类型-检查工具帮助消除由于不匹配的指针类型引发的错误。

由于所有的 CTypedPtrMap 函数都是内联的，所以使用这个模板不会明显地影响你的代码的大小或速度。

有关使用 CTypedPtrMap 的更多信息，参见“Visual C++程序员指南”中的文章“集合”和“集合：基于模板的类”。

```
#include <afxtempl.h>
```

请参阅 CMapPtrToPtr, CMapPtrToWord, CMapWordToPtr, CMapStringToPtr

CTypedPtrMap 类成员

Element Access	
Lookup	返回一个基于某个 *VALUE* 的 *KEY*
GetNextAssoc	获取下一个用于反复的元素
RemoveKey	删除一个由关键字指定的元素
SetAt	如果找到了匹配的关键字，则向映射中插入一个元素来代替一个已有的元素

Operators	
operator	向映射中插入一个元素

成员函数

CTypedPtrMap::GetNextAssoc

void GetNextAssoc(POSITION& *rPosition*, *KEY*& *rKey*, *VALUE*& *rValue*) const;

参数

rPosition
　　指定一个对先前调用 GetNextAssoc 或 *BASE_CLASS*::GetStartPosition 返回的 POSITION 值的引用。

KEY
　　指定映射的关键字的类型的模板参数。

rKey
　　指定被获取元素的返回关键字。

VALUE
　　指定映射的值的类型的模板参数。

rValue
　　指定被获取元素的返回值。

说明

此成员函数获取 *rNextPosition* 处的映射元素，然后更新 *rNextPosition* 使其指向映射中的下一个元素。在对映射中的所有元素进行反复时此函数是最有用的。注意，位置的顺序不必与关键值的顺序一样。

如果被获取的元素是映射中的最后一个元素，则 *rNextPosition* 的新值被设置为 NULL。

这个内联函数调用 *BASE_CLASS*::GetNextAssoc。

请参阅 CMapStringToOb::GetNextAssoc, CMapStringToOb::GetStartPosition

CTypedPtrMap::Lookup

BOOL Lookup(*BASE_CLASS*::BASE_ARG_KEY *key*, *VALUE*& *rValue*) const;

返回值

如果找到了该元素则返回非零值；否则返回 0。

参数

BASE_CLASS

指定此映射类的基类的模板参数。

key

指定将被查找的元素的关键字。

VALUE

指定保存在此映射中的值的类型的模板参数。

rValue

指定被获取元素的返回值。

说明

Lookup 使用散列法运算法则来快速查找与某个关键字恰好匹配的映射元素。这个内联函数调用 *BASE_CLASS*::Lookup。

请参阅 CMapStringToOb::Lookup

CTypedPtrMap::RemoveKey

BOOL RemoveKey(*KEY key*);

返回值

如果找到了指定的项并成功删除，则返回非零值；否则返回 0。

参数

KEY
  指定映射的关键字的类型的模板参数。

key
  要被删除的元素关键字。

说明

此成员函数调用了 *BASE_CLASS*::RemoveKey**。更多的细节说明，参见** CMapStringToOb:: RemoveKey。

CTypedPtrMap::SetAt

void SetAt(*KEY key*, *VALUE newValue*);

参数

KEY

指定映射的关键字的类型的模板参数。

key

指定 newValue **的关键字值。**

newValue

指定新元素的值的对象指针。

说明

此成员函数调用了 *BASE_CLASS*::SetAt**。更多的细节说明，参见** CMapStringToOb::SetAt。

操作符

CTypedPtrMap::operator[ ]

VALUE& operator [ ](*BASE_CLASS*::BASE_ARG_KEY *key*);

参数

VALUE
  指定保存在此映射中的值的类型的模板参数。

BASE_CLASS
  指定此映射类的基类的模板参数。

key
  指定映射中的将被查找或创建的元素的关键字。

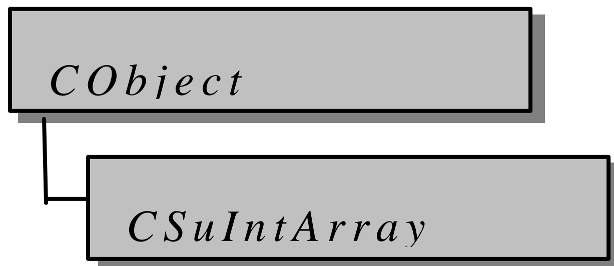
说明

此成员函数只能被使用在一个赋值语句的左边（一个 l-值）。如果对于指定的关键字没有映射元素，则创建一个新的元素。这个操作符没有“右边”（r-值）的等价操作符，因为在映射中有可能找不到某个关键字。使用 Lookup 成员函

数可以进行元素恢复。

请参阅 CTypedPtrMap::Lookup

CUIntArray



CUIntArray 类支持无符号整数数组。一个无符号的整数，或 UINT，与字和双字是不一样的，一个 UINT 的物理大小可以根据目标操作环境而改变。在 Windows 3.1 版下，一个 UINT 与一个 WORD 的大小是一样的。在 Windows NT 和 Windows 95 下，一个 UINT 的大小与一个双字一样大。

CUIntArray 的成员函数类似于类 CObArray 的成员函数。由于这个相似性，你可以使用特定成员函数的 CObArray 引用文件。在你看见一个 CObject 指针作为函数参数或返回值的地方，用一个 UINT 来代替。例如：

```
CObject* CObArray::GetAt( int <nIndex> ) const;
```

可以被转换为

UINT CUIntArray::GetAt(int <nIndex>) const;

CUIntArray 与 IMPLEMENT_DYNAMIC 宏一起支持运行时类型访问和转储到一个 CDumpContext 对象。如果你需要单个无符号整数元素的转储，你必须将转储环境的深度设置为 1 或更大。无符号整数数组不能被连续。

> 注意 在使用一个数组之前，使用 SetSize 来建立它的大小并给它分配内存。如果你不使用 SetSize，当向你的数组添加元素时会导致频繁地重定位和拷贝。频繁的重定位和拷贝会降低效率并产生内存碎片。

有关使用 CUIntArray 的更多信息，参见“ Visual C++程序员指南 ”中的文章“ 集合 ”。

#include <afxcoll.h>

CUIntArray 类成员

Construction

CUIntArray	构造一个无符号整数的空数组

Bounds

GetSize	获取此数组中的元素的数目

GetUpperBound	获取最大的有效索引
SetSize	设置包含在此数组中的元素数目

Operations

FreeExtra	释放在当前上边界之上的所有没有使用的内存
RemoveAll	从此数组中删除所有的元素

Element Access

GetAt	返回一个给定索引处的值
SetAt	设置一个给定索引的值，不允许数组增长
ElementAt	返回一个对数组中的元素指针的临时引用
GetData	允许访问数组中的元素。可以是 NULL

Growing the Array

SetAtGrow	设置一个给定索引的值。允许数组的必要增长
Add	在数组的最后添加一个元素，允许数组的必要增长
Append	给此数组添加另一个数组，允许数组的必要增长
Copy	将另一个数组拷贝到该数组，允许数组的必要增长

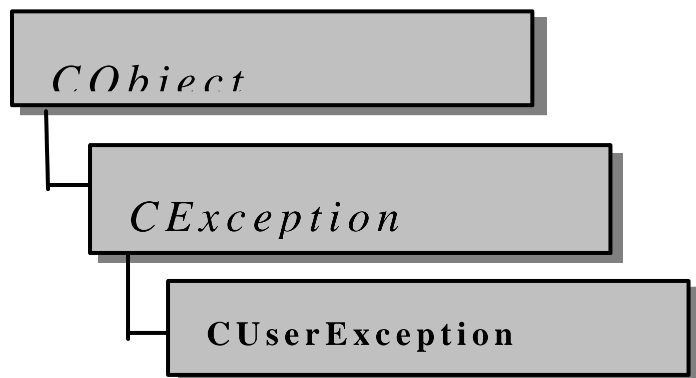
Insertion/Removal

InsertAt	在指定索引处插入一个元素（或另一个数组中的所有元素）
RemoveAt	删除指定索引处的元素

Operators

operator	设置或获取指定索引处的元素

CUserException



抛出一个 CUserException 来停止一个结束-用户操作。当你想要对应用程序指定的异常使用抛出/捕获异常机制时，可以使用 CUserException。在类名中的“User”可以解释为“我的用户做了某些我不想处理的异常事情”。

在调用全局函数 AfxMessageBox 之后，通常会抛出一个 CUserException 来通知用户操作失败了。当你编写一个异常处理程序时，由于用户已经获得了失败的通知，所以你要特别处理这个异常。在某些情况下，框架会抛出这个异常。你可以自己抛出一个 CUserException，以警告用户，然后调用全局函数 AfxThrowUserException。

在下面的示例中，一个函数包含的操作可能会失败，警告用户并抛出一个CUserException。调用函数捕获这个异常并特别处理它：

```
void DoSomeOperation( )
{
   // 如果出错则处理
   AfxMessageBox( "The x operation failed" );
   AfxThrowUserException( );
}
BOOL TrySomething( )
{
   TRY
   {
      // Could throw a CUserException or other exception.
      DoSomeOperation( );
   }
   CATCH( CUserException, e )
   {
      return FALSE;       // User already notified.
   }
   AND_CATCH( CException, e )
   {
      // For other exception types, notify user here.
```

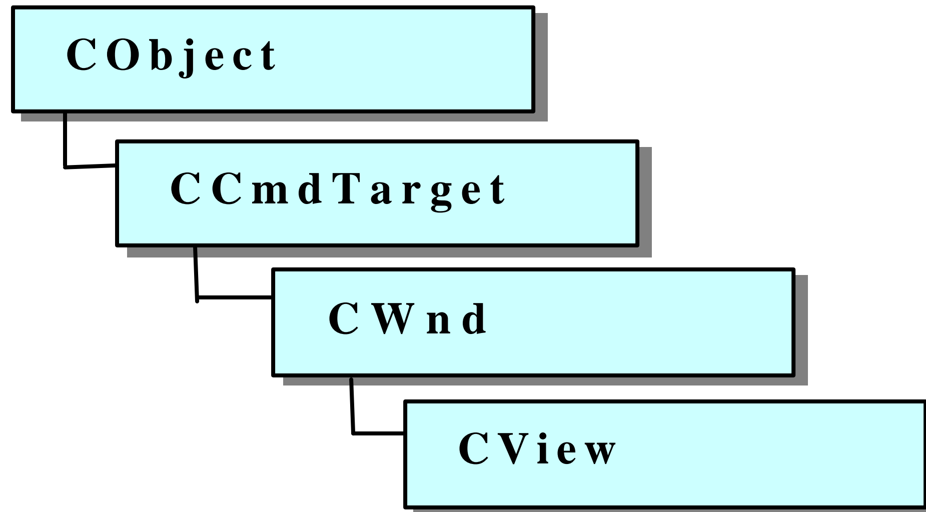
```
        AfxMessageBox( "Some operation failed" );
        return FALSE;
    }
    END_CATCH
    return TRUE;      // No exception thrown.
}
```

有关使用 CUserException **的更多信息，参见“** Visual C++**程序员指南”中的“异常”。**

```
#include <afxwin.h>
```

请参阅 Exception, AfxMessageBox, AfxThrowUserException

CView



CView 类为用户定义的视图类提供了基本的功能。视图被连接到文档上，用作文档和用户之间的媒介：视图在屏幕或打印机上显示文档的图像，并将用户的输入解释为对文档的操作。

视图是框架窗口的子窗口。可能会有多个视图共用一个框架窗口，就像在分隔窗口中那样。视图类、框架窗口类和文档类之间的联系通过 CDocTemplate 类来建立。当用户打开一个新窗口或将现有窗口分隔为多个时，框架会创建一个新视图并将它连接到文档对象上。

一个视图只能被连接到一个文档，但是一个文档可以有多个视图与之相连接，

例如，在分隔窗口或多文档界面（MDI）应用程序的多重子窗口中显示的文档就是如此。对于给定的文档类型，应用程序可以支持不同类型的视图；例如，一个字处理程序可能既要提供文档的完整文本视图，又要提供只显示每节标题的大纲视图。这些不同的视图类型可以放在不同的框架窗口中，如果你使用的是分隔窗口，你也可以把它们放在同一框架窗口的不同板块中。

视图可以响应几种类型的输入，例如键盘输入，鼠标输入或拖放输入，还有菜单、工具条和滚动条产生的命令输入。视图接收框架窗口发送给它的命令，如果视图不接受一个给定的命令，它就将这个命令发送给相连接的文档。与所有的命令目标一样，视图类通过消息映射处理消息。

当文档的数据发生变化时，视图类响应这种变化，通常调用文档的 CDocument::UpdateAllViews 函数，通知所有其它的视图调用 OnUpdate 函数。OnUpdate 数的缺省实现使视图的整个用户区域无效。你可以重载这个函数，只使视图中与文档的变化部分相对应的区域无效。

如果要使用 CView，应当从它派生一个类，并实现它的 OnDraw 函数以在屏幕上显示。你还可以利用 OnDraw 函数来进行打印和打印预览。框架将处理打印循环以实现对文档的打印和打印预览。

通过 CWnd::OnHScroll 和 CWnd::OnVScroll 成员函数来处理滚动条消息。你可以在这些函数中实现对滚动消息的处理，你也可以利用 CView 的派生类 SCrollView 来处理滚动。

除了 CScrollView 以外，微软基础类库还提供了其它的 CView 派生类：

- CCtrlView，允许你在树，列表和带格式编辑控件中使用文档/视图结构。
- CDaoRecordView，在对话框控件中显示数据库记录的视图。
- CEditView，提供了一个简单的多行文本编辑器的视图。你可以将CEditView用作对话框中的一个控件，也可以将它用作文档的视图。
- CFormView，一种可以滚动的视图，其中包含了对话框控件，它建立在对话框模板资源的基础上。
- CListView，使你能够在列表控件中使用文档/视结构的视图。
- CRecordView，在对话框控件中显示数据库记录的视图。
- CRichEditView，使你能够在带格式编辑控件中使用文档/视图结构的视图。
- CScrollView，自动提供滚动支持的一种视图。
- CTreeView，使你能够在树控件中使用文档/视图结构的视图。

CView 类还有一种派生类，名为 CPreviewView，它被框架用于实现打印预览。这个类提供了对打印预览窗口特性的支持，例如工具条、单页或双页预览以及放大（被用来放大预览的图像）等。你没有必要调用或重载 CPreviewView 的任何成员函数，除非你想实现自己的打印预览界面（例如，如果你希望支持在预览模式下的编辑）。有关使用 CView 的更多信息参见“Visual C++程序员指南”中的“文档/视图结构主题”和“打印”。另外，有关自定义打印预览的更多细节可以参见“Visual C++文档”中的“技术注释 30”。

#include <afxwin.h>

请参阅 CWnd, CFrameWnd, CSplitterWnd, CDC, CDocTemplate, CDocument

CView 类成员

Operations	
DoPreparePrinting	显示 Print 对话框，创建打印机设备环境；如果重载了 OnPreparePrinting 成员函数，则调用之
GetDocument	返回与视图相连接的文档

OLE Overridables	
OnDragEnter	当某项第一次被拖入视图的拖放区域时调用这个函数
OnDragLeave	当某个被拖的项离开视图的拖放区域时调用这个函数
OnDragOver	当某项被拖过视图的拖放区域时调用这个函数
OnDrop	当某项被放入视图的拖放区域时调用这个函数，这是缺省的处理函数
OnDropEx	当某项被放入视图的拖放区域时调用这个函数，这是主要的处理函数
OnDragScroll	调用这个函数以确定光标是否被拖入窗口的滚动区域
OnInitialUpdate	在一个视图第一次与文档连接的时候调用这个函数
OnScrollBy	当一个包含活动的现场可编辑 OLE 项的视图被滚动时调用这个函数
OnScroll	当 OLE 项被拖离视图的边界时调用这个函数

Overridables	
IsSelected	测试一个文档项是否被选中。用于 OLE 支持
OnActivateView	当一个视被激活时调用
OnActivateFrame	当包含了视的框架窗口被激活或失去活动状态时调用这个函数
OnBeginPrinting	开始打印作业时调用这个函数；重载这个函数以分配图形设备接口（GDI）资源
OnDraw	调用这个函数画出文档的图像，用于屏幕显示，打印或打印预览。需要提供其实现
OnEndPrinting	当打印作业结束时调用这个函数。重载这个函数以释放 GDI 资源 OnEnd PinrtPreview 在退出预览模式的时候调用这个函数
OnEndPrint Preview	当激活预览模式时调用
OnPrepareDC	在为屏幕显示调用 OnDraw 成员函数或为打印和打印预览调用 OnPrint 成员函数之前调用
OnPrepare Printing	在文档被打印或预览之前调用这个函数；重载这个函数以初始化 Print 对话框
OnPrint	调用这个函数以打印或预览文档的一页
OnUpdate	调用这个函数以通知一个视图，文档已经被修改

Constructors

Cview	构造一个 CView 对象

成员函数

CView::CView

```
CView( );
```

说明

构造一个CView对象。当生成一个新的框架窗口或者要分隔一个窗口时，框架将调用这个构造函数。重载OnInitialUpdate成员函数以在与文档连接之后初始化该视图。

请参阅 CView::OnInitialUpdate

CView::DoPreparePrinting

BOOL DoPreparePrinting(CPrintInfo* *pInfo*);

返回值

如果可以开始打印或打印预览，则返回非零值；否则返回0。

参数

pInfo
指向一个CPrintInfo结构，它描述了当前的打印作业。

说明

在你重载的OnPreparePrinting函数中调用这个函数以激活Print对话框并创建一个打印机设备环境。

这个函数的效果依赖于它是为打印还是打印预览调用的（由 *pInfo* 参数中的 m_bPreview 成员指定）。如果某个文件将要被打印，这个函数将使用 *pInfo* 所指向的 CPrintInfo 结构中的数据来激活 Print 对话框；当用户关闭这个对话框的时候，这个函数根据用户在对话框中指定的设置创建一个打印机设备环境，并将这个设备环境通过 *pInfo* 参数返回。这个设备环境被用于打印文档。

如果文件是要被预览，这个函数使用当前的打印机设置创建一个打印机设备环境；这个设备环境在预览过程中被用来仿真打印机。

请参阅 CPrintInfo, CView::OnPreparePrinting

CView::GetDocument

CDocument* GetDocument( ) const;

返回值

指向与视图相连接的文档的指针。如果这个视图没有与文档相连接，则返回NULL。

说明

调用这个函数以获得视图的文档指针。这使你能够调用文档的成员函数。

请参阅 CDocument

CView::IsSelected

virtual BOOL IsSelected(const CObject* *pDocItem*) const;

返回值

如果指定的文档项目被选中，则返回非零值；否则返回0。

参数

pDocItem

　　指向要测试的文档项目的指针。

说明

由框架调用，检查指定的文档项目是否被选中。这个函数的缺省实现返回FALSE。如果你使用CDocItem对象实现选择，则重载这个函数。如果你的视图中包含OLE项，则必须重载这个函数。

请参阅　CDocItem, COleClientItem

CView::OnActivateFrame

virtual void OnActiveFrame(UINT *nState*, CFrameWnd* *pFrameWnd*);

参数

nstate

　　指定框架窗口是被激活还是反之。它可以是下列值之一：

- WA_INACTIVE　**框架窗口将结束激活状态。**
- WA_ACTIVE　**框架窗口将通过不同于鼠标点击的方式而激活（例如，**

通过键盘接口来选择窗口）。
- WA_CLICKACTIVE 框架窗口将通过鼠标点击而激活。

pFrameWnd
指向要激活的框架窗口的指针。

说明

当包含视图的框架窗口要被激活或结束激活状态时，框架就调用这个函数。如果你希望在与视图相关的框架窗口被激活或结束激活状态的时候进行特殊处理，那么应当重载这个成员函数。例如，当CFormView保存或恢复具有焦点的控件的时候，它就重载了这个函数。

请参阅 CWnd::OnActivate, CFormView

CView::OnActivateView

virtual void OnActivateView(BOOL *bActivate*, CView* *pActivateView*, CView* *pDeactiveView*);

参数

bActivate
指明该视图是要被激活还是要结束激活状态。

pActivateView

指向要激活的视图的指针。

pDeactiveView

指向要结束激活状态的视图对象的指针。

说明

当视图被激活或结束激活状态的时候，框架调用这个函数。这个函数的缺省实现将焦点设置到要激活的视图中。如果你希望在视图被激活或结束激活状态的时候进行特殊处理，那么应当重载这个函数。例如，如果你希望提供特别的视觉效果，使活动的视图与非活动的视图能有区别，你应当检查bActivate的值，并根据结果相应地更新视图的外观。

如果应用程序的主框架窗口被激活，而它的活动视图没有发生变化，那么pActivateView 和 pDeactiveView 参数指向同一个视图——例如，焦点是从另一个应用程序传送到这个应用程序，而不是在应用程序内部从一个视图传送到另一个视图或是在 MDI 的子窗口之间传递。

当对一个视图调用 CFrameWnd::SetActiveView ，而这个视图与 CFrameWnd::GetActive-View 返回的视图不一致的时候，这些参数也是不同的。这通常在分隔窗口中发生。

请参阅 CWnd::OnActivate, CFrameWnd::SetActiveView, CFrameWnd::GetActiveView

CView::OnBeginPrinting

virtual void OnBeginPrinting(CDC* *pDC*, CPrintInfo* *pInfo*);

参数

pDC
　　指向打印机设备环境。

pInfo
　　指向一个CPrintInfo结构，该结构描述了当前的打印作业。

说明

框架在开始打印或打印预览作业之前，而在OnPreparePrinting被调用之后调用这个函数。这个函数的缺省实现不做任何操作。重载这个函数以分配打印所需的GDI资源，如画笔或字体。在OnPrint成员函数内部将这些GDI资源选入设备环境。如果你使用同一个视图对象来执行打印和打印预览，那么对每种显示所需的GDI资源使用不同的变量；这样使你能够在打印的时候更新屏幕。
你还可以使用这个函数来实现依赖于打印机设备环境的属性的初始化工作。例如，打印文档所需的页数依赖于用户在Print对话框中指定的设置（例如页长度等）。在这种情况下，你不能在OnPreparePrinting成员函数中指定文档的长度，而在通常情况下你可以这么做；你必须等待，直到已经根据对话框设置创建了打印机设备环境。OnBeginPrinting是使你能够访问代表了打印机设备环境的CDC

对象的第一个重载函数，因此你可以在这个函数内部设置文档的长度。注意如果这时没有直到文档的长度，在预览的过程中将不会显示滚动条。

请参阅 CView::OnEndPrinting, CView::OnPreparePrinting, CView::OnPrint

CView::OnDragEnter

virtual DROPEFFECT OnDragEnter(COleDataObject* *pDataObject*, DWORD *dwKeyState*, CPoint *point*);

返回值

DROPEFFECT枚举类型中的一个值，指明了如果用户在这个位置放下对象时将会发生的放下动作类型。放下动作的类型通常与dwKeyState指明的当前按键状态有关。标准的从按键状态到DROPEFFECT值的映射如下：

- DROPEFFECT_NONE 数据对象不能在这个窗口中放下。
- DROPEFFECT_LINK 对应MK_CONTROL|MK_SHIFT，在对象及其服务器之间建立连接。
- DROPEFFECT_COPY 对应MK_CONTROL，创建放下的对象的一个拷贝。
- DROPEFFECT_MOVE 对应MK_ALT，创建放下的对象的一个拷贝并删除原来的对象。当视图能够接收这个数据对象时，通常这是缺省的下放效果。

有关的更多信息参见 MFC 高级概念示例 OCLIENT。

参数

pDataObject
　　指向将要被拖入视图的下放区域的COleDataObject对象的指针。

dwKeyState
　　包含了特殊键的状态。这是下列值的组合：MK_CONTROL, MF_SHIFT, MK_ALT, MK_LBUTTON, MK_MBUTTON和MK_RBUTTON。

point
　　相对于视图的客户区域的当前鼠标位置。

说明

当鼠标第一次进入下放目标窗口的非滚动区域时，框架就调用这个函数。缺省的实现不做任何操作，并返回DROPEFFECT_NONE。

将来可能产生的对 OnDragOver 成员函数的调用重载这个函数。任何对数据对象要求的数据都在此时获得，以备随后 OnDragOver 成员函数的使用。此时视图也应当被更新，以给用户视觉反馈。有关的更多信息参见“Visual C++程序员指南”中的文章“拖放：实现下放目标”。

请参阅 CView::OnDragOver, CView::OnDrop, CView::OnDropEx, CView::OnDragLeave, COleDropTarget::OnDragEnter

CView::OnDragLeave

virtual void OnDragLeave( );

说明

当鼠标移出窗口的有效下放位置时，框架调用这个函数。

如果当前的视图需要清除在 OnDragEnter **或** OnDragOver **函数中产生的动作，例如清除对象被拖动或下放时用户的反馈，则应当重载这个函数。**

请参阅 CView::OnDragEnter, CView::OnDragOver, CView::OnScroll, COleDropTarget::On- DragLeave

CView::OnDragOver

virtual DROPEFFECT OnDragOver(COleDataObject* *pDataObject*, DWORD *dwKeyState*, CPoint *point*);

返回值

DROPEFFECT**枚举类型中的一个值，它指明了当用户在当前位置下放对象时会发生的下放动作类型。下放动作的类型依赖于**dwKeyState**指明的当前按键状态。从按键状态到**DROPEFFECT**枚举值的标准映射为：**

- DROPEFFECT_NONE 数据对象不能在这个窗口中放下。
- DROPEFFECT_LINK 对应MK_CONTROL|MK_SHIFT，在对象及其服务器之间建立连接。
- DROPEFFECT_COPY 对应MK_CONTROL，创建下放的对象的一个拷贝。
- DROPEFFECT_MOVE 对应MK_ALT，创建下放的对象的一个拷贝并删除原来的对象。当视图能够接收这个数据对象时，通常这是缺省的下放效果。

有关的更多信息参见 MFC 高级概念示例 OCLIENT。

参数

pDataObject

指向要被拖过下放目标的COleDataObject对象的指针。

dwKeyState

包含了特殊键的当前状态。它是下列值的组合：MK_CONTROL, MK_SHIFT, MK_ALT, MK_LBUTTON, MK_MBUTTON 和 MK_RBUTTON。

point

相对于视图的客户区域的当前鼠标位置。

说明

当鼠标从下放目标窗口上方经过时，框架窗口调用这个函数。缺省的实行不做任何操作，并返回DROPEFFECT_NONE。

重载这个函数以在拖放操作的过程中向用户提供视觉反馈。由于这个函数是连续调用的，因此这个函数内部的代码必须尽可能地优化。有关的更多信息参见“Visual C++**程序员指南**”**中的文章**“**拖放：实现下放目标**”。

请参阅 CView::OnDragEnter, CView::OnDrop, CView::OnDropEx, CView::OnDragLeave, COleDropTarget::OnDragOver

CView::OnDragScroll

vitual DROPEFFECT OnDragScroll(DWORD *dwKeyState*, CPoint *point*);

返回值

DROPEFFECT**枚举类型中的一个值，它指明了当用户在当前位置下放对象时会发生的下放动作类型。下放动作的类型依赖于**dwKeyState**指明的当前按键状态。从按键状态到**DROPEFFECT**枚举值的标准映射为：**

- DROPEFFECT_NONE **数据对象不能在这个窗口中放下。**
- DROPEFFECT_LINK **对应**MK_CONTROL|MK_SHIFT**，在对象及其服务器之间建立连接。**

- DROPEFFECT_COPY 对应MK_CONTROL，创建下放的对象的一个拷贝。
- DROPEFFECT_MOVE 对应MK_ALT，创建下放的对象的一个拷贝并删除原来的对象。
- DROPEFFECT_SCROLL 指明在目标视图中将要或正在发生一个下放滚动操作。

有关的更多信息参见MFC高级概念示例OCLIENT。

参数

dwKeyState
包含了特殊键的当前状态。它是下列值的组合：MK_CONTROL, MK_SHIFT, MK_ALT, MK_LBUTTON, MK_MBUTTON和MK_RBUTTON。

point
相对于视图的客户区域的当前鼠标位置。

说明

框架在调用OnDragOver和OnDragEnter之前调用这个函数以确定该点是否位于滚动区域。如果你想要为这个时间提供特别的动作，则可以重载这个函数。当鼠标被移进窗口边界内的滚动区域时，该函数的缺省实现自动滚动窗口。有关

的更多信息参见“Visual C++程序员指南”中的文章“拖放：实现下放目标”。

请参阅 CView::OnDragEnter, CView::OnDragOver, CView::OnDrop, CView::OnDragLeave,COleDropTarget::OnDragScroll

CView::OnDraw

virtual void OnDraw(CDC* *pDC*=0);

参数

pDC

指向设备环境的指针，该环境被用于画出文档的图像。

说明

框架调用这个函数以画出文档的图像。框架调用这个函数来实现屏幕显示，打印以及打印预览，在各种情况下，它传递不同的参数。没有缺省的实现。

为了显示你的文档的视图，必须重载这个函数。你可以通过 pDC 指向的 CDC 对象来调用图形设备接口（GDI）。你可以在显示之前将 GDI 资源，如画笔和字体等选入设备环境，在显示完成之后，再把它们选出去。通常你的绘图代码是与设备无关的，这意味着，它不需要有关显示图形的设备类型的信息。

如果要优化绘图操作，调用设备环境的 RectVisible 成员函数以确定一个给定的

矩形是否需要重画。如果你需要区别普通的屏幕显示和打印，那么应调用设备环境的 IsPrinting **函数。**

请参阅 CDC::IsPrinting, CDC::RectVisible, CView::OnPrint, CWnd::OnCreate, CWnd::OnDestroy, CWnd::PostNcDestroy

CView::OnDrop

virtual BOOL OnDrop(COleDataObject* *pDataObject*, DROPEFFECT *dropEffect*, Cpoint *point*);

返回值

如果成功地下放，则返回非零值，否则返回0**。**

参数

pDataObject
指向将要被放入下放目标的COleDataObject**对象。**

dropEffect
用户要求的下放效果。

- DROPEFFECT_COPY **创建被下放的数据对象的一个拷贝。**
- DROPEFFECT_MOVE **将数据对象移动到当前鼠标位置。**

- DROPEFFECT_LINK **在数据对象和它的服务器之间创建连接。**

point

相对于视图的客户区域的当前鼠标位置。

说明

当用户在有效的下放目标上方放开一个数据对象时，框架调用这个函数。缺省的实现不做任何操作，并且返回FALSE。

重载这个函数以实现视图的客户区内的 OLE **下放效果。可以通过** pDataObject **来检查数据对象的剪贴板数据格式和指定点的下放数据。**

注意 如果在这个视类中存在 OnDropEx 的重载函数，则框架不会调用这个函数。

请参阅 CView::OnDragEnter, CView::OnDragOver, CView::OnDropEx, CView::OnDragLeave, COleDropTarget::OnDrop

CView::OnDropEx

virtual DROPEFFECT OnDropEx(COleDataObject* *pDataObject*, DROPEFFECT *dropDefault*, DROPEFFECT *dropList*, Cpoint *point*);

返回值

在point所指定的位置进行的下放动作的效果。它必须是dropEffectList所指定的值之一。在说明部分讨论了下放效果。

参数

pDataObject
指向将要放入下放目标的COleDataObject对象的指针。

dropDefault
用户根据当前按键状态选择的下放操作的效果。它可能是DROPEFFECT_NONE。在说明部分讨论了下放效果。

dropList
下放源所支持的下放效果的列表。下放效果值可以通过位或操作符（|）组合起来。在说明部分讨论了下放效果。

point
相对于视图的客户区的当前鼠标位置。

说明

当用户在有效的下放目标上方放开一个数据对象时，框架调用这个函数。缺省的实现不做任何操作，并且返回一个空值（-1）以指明框架必须调用OnDrop处

理函数。

如果要实现鼠标右键的拖放效果，则应重载这个函数。通常在释放右键的时候，右键拖放操作会显示一个选择菜单。

你重载的 OnDropEx 必须查询鼠标右键的状态。你可以调用 GetKeyState 或者从 OnDragEnter 处理函数中保存鼠标右键的状态。

- 如果鼠标右键被按下，你的重载函数必须显示一个弹出菜单，提供下放源支持的下放效果。
  - 检查dropList以确定下放源所支持的下放效果。在弹出菜单中仅使这些动作有效。
- 使用SetMenuDefaultItem以根据dropDefault来设置缺省的动作。
- 最后，执行用户通过弹出菜单所选择的动作。
- 如果鼠标右键没有按下，你的重载函数应当将它当作标准的下放请求来处理。使用dropDefault中指定的下放效果。另外，你的重载函数业可以返回空值（-1）以指明将由OnDrop来处理这个下放操作。

使用 pDataObject 来检查 COleDataObject 的剪贴板数据格式和指定点的下放数据。
下放效果描述了与下放操作相关的动作。参看下面的下放效果列表：

- DROPEFFECT_NONE　不允许下放操作。
- DROPEFFECT_COPY　将执行拷贝动作。
- DROPEFFECT_MOVE　将执行移动动作。

- DROPEFFECT_LINK 将建立被下放数据和原始数据的连接。
- DROPEFFECT_SCROLL 指明在下放目标中将要或正在发生下放滚动操作。

有关设置缺省菜单命令的更多信息参见 Win32 文档中的 SetMenuDefaultItem 以及本书中的 CMenu::GetSafeHmenu。

请参阅 CView::OnDragEnter, CView::OnDragOver, CView::OnDrop, CView::OnDragLeave, COleDropTarget::OnDropEx

CView::OnEndPrinting

virtual void OnEndPrinting(CDC* *pDC*, CPrintInfo* *pInfo*);

参数

pDC

指向打印机设备环境的指针。

pInfo

指向一个CPrintInfo结构，描述了当前的打印作业。

说明

框架在打印或预览文档之后调用这个函数。这个函数的缺省实现不做任何操

作。重载这个函数以释放你在OnBeginPrinting成员函数中分配的GDI资源。

请参阅 CView::OnBeginPrinting

CView::OnEndPrintPreview

virtual void OnEndPrintPreview(CDC* *pDC*, CPrintInfo* *pInfo*, POINT *point*, CPreviewView* *pView*);

参数

pDC

指向打印机设备环境的指针。

pInfo

指向一个CPrintInfo结构，描述了当前的打印作业。

point

指向在预览模式下最后显示的页面中的位置。

pView

指向用于预览的视图对象的指针。

说明

当用户退出打印预览模式时，框架调用这个函数。这个函数的缺省实现调用OnEndPrinting成员函数并将主框架窗口恢复到打印预览开始之前的状态。如果在预览模式结束的时候需要进行一些特殊的处理，则应重载这个函数。例如，如果你想要在从预览模式切换到普通显示模式的时候保持用户在文档中的位置，你可以滚动到pInfo参数指向的CPrintInfo结构中m_nCurPage所指向的页面，point参数所描述的位置。
在你的重载函数中总是要调用基类的OnEndPrintPreview函数，通常是在函数的末尾。

请参阅 CPrintInfo, CView::OnEndPrinting

CView::OnInitialUpdate

virtual void OnInitialUpdate( );

说明

当视图第一次被连接到文档，但是还没有被显示的时候，框架就调用这个函数。这个函数的缺省实现调用OnUpdate成员函数，不给任何提示信息（这意味着将lHint设为0，将pHint参数设为NULL）。如果你想进行一些需要文档信息的一次性初始化工作，那么应当重载这个函数。例如，如果你的应用程序具有固定

大小的文档，你可以利用这个成员函数来根据文档的大小初始化视的滚动范围。如果你的应用程序支持可变大小的文档，则每当文档发生变化时，就使用OnUpdate函数来更新滚动范围。

请参阅　CView::OnUpdate

CView::OnPrepareDC

```
virtual void OnPrepareDC( CDC* pDC, CPrintInfo* pInfo = NULL );
```

参数

pDC

指向设备环境的指针，用于画出文档的图像。

pInfo

指向CPrintInfo结构的指针，如果OnPrepareDC是为打印或打印预览调用的，则该结构描述了当前打印任务，m_nCurPage成员指定了要打印的页数。如果OnPrepareDC是为屏幕显示而调用的，则这个参数为NULL。

说明

在为屏幕显示而调用OnDraw成员函数或者为打印或打印预览每一页而调用OnPrint成员函数之前，框架调用这个函数。如果这个函数是为屏幕显示而调用

的，则这个函数的缺省实现不做任何操作。但是，这个函数在派生类中被重载，例如在CScrollView中，以调整设备环境的属性，因此，在你的重载代码的开始部分，总应该调用基类的实现。

如果这个函数是为打印而调用的，缺省的实现检查保存在 pInfo 参数中的页面信息。如果没有指定文档的长度，OnPrepareDC 假定文档只有一页，并且在打印完一页以后停止打印循环。这个函数通过将结构的 m_bContinuePrinting 成员设为 FALSE 来结束打印循环。
如果具有以下的原因，则因重载OnPrepareDC：

- 要为指定的页面调整设备环境的属性。例如，如果你想要设置设备环境的映射模式或者其它特征，则应在这个函数中完成这些操作。
- 要实现打印时的分页。通常你应当在打印开始时利用OnPreparePrinting成员函数来指定文档的长度。但是，如果你并不准确地知道文档的长度（例如，当打印数据库中未知数量的记录时），则应重载OnPrepareDC函数，以在打印时检测是否到了文档的末尾。如果已经没有文档需要打印了，将CPrintInfo结构的m_bContinuePrinting成员设为FALSE。
- 要按页发送打印机的转义序列码。要在OnPrepareDC中发送转义序列码，则应调用pDC成员的Escape成员函数。

请参阅　CDC::Escape, CPrintInfo, CView::OnBeginPrinting, CView::OnDraw, CView::OnPre- parePrinting, CView::OnPrint

CView::OnPreparePrinting

virtual BOOL OnPreparePrinting(CPrintInfo* *pInfo*);

返回值

如果要开始打印，则返回非零值；如果取消了打印任务，则返回0。

参数

pInfo
指向一个CPrintInfo结构，该结构描述了当前打印任务。

说明

框架在文档被打印或打印预览之前调用这个函数。缺省的实现不做任何操作。你必须重载这个函数才能够打印或打印预览。调用DoPreparePrinting成员函数，将pInfo参数传递给它，然后返回它的返回值。

DoPreparePrinting 显示 Print 对话框并创建一个打印机设备环境。如果你想要用不同于缺省值的值初始化 Print 对话框，则将这些值赋给 pInfo。例如，如果你知道文档的长度，在调用 DoPreparePrinting 函数之前将这个值传递给 pInfo 的 SetMaxPage 成员函数。这个值将显示在 Print 对话框中范围部分的 To: 框中。

对于打印预览作业，DoPreparePrinting 并不显示 Print 对话框。如果你希望在打

印作业中跳过Print对话框，检查pInfo中的m_bPreview成员为FALSE，在将它传递给DoPrepare- Printing之前把它设为TRUE，然后在把它恢复为FALSE。

如果你想要进行的初始化工作需要访问代表打印机设备环境的CDC对象（例如，如果你在设定文档长度之前需要知道页面的大小），则应重载OnBeginPrinting成员函数。

如果你想要设置pInfo参数的m_nNumPreviewPages或m_strPageDesc成员的值，则应在调用DoPreparePrinting之后进行这些操作。DoPreparePrinting成员函数会把m_nNum- PreviewPages设为在应用程序的.INI文件中找到的值，并将m_strPageDesc设为它的缺省值。

示例

如果你在创建起始文件的时候选择了打印选项，则AppWizard会提供重载的OnPrepare- Printing函数，下面就是例子。除非你想要初始化Print对话框，否则这个重载函数的功能已经足够了：

```
BOOL CMyView::OnPreparePrinting( CPrintInfo *pInfo )
{
return DoPreparePrinting( pInfo );
}
```

请参阅 CPrintInfo, CView::DoPreparePrinting, CView::OnBeginPrinting, CView::OnPrepareDC, CView::OnPrint

CView::OnPrint

virtual void OnPrint(CDC* *pDC*, CPrintInfo* *pInfo*);

参数

pDC
　　指向打印机设备环境的指针。

pInfo
　　指向CPrintInfo结构的指针，该结构描述了当前打印作业。

说明

框架调用这个函数以打印或预览文档的一页。对于要被打印的每一页，框架在调用OnPrepareDC成员函数之后立即调用这个函数。要被打印的页是在pInfo指向的CPrintInfo结构的m_nCurPage成员中指定的。缺省的实现调用OnDraw函数并将打印机设备环境传递给它。

如果具有以下原因，则应重载这个函数：

- 要允许打印多页文档。仅画出与当前要打印的页相对应的文档内容。如果你要OnDraw函数来绘图，你可以调整视图口的原点，这样只有文档的适当的部分才会被打印。
- 要使打印出来的图像与屏幕显示的图像不同（如果你的应用程序不是所见

即所得的）。不应将打印机设备环境传递给OnDraw函数，而是使用设备环境，用没有在屏幕上显示的属性来画出图像。

如果你在打印时需要一些 GDI 资源，而在屏幕显示中没有使用它们，则应在绘图之前将它们选入设备环境，随后把它们选出。这些 GDI 资源必须在 OnBeginPrinting 函数中分配，而在 OnEndPrinting 函数中释放。

- 要实现页眉和页脚。只要你限制 OnDraw 可以打印的区域，你还可以使用 OnDraw 函数来绘图。

注意 pInfo 参数的 m_rectDraw 成员以逻辑单位描述了页面中可以打印的区域。在你重载的OnPrint中不要调用OnPrepareDC，框架在调用OnPrint之前自动调用了OnPrepareDC。

示例

下面是重载的OnPrint函数的基本结构：

```
void CMyView::OnPrint( CDC *pDC, CPrintInfo *pInfo )
{
    // Print headers and/or footers, if desired.
  // Find portion of document corresponding to pInfo->m_nCurPage.
  OnDraw( pDC );
}
```

请参阅 CView::OnBeginPrinting, CView::OnEndPrinting, CView::OnPrepareDC,

CView::OnDraw

CView::OnScroll

virtual BOOL CView::OnScroll(UINT *nScrollCode*, UINT *nPos*, BOOL *bDoScroll* = TRUE);

返回值

如果 bDoScroll **为** TRUE**，并且视图确实被滚动了，则返回非零值；否则返回** 0**。如果** bDoScroll **为** FALSE**，则返回当** bDoScroll **为** TRUE **时应当返回的值，即使你没有作实际的滚动。**

参数

nScrollCode

滚动条代码，指明用户的滚动请求。这个参数由两个部分组成：低字节确定了水平滚动的类型，高字节确定了垂直滚动的类型：

- SB_BOTTOM **滚动到底部**
- SB_LINEDOWN **往下滚动一行**
- SB_LINEUP **往上滚动一行**
- SB_PAGEDOWN **往下滚动一页**

- SB_PAGEUP 往上滚动一页
- SB_THUMBTRACK 将滚动块拖至指定位置。当前的位置由 nPos 指定
- SB_TOP 滚动到顶部

nPos

如果滚动条代码为SB_THUMBTRACK，则包含了当前的滚动块位置；否则没有被使用。根据初始的滚动范围值，*nPos*有可能为负值，并且在必要时应当被强制转换为整数。

bDoScroll

确定你是否需要实际完成指定的滚动动作。如果该值为TRUE，则必须执行滚动操作；如果为FALSE，则不应执行滚动操作。

说明

框架调用这个函数以确定是否能够滚动。

有一种情况是，如果视图接受到一个滚动条消息，则框架调用这个函数并将 bDoScroll 设为 TRUE。在这种情况下，你必须执行实际的滚动。其它的情况是，当某个 OLE 项被拖入下放对象的自动滚动区域，在发生实际的滚动之前，框架调用这个函数并将 bDoScroll 设为 FALSE。在这种情况下，你不应当执行实际的滚动。

请参阅 CView::OnScrollBy, COleClientItem

CView::OnScrollBy

BOOL CView::OnScrollBy(CSize *sizeScroll*, BOOL *bDoScroll* = TRUE);

返回值

如果该视图能被滚动，则返回非零值；否则返回0。

参数

sizeScroll

水平及垂直滚动的像素的数目。

bDoScroll

确定视图的滚动是否发生。如果该值为TRUE**，则发生了滚动；如果为**FALSE**，则没有发生滚动。**

说明

当用户将OLE**项拖到当前视图的边界之外，或者操作水平、垂直滚动条，观察超出文档已显示的部分的内容时，框架就会调用这个函数。缺省的实现不做任何操作。在派生类中，这个函数检查该视图在用户请求的方向上是否可以滚动，然后在必要时更新新的区域。为了执行实际的滚动请求，**CWnd::OnHScroll**和**CWnd::OnVScroll**自动调用这个函数。**

CView::OnUpdate

virtual void OnUpdate(CView* *pSender*, LPARAM *lHint*, CObject* *pHint*);

参数

pSender
　　指向修改了文档的视图，如果需要更新所有的视图，则为NULL。

lHint
　　包含了与修改有关的信息。

pHint
　　指向保存了与修改有关的信息的对象。

说明

框架在视图的文档被修改后调用这个函数；这个函数被CDocument::UpdateAllViews调用的，使视图能够更新它的显示以反映那些变化。它也被OnInitialUpdate的缺省实现所调用。缺省的实现使整个客户区域无效，使得下一次接收到WM_PAINT消息时重画这些区域。如果你只想更新与文档的修改部分对应的区域，则应重载这个函数。为此你必须利用提示参数传递有关修改的信息。

如果要使用 lHint，则应定义特殊的提示值，通常是位掩码或枚举值，并且使文

档传递其中的一个值。要使用 pHint，则应从 CObject 集成一个提示类，并使文档传递提示对象的指针。当你重载 OnUpdate 函数的时候，应使用 CObject::IsKindOf 成员函数来确定提示对象的运行时类型。

通常你不用在 OnUpdate 中直接执行任何绘图操作。相反，确定以设备坐标表示的矩形，描述要更新的区域，将这个矩形传递给 CWnd::InvalidateRect。这会使下一次接收到 WM_PAINT 消息时产生绘图操作。

如果 lHint 为 0，pHint 为 NULL，文档将发送一个一般的更新通知。如果一个视图接收到了一般的更新通知，或者它没有解码出提示，它将会使整个客户区无效。

请参阅 CDocument::UpdateAllViews, CView::OnInitialUpdate, CWnd::Invalidate, CWnd::InvalidateRect

CWaitCursor

CWaitCursor没有基类。

CWaitCursor类提供了显示等待光标的直接方式，当你进行冗长的操作时，它通常显示出一个沙漏。好的Windows编程方式要求你在执行耗用大量时间的操作时显示等待光标。

如果要显示等待光标，仅需在进入执行冗长操作的代码之前定义一个CWaitCursor变量。整个对象的构造函数自动地显示等待光标。

当对象超出作用域时（在定义了CWaitCursor对象的代码块的末尾），它的析构函数将光标设为原来的光标。换句话说，该对象自动执行必要的清除工作。

> 注意　由于它们的构造函数和析构函数的工作方式，CWaitCursor对象总是被定义为局部变量 -- 它们从不被定义为全局变量，也不用new来分配。

如果你执行了可能会使光标改变的操作，比如显示消息框或对话框，则应调用Restore成员函数以恢复光标。即使当前正在显示等待光标也可以调用Restore函数。

显示等待光标的另一种方式是使用 CCmdTarget::BeginWaitCursor 和 CCmdTarget::EndWait Cursor 的组合，可能还有 CCmdTarget::RestoreWaitCursor。但是，CWaitCursor 更易于使用，因为你在完成了冗长的操作之后不必将光标恢复到原来的光标。

> 注意 MFC 使用虚拟函数 CWinApp::DoWaitCursor 来设置和恢复光标。你可以重载这个函数以提供自定义的表现方式。

#include <afxwin.h>

请参阅 CCmdTarget::BeginWaitCursor, CCmdTarget::EndWaitCursor, CCmdTarget::Restore- W aitCursor, CWinApp::DoWaitCursor

CWaitCursor 类成员

Construction/Destruction	
CWaitCursor	构造一个 CWaitCursor 对象并显示等待光标

Operations	
Restore	在光标被改变后恢复等待光标

成员函数

CWaitCursor::CWaitCursor

CWaitCursor( );

说明

如果要显示等待光标，只需在冗长操作的代码之前定义一个CWaitCursor对象。其构造函数自动显示等待光标。

当对象超出作用域（在定义 CWaitCursor 对象的代码块的末尾处），它的析构函数将光标设为原来的光标。换句话说，这个对象自动执行必要的清除操作。

析构函数是在代码块的末尾被调用的（可能是在函数末尾的前面），你可以利用这个特性使等待光标只将你的函数的一部分激活。下面的第二个例子演示了这种技术。

> 注意　由于它们的构造函数和析构函数的工作方式，CWaitCursor 对象总是被定义为局部变量 -- 它们从不被定义为全局变量，也不用 new 来分配。

示例

```
// 下面的例子演示了在冗长操作的过程中显示等待光标的一般方式。
void LengthyFunction( )
{
   // 你有可能在显示等待光标之前显示一个对话框
CWaitCursor wait;      // 显示等待光标
// 执行一些冗长的操作
}  // 析构函数自动清除等待光标
// 这个例子演示了在代码块内部使用 CWaitCursor 对象，
// 因此只有当程序执行冗长的操作时才会显示等待光标。
void ConditionalFunction( )
{
   if ( SomeCondition )
   {
   CWaitCursor wait;      // 仅在本代码块内部才显示等待光标。
   // 执行一些冗长的操作。
   }  // 在这个地方，析构函数清除等待光标。
   else
  {
      // 没有等待光标 -- 只有快速操作
   }
```

}

请参阅 CWaitCursor::Restore, CCmdTarget::BeginWaitCursor, CCmdTarget::EndWaitCursor

CWaitCursor::Restore

void Restore( );

说明

如果想要恢复等待光标，则应在完成操作后调用这个函数，比如显示消息框或对话框，这可能会使等待光标变为其它光标。

如果当前正显示等待光标，也可以调用 Restore 函数。

如果你需要在没有定义 CWaitCursor 对象的函数内部恢复等待光标，则应调用 CCmdTar- get::RestoreW aitCursor。

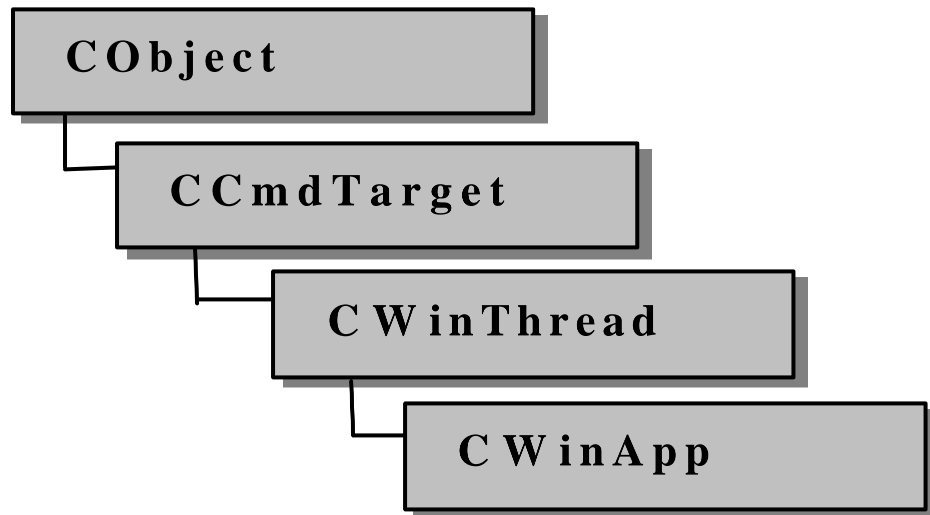
示例

```
// 这个例子演示了改变等待光标的操作
// 在完成改变光标的操作以后，你应当调用
// CWaitCursor::Restore 以恢复光标。
void AnotherLengthyFunction( )
```

```
{
    CWaitCursor wait;        // 显示等待光标
    // 执行一些冗长操作
    // 对话框通常会将光标变为标准的箭头光标。
    CSomeDialog dlg;
    dlg.DoModal( );
    // 为了将光标变回等待光标，必须调用 Restore 函数。
    wait.Restore( );
    // 执行一些冗长操作
    // 析构函数自动清除等待光标
}
// 如果等待光标被创建它的函数所调用的函数改变，
// 你可以调用 CCmdTarget::RestoreWaitCursor 来恢复等待光标。
void CalledFunction()
{
    CSomeDialog dlg;
    dlg.DoModal();
    // 由于 CWinApp 是从 CCmdTarget 继承的，我们可以
    // 使用应用程序对象的指针来调用 CCmdTarget::RestoreWaitCursor。
    AfxGetApp()->RestoreWaitCursor( );
  // 还有其它冗长操作...
}
```

请参阅 CCmdTarget::RestoreWaitCursor

CWinApp



CWinApp是一个基类，你通过它来继承Windows应用程序对象。应用程序对象为你提供了初始化应用程序（以及它的每一个实例）和运行应用程序所需的成员函数。

每个使用微软基础类库的应用程序都只能包含一个从 CWinApp 继承的对象。当 Windows 调用 WinMain 函数时，这个对象在其它 C++全局对象都已经生成并且可用之后才被创建，WinMain 函数是由微软基础类库提供的。将你的CWinApp 对象定义为全局的。

当你从 CWinApp 继承应用程序类的时候，应重载 InitInstance 成员函数以创建应用程序的主窗口对象。

除了 CWinApp 的成员函数以外，微软基础类库还提供了以下全局函数，用于访问你的 CWinApp 对象以及其它全局信息：

- AfxGetApp 获得指向 CWinApp 对象的指针。
- AfxGetInstanceHandle 获得当前应用程序实例的句柄。
- AfxGetResourceHandle 获得应用程序资源的句柄。
- AfxGetAppName 获得一个字符串指针，其中包含了应用程序的名字。另外，如果你拥有一个指向CWinApp对象的指针，可以通过m_pszExename来获得应用程序的名字。

有关 CWinApp 类的更多信息参见“Visual C++ 程序员指南”中的“CWinApp：应用程序类”，其中包括下面的内容：

- AppWizard 生成的 CWinApp 派生代码
- CWinApp在你的应用程序执行顺序中作用
- CWinApp的缺省成员函数实现
- CWinApp的主要可重载函数

#include <afxwin.h>

CWinApp 类成员

Data Members

m_pszAppName	指定了应用程序的名字
m_hInstance	标识了应用程序的当前实例
m_hPrevInstance	在 32 位应用程序中被设为 NULL
m_lpCmdLine	指向一个以 null 结尾的字符串，指定了应用程序的命令行
m_nCmdShow	指定最初如何显示窗口
m_bHelpMode	指明用户是否处于 Help 上下文模式(通常用 SHIFT+F1 激活)
m_pActiveWnd	当一个 OLE 服务器是现场可激活时，它指向容器应用程序的主窗口
m_pszExeName	应用程序的模块名字
m_pszHelpFilePath	应用程序的帮助文件的路径
m_pszProfileName	应用程序的.INI 文件名
m_pszRegistryKey	用于确定保存应用程序主要设置的完整的注册表键

Construction

CWinApp	构造一个 CWinApp 对象

Operations	
LoadCursor	载入光标资源
LoadStandardCursor	载入 WINDOWS.H 中 IDC_常量所指定的 Windows 预定义光标
LoadOEMCursor	载入 WINDOWS.H 中 OCR_常量所指定的 Windows OEM 预定义光标
LoadIcon	载入图标资源
LoadStandardIcon	载入 WINDOWS.H 中 IDI_常量所指定的 Windows 预定义图标
LoadOEMIcon	载入 WINDOWS.H 中 OIC_常量所指定的 Windows OEM 预定义图标
RunAutomated	检查应用程序的命令行是否指定/Automation 选项。已不用。应当在调用 ParseCommandLine 之后使用 CCommandLineInfo::m_bRunEmbedded 中的值
RunEmbedded	检查应用程序的命令行是否指定/Embedding 选项。已不用。应当在调用 ParseCommandLine 之后使用 CCommandLineInfo::m_bRunEmbedded 中的值
ParseCommandLine	解析命令行中的每个参数和标志
ProcessShellCommand	处理命令行参数和标志
GetProfileInt	从应用程序的.INI 文件的一个入口中获取一个整数

续表

WriteProfileInt	将一个整数写到应用程序的.INI文件的入口
GetProfileString	从应用程序的.INI 文件的一个入口中获取一个字符串
WriteProfileString	将一个字符串写到应用程序的.INI文件的入口
AddDocTemplate	将一个文档模板加到应用程序的可用文档模板列表中
GetFirstDocTemplatePosition	获取第一个文档模板的位置
GetNextDocTemplate	获得文档模板的位置。可以递归调用
OpenDocumentFile	由框架调用，用以从文件打开一个文档
AddToRecentFileList	将一个文件名加入最近使用（MRU）的文件列表
SelectPrinter	选择先前由用户在打印对话框中指定的打印机
CreatePrinterDC	创建一个打印机设备环境
GetPrinterDeviceDefaults	获得缺省的打印机设备

Overridables	
InitInstance	可被重载以执行 Windows 的实例初始化，比如创建窗口对象
Run	运行缺省的消息循环。可被重载以定制消息循环
OnIdle	可被重载以执行应用程序指定的空闲时处理

续表

ExitInstance	可被重载以在应用程序结束时执行清除操作
HideApplication	在关闭所有的文档之前隐藏应用程序
CloseAllDocuments	关闭所有打开的文档
PreTranslateMessage	在消息被分派到 Windows函数::TranslateMessage 和::DispatchMessage 之前过滤消息
SaveAllModified	提示用户保存所有改变了的文档
DoMessageBox	为应用程序实现 AfxMessageBox
ProcessMessageFilter	在消息到达应用程序之前截取特定的消息
ProcessWndProcException	截取应用程序的消息和命令处理函数抛出的未被处理的异常
DoWaitCursor	打开或关闭等待光标
OnDDECommand	框架调用这个函数以响应动态数据交换（DDE）执行命令
WinHelp	调用 Windows 的 WinHelp 函数

Initialization	
LoadStdProfileSettings	载入标准的.INT 文件设置并允许 MRU 文件列表特性
SetDialogBkColor	设置对话框和消息框的缺省背景颜色
SetRegistryKey	使应用程序的设置保存在注册表中，而不是.INI 文件中

续表

EnableShellOpen	允许用户通过 Windows 的文件管理器打开数据文件
RegisterShellFileTypes	在 Windows 的文件管理器中注册所有的应用程序文档类型
Enable3dControls	使控件具有三维外观
Enable3dControlsStatic	使控件具有三维外观

Command Handlers	
OnFileNew	实现 ID_FILE_NEW 命令
OnFileOpen	实现 ID_FILE_NEW 命令
OnFilePrintSetup	实现 ID_FILE_PRINT_SETUP 命令
OnContextHelp	处理应用程序内的 SHIFT+F1 命令
OnHelp	处理应用程序内的 F1 帮助命令（使用当前的上下文）
OnHelpIndex	处理 ID_HELP_INDEX 命令，提供缺省的帮助主题
OnHelpFinder	处理 ID_HELP_FINDER 和 ID_DEFAULT_HELP 命令
OnHelpUsing	处理 ID_HELP_USING 命令

成员函数

CWinApp::AddDocTemplate

void AddDocTemplate(CDocTemplate* *pTemplate*);

参数

pTemplate

指向要增加的CDocTemplate的指针。

说明

调用这个成员函数，将文档模板加入应用程序维护的可用文档模板列表中。你可以在调用RegisterShellFileTypes之前加入所有的文档模板。

示例

```
BOOL CMyApp::InitInstance()
{
// ...
// 下面的代码是在你选择MDI（多文档界面）选项时
```

```
// AppWizard 为你生成的。
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
      IDR_MYTYPE,
      RUNTIME_CLASS(CMyDoc),
      RUNTIME_CLASS(CMDIChildWnd),          // 标准的 MDI 子框架
      RUNTIME_CLASS(CMyView));
AddDocTemplate(pDocTemplate);
// ...
}
```

请参阅 CWinApp::RegisterShellFileTypes, CMultiDocTemplate, CSingleDocTemplate

CWinApp::AddToRecentFileList

virtual void AddToRecentFileList(LPCTSTR *lpszPathName*);

参数

lpszPathName

文件的路径。

说明

调用这个成员函数以把lpszPathName**加入**MRU**文件列表。你必须在使用这个成员函数之前调用**LoadStdProfileSetting**成员函数以载入当前的**MRU**文件列表。**

当框架打开一个文件或者执行 Save As **命令用新名字保存文件时，它就调用这个成员函数。**

示例

```
// 这个例子将路径名 c:\temp\test.doc 加入
// File菜单中的最近使用（MRU）文件列表
AfxGetApp()->AddToRecentFileList("c:\\temp\\test.doc");
```

请参阅 CWinApp::LoadStdProfileSettings

CWinApp::CloseAllDocuments

void CloseAllDocuments(BOOL *bEndSession*);

参数

bEndSession

指定Windows会话是否要结束。如果为TRUE，则会话将结束；否则为FALSE。

说明

调用这个函数以在退出之前关闭所有打开的文档。在调用CloseAllDocuments之前调用HideApplication。

请参阅 CWinApp::SaveAllModified, CWinApp::HideApplication

CWinApp::CreatePrinterDC

BOOL CreatePrinterDC(CDC& dc);

返回值

如果成功地创建了打印机设备环境，则返回非零值；否则返回0。

参数

dc

对打印机设备环境的引用。

说明

调用该成员函数从选定的打印机中创建打印机设备上下文（DC）。CreatePrinterDC初始化通过引用传递过来的设备上下文，因此，你可以使用该设备上下文进行打印。

如果该函数调用成功，在你打印完毕之后，必须销毁该设备上下文。你可以让CDC对象的析构器去做这件事，也可以显式地调用CDC::DeleteDC。

请参阅 CWinApp::SelectPrinter

CWinApp::CWinApp

CWinApp(LPCTSTR *lpszAppName* = NULL);

参数

lpszAppName

一个以null结尾的字符串，其中包含了Windows使用的应用程序的名字。如果没有提供这个参数，或者其值为NULL，CWinApp使用资源字符串AFX_IDS_APP_TITLE或可执行文件的文件名。

说明

构造一个CWinApp对象并将lpszAppName传递给它，当作应用程序的名字保存。你必须创建一个CWinApp派生类的全局对象。在你的应用程序中只能有一个CWinApp对象。构造函数保存了执行CWinApp对象的指针，因此WinMain可以调用对象的成员函数以初始化并运行应用程序。

CWinApp::DoMessageBox

virtual int DoMessageBox(LPCTSTR *lpszPrompt*, UINT *nType*, UINT *nIDPrompt*);

返回值

返回与AfxMessageBox相同的值。

参数

lpszPrompt
    消息框中文本的地址。

nType
    消息框的风格。

nIDPrompt

帮助上下文字符串的索引。

说明

框架调用这个成员函数来实现全局函数AfxMessageBox中的消息框。

不要用这个成员函数来打开消息框，应该使用 AfxMessageBox。
重载这个函数可以在你的应用程序的范围内自定义AfxMessageBox的处理。

请参阅　AfxMessageBox, ::MessageBox

CWinApp::DoWaitCursor

virtual void DoWaitCursor(int *nCode*);

参数

nCode
　　如果这个参数为1，则出现等待光标。如果为0，则恢复等待光标，但不增加引用计数。如果为-1，则结束等待光标。

说明

这个成员函数被框架调用，用以实现CWaitCursor，CCmdTarget::BeginWaitCursor，CCmdTarget::EndWaitCursor 和

CCmdTarget::RestoreWaitCursor。缺省的实现提供一个沙漏光标。DoWaitCursor 维护一个引用计数。当它为正时，将显示沙漏光标。

尽管通常你不用直接调用 DoWaitCursor 函数，你也可以重载这个成员函数以改变等待光标或者在显示等待光标时加入附加的处理。

如果需要更简单有效的实现等待光标的方式，使用 CWaitCursor。

请参阅　CCmdTarget::BeginWaitCursor, CCmdTarget::EndW aitCursor, CCmdTarget::RestoreWaitCursor, CWaitCursor

CWinApp::Enable3dControls

```
BOOL Enable3dControls( );
BOOL Enable3dControlsStatic( );
```

注意　在这个部分同时描述了 Enable3dControls 和 Enable3dControlsStatic。

返回值

如果成功地载入了CTL3D32.DLL，则为TRUE；否则为FALSE。
如果操作系统支持控件的三维外观，则这个函数将返回FALSE。

说明

在你重载的InitInstance成员函数内调用这些成员函数以使对话框和窗口的控件能够具有三维外观。这些成员函数载入CTL3D32.DLL并向它注册应用程序。如果你调用了Enable3d- Controls或Enable3dControlsStatic，你不需要调用SetDialogBkColor成员函数。

在与 MFC DLL 连接时，必须使用 Enable3dControls。当与 MFC 库进行静态连接时，必须使用 Enable3dControlsStatic。

> 仅在专业版和企业版中才具有的特征　只有 Visual C++的专业版和企业版才支持与 MFC 的静态连接。有关的更多信息参见“ Visual C++ ”。

MFC 自动为下列的窗口类提供 3D 控件效果：

- CDialog
- CDialogBar
- CFormView
- CPropertyPage
- CPropertySheet
- CControlBar
- CToolBar

如果你希望具有 3D 外观的控件所在窗口属于上述类，那你只需调用 Enable3dControls 或 Enable3dControlsStatic。如果你希望为基于其它类的窗口中的控件提供 3D 外观，则必须直接调用 CTL3D32 的 API 函数。

示例

```
#ifdef _AFXDLL
   Enable3dControls( );      // 调用 Enable3dControls
#else
   Enable3dControlsStatic( );
// 调用 Enable3dControlsStatic
#endif
```

请参阅 CWinApp::InitInstance, CWinApp::SetDialogBkColor

CWinApp::EnableShellOpen

void EnableShellOpen( );

说明

通常在你重载的InitInstance**函数内调用这个函数，使你的应用程序的用户能够通过在**Windows**的文件管理器内双击文件的方式打开数据文件。与这个函数一起调用**RegisterShellFileTypes**成员函数，或者随应用程序提供一个**.REG**文件，用于手动注册文档类型。**

示例

```
BOOL CMyApp::InitInstance()
```

```
{
// ...
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_MYTYPE,
    RUNTIME_CLASS(CMyDoc),
    RUNTIME_CLASS(CMDIChildWnd),              // 标准的 MDI 子框架
    RUNTIME_CLASS(CMyView));
AddDocTemplate(pDocTemplate);
// 创建 MDI 的主框架窗口
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
// 保存指向主框架窗口的指针。
// 这是框架获得主框架窗口类型的唯一方式。
m_pMainWnd = pMainFrame;
// 打开文件管理器的拖/放和 DDE 打开特性。
EnableShellOpen();
RegisterShellFileTypes();
// ...
// 根据应用程序启动时传递的 nCmdShow 参数显示主窗口
pMainFrame->ShowWindow(m_nCmdShow);
```

```
pMainFrame->UpdateWindow();
// ...
}
```

请参阅 CWinApp::OnDDECommand, CWinApp::RegisterShellFileTypes

CWinApp::ExitInstance

```
virtual int ExitInstance( );
```

返回值

应用程序的退出码；0表示没有错误，大于0的值表示有错误。这个值被用作WinMain的返回值。

说明

框架在Run成员函数的内部调用这个函数以退出应用程序的实例。

不能在其它的任何地方调用这个成员函数，只能在Run成员函数内部调用。

这个函数的缺省实现将框架的选项写入应用程序的.INI文件。

重载这个函数以在应用程序退出的时候执行一些清除操作。

请参阅 CWinApp::Run, CWinApp::InitInstance

CWinApp::GetFirstDocTemplatePosition

POSITION GetFirstDocTemplatePosition( ) const;

返回值

一个POSITION值，可以被用于反复或获取对象指针。如果这个列表为空则返回NULL。

说明

获得应用程序的第一个文档模板的位置。使用调用GetNextDocTemplate时返回的POSITION值来获得第一个CDocTemplate对象。

请参阅 CWinApp::AddDocTemplate, CWinApp::GetNextDocTemplate

CWinApp::GetNextDocTemplate

CDocTemplate* GetNextDocTemplate(POSITION& *pos*) const;

返回值

指向CDocTemplate对象的指针。

参数

pos

对一个POSITION值的引用，该值是上一次对GetNextDocTemplate或GetFirstDoc- Template的调用所返回的。这一次调用将这个值更新为下一个位置。

说明

获得pos所标识的文档模板，然后将pos设置为POSITION值。如果你通过对GetFirstDocTemplatePosition的调用建立了初始的位置，你可以在一个向前的循环中使用GetNextDocTemplate。

你必须确保 POSITION 值是有效的。如果它无效，那么微软基础类库的调试版本将引起断言。

如果获得的文档模板是最后一个有效模板，则新的 pos 值将被设为 NULL。

请参阅　CWinApp::AddDocTemplate, CWinApp::GetFirstDocTemplatePosition

CWinApp::GetPrinterDeviceDefaults

BOOL GetPrinterDeviceDefaults(PRINTDLG* *pPrintDlg*);

返回值

如果成功，则为非零值；否则为0。

参数

pPrintDlg

指向PRINTDLG结构的指针。

说明

调用这个成员函数来为打印准备打印机设备环境。必要时从Windows的.INI文件中获得缺省的打印机，或者使用用户在Print Setup中设置的打印机配置。

请参阅 CPrintDialog

CWinApp::GetProfileInt

UINT GetProfileInt(LPCTSTR *lpszSection*, LPCTSTR *lpszEntry*, int *nDefault*);

返回值

如果这个函数执行成功，则返回指定入口下的字符串的整数值。如果函数没有找到入口，则返回值为nDefault的值。如果与指定入口对应的值不是整数，则

返回值为0。

对于.INI 文件中的值，这个函数支持十六进制符号。当你获得一个带符号整数时，你必须将其值强制转换为整数值。

参数

lpszSection
指向一个以null结尾的字符串，指定了包含入口的部分。

lpszEntry
指向一个以null结尾的字符串，包含了要获取值的入口。

nDefault
指定了当框架找不到入口时的缺省返回值。这个值可以是介于0和65535之间的无符号数，也可以是介于-32768和32767之间的带符号值。

说明

调用这个函数以获得应用程序的注册表或.INI文件中指定部分的入口中的整数值。

这些入口按照如下方式保存：

- Windows NT　该值保存在注册表中
- Windows 3.X　该值保存在WIN.INI文件中。
- Windows 95　该值保存在WIN.INI的缓冲版本中。

这个函数对大小写不敏感，因此 lpszSection 和 lpszEntry 参数中的字符串在大小写状态上可以是不同的。

请参阅

CWinApp::GetProfileString, CWinApp::WriteProfileInt, ::GetPrivateProfileInt

CWinApp::GetProfileString

CString GetProfileString(LPCTSTR *lpszSection*, LPCTSTR *lpszEntry*, LPCTSTR *lpszDefault* = NULL);

返回值

返回值是应用程序的.INI文件中的字符串，如果找不到该字符串，则为lpszDefault。框架支持的字符串最大长度为_MAX_PATH。如果lpszDefault为NULL，则返回值是一个空字符串。

参数

lpszSection
指向一个以null结尾的字符串，指定了包含入口的部分。

lpszEntry
指向一个以null结尾的字符串，其中包含了要获取字符串的入口。这个值

不能为NULL。

lpszDefault
指向给定入口的缺省字符串值，当初始化文件中找不到入口时使用该值。

说明

调用这个函数以获得与应用程序的注册表或.INI文件中指定部分的入口相关的字符串。
这些入口按照如下方式保存：

- Windows NT 该值保存在注册表中
- Windows 3.X 该值保存在WIN.INI文件中
- Windows 95 该值保存在WIN.INI的缓冲版本中

示例

```
CString strSection          = "My Section";
CString strStringItem       = "My String Item";
CString strIntItem          = "My Int Item";
CWinApp* pApp = AfxGetApp();
pApp->WriteProfileString(strSection, strStringItem, "test");
CString strValue;
strValue = pApp->GetProfileString(strSection, strStringItem);
```

```
ASSERT(strValue == "test");
pApp->WriteProfileInt(strSection, strIntItem, 1234);
int nValue;
nValue = pApp->GetProfileInt(strSection, strIntItem, 0);
ASSERT(nValue == 1234);
```

请参阅

CWinApp::GetProfileInt, CWinApp::WriteProfileString, ::GetPrivateProfileString

CWinApp::HideApplication

void HideApplication( );

说明

调用这个成员函数以在关闭打开的文档之前隐藏应用程序。

请参阅 CWinApp::CloseAllDocuments

CWinApp::InitInstance

virtual BOOL InitInstance( );

返回值

如果初始化成功，则返回非零值；否则返回0。

说明

Windows允许在同一时刻运行程序的几份拷贝。在概念上，应用程序的初始化可以被分为两个部分：一次性的应用程序初始化工作，这些在应用程序第一次运行时完成，以及示例的初始化工作，每次运行程序的一个拷贝时都会执行这些操作，包括第一次运行时。框架中WinMain的实现调用这个函数。

重载InitInstance以初始化在Windows下运行的应用程序的每个新实例。通常，你重载InitInstance以构造主窗口对象并设置CWinThread::m_pMainWnd数据成员，使其指向这个窗口。有关重载这个成员函数的更多信息参见“Visual C++程序员指南”中的“CWinApp：应用程序类”。

示例

```
// AppWizard 根据你选择的选项实现重载的 InitInstance 函数。
// 例如，对于下面由 AppWizard 创建的代码，选择了单文档界面（SDI）选项。
// 你可以在 AppWizard 创建的代码中加入其它的每个实例都执行的初始化代码。
BOOL CMyApp::InitInstance()
{
```

```
// 标准的初始化工作
// 如果你没有使用这些特性，并且希望减小最终可执行程序的大小，
// 你应当从下面的初始化例程中移去不必要的代码。
SetDialogBkColor();            // 将对话框的背景色设为灰色。
LoadStdProfileSettings();  // 载入标准的 INI 文件选项（包括 MRU）
// 注册应用程序的文档模板。文档模板
//被用作文档、框架窗口和视图之间的联系。
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CMyDoc),
    RUNTIME_CLASS(CMainFrame),        // SDI 的主框架窗口
    RUNTIME_CLASS(CMyView));
AddDocTemplate(pDocTemplate);
// 创建一个新（空的）文档
OnFileNew();
if (m_lpCmdLine[0] != '\0')
{
    // 任务：在这里加入命令行处理代码
}
return TRUE;
}
```

CWinApp::LoadCursor

HCURSOR LoadCursor(LPCTSTR *lpszResourceName*) const;
HCURSOR LoadCursor(UINT *nIDResource*) const;

返回值

如果成功，则返回光标的句柄；否则返回NULL。

参数

lpszResourceName

指向一个以null**结尾的字符串，其中包含了光标资源的名字。你可以在这个参数中使用**CString**对象。**

nIDResource

光标资源的ID。

说明

载入当前可执行文件中以lpszResourceName**为名或**nIDResource**指定的光标资源。仅当光标以前没有被载入内存时，**LoadCursor**才会将它载入；否则，它获得现存资源的句柄。**

如果要访问预定义的 Windows **光标，则使用** LoadStandardCursor **或**

LoadOEMCursor。

示例

```
HCURSOR hCursor;
// 载入最初用图形编辑器创建并赋给ID值IDC_MYCURSOR的光标资源
hCursor = AfxGetApp()->LoadCursor(IDC_MYCURSOR);
```

请参阅 CWinApp::LoadStandardCursor, CWinApp::LoadOEMCursor, ::LoadCursor

CWinApp::LoadIcon

HICON LoadIcon(LPCTSTR *lpszResourceName*) const;
HICON LoadIcon(UINT *nIDResource*) const;

返回值

如果成功，则返回图标的句柄；否则返回NULL。

参数

lpszResourceName

指向一个以null**结尾的字符串，其中包含了图标资源的名字。你可以在这**

个参数中使用CString对象。

nIDResource

图标资源的ID。

说明

载入可执行文件中以lpszResourceName为名或是nIDResource指定的图标资源。仅当以前图标没有被载入内存时才将它载入；否则它获得现存资源的句柄。

你可以使用LoadStandardIcon或LoadOEMIcon成员函数来访问预定义的Windows图标。

注意 这个成员函数调用Win32 API函数LoadIcon，它仅能载入大小在SM_CXICON和SM_CYICON之内的图标。

请参阅 CWinApp::LoadStandardIcon, CWinApp::LoadOEMIcon, ::LoadIcon

CWinApp::LoadOEMCursor

HCURSOR LoadOEMCursor(UINT *nIDCursor*) const;

返回值

如果成功，则返回光标的句柄；否则返回NULL。

参数

nIDCursor

指定了预定义的Windows光标的ODC_常量标识符。你必须在#include <afxwin.h> 语句之前定义 #define OEMRESOURCE，以获取对WINDOWS.H中OCR_常量的访问权限。

说明

这个函数载入nIDCursor指定的Windows预定义光标。
要访问预定义的Windows光标，应使用LoadOEMCursor或LoadStandardCursor成员函数。

示例

```
// 在 stdafx.h 文件中，加入 #define OEMRESOURCE 以
// 包含 windows.h 对 OCR_ 值的定义
#define OEMRESOURCE
#include <afxwin.h>              // MFC 内核和标准组件
#include <afxext.h>              // MFC 扩展（包括 VB）
HCURSOR hCursor;
// 载入预定义的 Windows 全尺寸光标
hCursor = AfxGetApp()->LoadOEMCursor(OCR_SIZEALL);
```

请参阅 CWinApp::LoadCursor, CWinApp::LoadStandardCursor, ::LoadCursor

CWinApp::LoadOEMIcon

HICON LoadOEMIcon(UINT *nIDIcon*) const;

返回值

如果成功，则返回图标的句柄；否则返回NULL。

参数

nIDIcon

指定了预定义的Windows图标的OIC_常量标识符。你必须在#include <afxwin.h> 语句之前定义 #define OEMRESOURCE，以获取对WINDOWS.H中OIC_常量的访问权限。

说明

这个函数载入nIDIcon指定的Windows预定义图标资源。
要访问预定义的Windows图标，应使用 LoadOEMIcon 或 LoadStandardIcon 成员函数。

请参阅 CWinApp::LoadStandardIcon, CWinApp::LoadIcon, ::LoadIcon

CWinApp::LoadStandardCursor

HCURSOR LoadStandardCursor(LPCTSTR *lpszCursorName*) const;

返回值

如果成功，则返回光标的句柄；否则返回NULL。

参数

lpszCursorName

标识了预定义的 Windows **光标的** IDC_**常量标识符。这些标识符在** WINDOWS.H **中定义。下面给出的是可能取值的列表和** *lpszCursorName* **的含义：**

- IDC_ARROW **标准的箭头光标**
- IDC_IBEAM **标准的插入文本光标**
- IDC_WAIT **当** Windows **执行耗时操作时使用的沙漏光标**
- IDC_CROSS **用于选择的十字光标**
- IDC_UPARROW **向上指的箭头光标**
- IDC_SIZE **已不被支持。应使用** IDC_SIZEALL
- IDC_SIZEALL **四向箭头。用于改变窗口大小。**
- IDC_ICON **以不被支持。应使用** IDC_ARROW。
- IDC_SIZENWSE **双向箭头，尾部在左上角和右下角。**

- IDC_SIZENESW 双向箭头，尾部在右上角和左下角。
- IDC_SIZEWE 双向水平箭头
- IDC_SIZENS 双向垂直箭头

说明

载入lpszCursorName指定的Windows预定义光标。
要访问Windows的预定义光标，应使用LoadStandardCursor 或 LoadOEMCursor成员函数。

示例

```
HCURSOR hCursor;
// 载入 Windows预定义的上箭头光标。
hCursor = AfxGetApp()->LoadStandardCursor(IDC_UPARROW);
```

请参阅 CWinApp::LoadOEMCursor, CWinApp::LoadCursor, ::LoadCursor

CWinApp::LoadStandardIcon

HICON LoadStandardIcon(LPCTSTR *lpszIconName*) const;

返回值

如果成功，则返回图标的句柄；否则返回NULL。

参数

lpszIconName

指定了Windows的预定义图标的常量标识符。这些标识符在WINDOWS.H中定义。下面列出的是可能的取值和*lpszIconName*的含义：

- IDI_APPLICATION 缺省的应用程序图标
- IDI_HAND 手形图标，用于严重警告信息
- IDI_QUESTION 问号图标，用于提示信息
- IDI_EXCLAMATION 感叹号图标，用于警告信息
- IDI_ASTERISK 星号图标，用于通知消息

说明

这个函数载入lpszIconName指定的Windows预定义图标。
要访问Windows的预定义图标，应使用LoadStandardIcon或LoadOEMIcon成员函数。

请参阅 CWinApp::LoadOEMIcon, CWinApp::LoadIcon, ::LoadIcon

CWinApp::LoadStdProfileSettings

void LoadStdProfileSettings(UINT *nMaxMRU* = _AFX_MRU_COUNT);

参数

nMaxMRU

保存的最近使用文件个数。

说明

在InitInstance**成员函数内部调用这个成员函数以允许并载入最近使用（**MRU**）文件列表和最后一次预览状态。如果**nMaxMRU**为**0**，则不维护**MRU**列表。**

请参阅 CWinApp::OnFileOpen, CWinApp::AddToRecentFileList

CWinApp::OnContextHelp

afx_msg void OnContextHelp( );

说明

你必须在CWinApp**类的消息映射中加入**

ON_COMMAND(ID_CONTEXT_HELP, OnContextHelp)

语句，同时加入加速键表入口，通常是SHIFT+F1，以允许这个成员函数。

OnContextHelp使应用程序进入帮助模式。光标变为箭头和问号，用户可以移动鼠标指针，然后按下鼠标左键以选择对话框、窗口、菜单或命令按钮。这个成员函数获得光标下方对象的帮助上下文并据此调用Windows函数WinHelp。

请参阅 CWinApp::OnHelp, CWinApp::WinHelp

CWinApp::OnDDECommand

virtual BOOL OnDDECommand(LPTSTR *lpszCommand*);

返回值

如果处理了命令，则返回非零值；否则返回0。

参数

lpszCommand
指向应用程序接收到的DDE命令字符串。

说明

当主框架窗口接收到一个DDE执行消息时，框架就调用这个函数。缺省的实现检查此命令是否申请打开一个文档，如果是，则打开指定的文档。通常当用户

双击一个数据文件的时候，Windows的文件管理器就发送这样的DDE命令字符串。如果要处理其它的DDE执行命令，如打印命令，则应重载此函数。

示例

```
BOOL CMyApp::OnDDECommand(LPTSTR lpszCommand)
{
if (CWinApp::OnDDECommand(lpszCommand))
   return TRUE;
// 处理应用程序能识别的任何 DDE 命令名返回 TRUE。
// 分析 DDE 命令字符串的例子参见 CWinApp::OnDDECommand 的实现
// 对于不能处理的 DDE 命令，返回 FALSE。
return FALSE;
}
```

请参阅　CWinApp::EnableShellOpen

CWinApp::OnFileNew

afx_msg void OnFileNew( );

说明

你必须在CWinApp类的消息映射中加入以下语句：

ON_COMMAND(ID_FILE_NEW, OnFileNew)

以允许这个成员函数。

如果允许这个成员函数，则这个函数将处理File New命令。

有关此函数的缺省动作以及替换这个成员函数的方法参见技术注释22。

示例

```
// 下面的消息映射是 AppWizard 生成的，将
// File New，Open 和 Print Setup 菜单命令
// 映射到框架中这些命令的缺省实现。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE - the ClassWizard will add and remove mapping macros here.
    //       DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
// 标准的基于文件的文档命令
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// 标准的打印设置命令
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

```
// 下面的消息映射演示了如何将
// File New，Open 和 Print Setup 菜单命令重定向到
// 你的 CWinApp 派生类中实现的处理函数。你可以
// 使用 ClassWizard 来绑定这些命令，如下面所演示的，
// 因为消息映射入口用//{{AFX_MSG_MAP 和//}}AFX_MSG_MAP
// 括起来。注意，如果需要，你可以将处理函数命名为
// CMyApp::OnFileNew，而不是 CMyApp::OnMyFileNew，其它处理函数类似。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_FILE_NEW, OnMyFileNew)
ON_COMMAND(ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND(ID_FILE_PRINT_SETUP, OnMyFilePrintSetup)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

请参阅 CWinApp::OnFileOpen

CWinApp::OnFileOpen

```
afx_msg void OnFileOpen( );
```

说明

你必须在CWinApp类的消息映射中加入以下语句：

```
ON_COMMAND( ID_FILE_OPEN, OnFileOpen )
```

以允许这个成员函数。

如果允许这个成员函数，它将处理File Open命令。

有关此函数的缺省动作以及替换这个成员函数的方法参见技术注释22。

示例

```
// 下面的消息映射是 AppWizard 生成的，将
// File New， Open 和 Print Setup 菜单命令
// 映射到框架中这些命令的缺省实现。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
   // NOTE - the ClassWizard will add and remove mapping macros here.
   //       DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
// 标准的基于文件的文档命令
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// 标准的打印设置命令
```

```
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
// 下面的消息映射演示了如何将
// File New，Open 和 Print Setup 菜单命令重定向到
// 你的 CWinApp 派生类中实现的处理函数。你可以
// 使用 ClassWizard 来绑定这些命令，如下面所演示的，
// 因为消息映射入口用//{{AFX_MSG_MAP 和//}}AFX_MSG_MAP
// 括起来。注意，如果需要，你可以将处理函数命名为
// CMyApp::OnFileNew，而不是 CMyApp::OnMyFileNew，其它处理函数类似。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
    //{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_FILE_NEW, OnMyFileNew)
ON_COMMAND(ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND(ID_FILE_PRINT_SETUP, OnMyFilePrintSetup)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

请参阅 CWinApp::OnFileNew

CWinApp::OnFilePrintSetup

```
afx_msg void OnFilePrintSetup( );
```

说明

你必须在CWinApp类的消息映射中加入以下语句：

ON_COMMAND(ID_FILE_PRINT_SETUP, OnFilePrintSetup)

以允许这个成员函数。
如果允许这个成员函数，它将处理File Print命令。
有关此函数的缺省动作以及替换这个成员函数的方法参见技术注释22。

示例

```
// 下面的消息映射是 AppWizard 生成的，将
// File New，Open 和 Print Setup 菜单命令
// 映射到框架中这些命令的缺省实现。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
// NOTE - the ClassWizard will add and remove mapping macros here.
//       DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
// 标准的基于文件的文档命令
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
```

```
// 标准的打印设置命令
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
// 下面的消息映射演示了如何将
// File New，Open 和 Print Setup 菜单命令重定向到
// 你的 CWinApp 派生类中实现的处理函数。你可以
// 使用 ClassWizard 来绑定这些命令，如下面所演示的，
// 因为消息映射入口用//{{AFX_MSG_MAP 和//}}AFX_MSG_MAP
// 括起来。注意，如果需要，你可以将处理函数命名为
// CMyApp::OnFileNew，而不是 CMyApp::OnMyFileNew，其它处理函数类似。
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
//{{AFX_MSG_MAP(CMyApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_FILE_NEW, OnMyFileNew)
ON_COMMAND(ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND(ID_FILE_PRINT_SETUP, OnMyFilePrintSetup)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

请参阅 CWinApp::OnFileNew

CWinApp::OnHelp

afx_msg void OnHelp( );

说明

你必须在CWinApp类的消息映射中加入以下语句：

ON_COMMAND(ID_HELP, OnHelp)

以允许这个成员函数。通常你还将为F1键加入一个加速键入口。允许F1键只是一个惯例，并不是必须的。
如果允许这个成员函数，当用户按下F1键的时候，框架就调用这个函数。
这个消息处理函数的缺省实现确定对应当前窗口、对话框或菜单项的帮助上下文，然后调用WINHELP.EXE。如果当前没有可用的上下文，则使用缺省的上下文。
如果要为不是当前具有焦点的窗口、对话框、菜单项或工具条按钮的其它对象设置上下文，则应重载这个成员函数。使用所需的帮助上下文ID来调用WinHelp。

请参阅 CWinApp::OnContextHelp, CWinApp::OnHelpUsing, CWinApp::OnHelpIndex, Cwin- App::WinHelp

CWinApp::OnHelpFinder

```
afx_msg void OnHelpFinder( );
```

说明

你必须在CWinApp**类的消息映射中加入以下语句：**

ON_COMMAND(ID_HELP_FINDER, OnHelpFinder)

以允许这个成员函数。

如果允许这个成员函数，当应用程序的用户选择了 Help Finder**命令，用标准的**HELP_FINDER**主题激活了**WinHelp**时，框架就调用这个消息处理函数。**

请参阅 CWinApp::OnHelp, CWinApp::OnHelpUsing, CWinApp::WinHelp, CWinApp::OnHelpIndex

CWinApp::OnHelpIndex

```
afx_msg void OnHelpIndex( );
```

说明

你必须在CWinApp**类的消息映射中加入以下语句：**

ON_COMMAND(ID_HELP_INDEX, OnHelpIndex)

以允许这个成员函数。

如果允许这个成员函数，当应用程序的用户选择了 Help Index 命令，用标准的 HELP_ INDEX 主题激活了 WinHelp 时，框架就调用这个消息处理函数。

请参阅 CWinApp::OnHelp, CWinApp::OnHelpUsing, CWinApp::WinHelp

CWinApp::OnHelpUsing

afx_msg void OnHelpUsing( );

说明

你必须在CWinApp类的消息映射中加入以下语句：

ON_COMMAND(ID_HELP_USING, OnHelpUsing)

以允许这个成员函数。

如果允许这个成员函数，当应用程序的用户选择了Help Using命令，用标准的HELP_HELPONHELP主题激活了WinHelp时，框架就调用这个消息处理函数。

请参阅 CWinApp::OnHelp, CWinApp::OnHelpIndex, CWinApp::WinHelp

CWinApp::OnIdle

virtual BOOL OnIdle(LONG *lCount*);

返回值

如果要接收更多的空闲处理时间，则返回非零值；如果不需要更多的空闲时间则返回0。

参数

lCount

该参数是一个计数值，当应用程序的消息队列为空，OnIdle**函数被调用时，该计数值就增加**1。**每当一条新消息被处理时，该计数值就被复位为**0。**你可以使用**lCount**参数来确定应用程序不处理消息时空闲时间的相对长度。**

说明

如果要执行空闲时处理，则重载这个成员函数。当应用程序的消息队列为空时，OnIdle**就在缺省的消息循环中被调用。你可以用重载函数来调用自己的后台空闲处理任务。**

OnIdle **应返回** 0 **以表明不需要更多的空闲处理时间。当消息队列为空时，**OnIdle

每被调用一次 lCount 参数就增加，而每处理一条新消息 lCount 就被复位为 0。你可以根据这个计数值调用不同的空闲处理例程。

下面总结了空闲循环处理：

1. 如果微软基础类库中的消息循环检查消息队列并发现没有未被处理的消息，它就为应用程序对象调用 OnIdle 函数，并将 lCount 参数设为 0。
2. OnIdle 执行一些处理，然后返回一个非零值，表示它还需要被调用，以进行进一步处理。
3. 消息循环再次检查消息队列。如果没有未处理的消息，则再次调用 OnIdle，增加 lCount 参数。
4. 最后，OnIdle 结束所有的空闲任务并返回 0。这就告诉消息循环停止调用 OnIdle 直到在消息队列中接收到下一条消息为止，在那时，空闲循环将重新启动，而参数被设为 0。

因为只有在 OnIdle 返回之后应用程序才能处理用户输入，因此在 OnIdle 中不应进行较长的任务。

> 注意　OnIdle 的缺省实现更新命令用户接口对象，如菜单项和工具条等，还实现了内部数据结构的清理。因此，如果你重载了 OnIdle，你必须用重载版本中使用的 lCount 值来调用 CWinApp::OnIdle。首先调用所有基类的空闲处理（即直到基类的 OnIdle 返回 0）。如果你需要在基类处理完成之前进行一些工作，则应回顾基类的实现以在自己的工作期间选择一个合适的 lCount 值。

示例

下面的两个例子演示了OnIdle**的用法。第一个例子处理两个空闲任务，用**lCount**参数来排列这些任务的优先权。第一个任务优先权较高，一旦可能你就应当执行此任务。第二个任务不十分重要，只有当用户输入有一个较长时间的间歇的时候才应执行此任务。注意其中对基类的**OnIdle**的调用。第二个例子管理着一组具有不同优先权的空闲任务。**

```
BOOL CMyApp::OnIdle(LONG lCount)
{
BOOL bMore = CWinApp::OnIdle(lCount);
if (lCount == 0)
{
TRACE("App idle for short period of time\n");
bMore = TRUE;
}
else if (lCount == 10)
{
TRACE("App idle for longer amount of time\n");
   bMore = TRUE;
}
else if (lCount == 100)
{
```

```
    TRACE("App idle for even longer amount of time\n");
    bMore = TRUE;
}
    else if (lCount == 1000)
{
    TRACE("App idle for quite a long period of time\n");
// bMore 没有被设为 TRUE，不在需要空闲
// 重要： bMore 没有被设为 FALSE，因为 CWinApp::OnIdle 可能
// 还有其它空闲任务要完成。
}
return bMore;
// 返回 TRUE，只要还有其它空闲任务
}
```

第二个示例

```
// 在这个例子中，有四个空闲循环任务，它们被赋予
// 不同的优先权，运行的机会不同：
// Task1 在空闲时总能运行，要求在框架处理它自己
// 的空闲循环任务时没有消息在等候。（lCount 为 0 或 1）
// Task2 仅当 Task1 以及运行时才能运行，要求当 Task1
// 运行时没有消息在等候。
// Task3 和 Task4 仅当 Task1 和 Task2 都运行之后才能运行，
// 并且在此期间没有消息在等候。如果 Task3 能够运行，
```

```
// 则 Task4 总是在 Task3 之后立即运行。
BOOL CMyApp::OnIdle(LONG lCount)
{
// 在这个例子中，像多数应用程序一样，你应该让基类
// 的 CWinApp::OnIdle 在你试图进行任何附加的空闲循环
// 过程之前完成它的处理。
if (CWinApp::OnIdle(lCount))
return TRUE;
// 基类的 CWinApp::OnIdle 为 lCount 保留 0 和 1 给框架自己的
// 空闲处理使用。如果你希望与框架平等地共享空闲处理
// 时间，则应替换上面的 if 语句，直接调用 CWinApp::OnIdle，
// 然后为 lCount 的值 0 和/或 1 加入一个 case 语句。首先应当研
// 究基类的实现以理解你的空闲循环任务将会如何与框架的
// 空闲循环处理竞争。
switch (lCount)
{
case 2:
    Task1();
        return TRUE; // 下一次给 Task2 一个机会
    case 3:
        Task2();
        return TRUE; // 下一次给 Task3 和 Task4 一个机会
```

```
    case 4:
        Task3();
        Task4();
        return FALSE; // 再次回到空闲循环任务
}
return FALSE;
}
```

CWinApp::OpenDocumentFile

virtual CDocument* OpenDocumentFile(LPCTSTR *lpszFileName*);

返回值

如果成功，则返回指向CDocument**对象的指针；否则返回**NULL**。**

参数

lpszFileName

要打开的文件的名字。

说明

框架调用这个成员函数为应用程序打开指定名字的CDocument文件。如果具有该名字的文档已经被打开了，则包含这个文档的第一个框架窗口将被激活。如果应用程序支持多文档模板，则框架使用文件扩展名查找适当的文档模板，试图载入此文档。如果成功，则文档模板为该文档创建一个框架窗口和视。

示例

```
BOOL CMyApp::InitInstance()
{
// ...
if (m_lpCmdLine[0] == '\0')
{
// 创建一个新（空的）文档
OnFileNew();
}
else
{
// 打开作为第一个命令行参数传递的文件
OpenDocumentFile(m_lpCmdLine);
}
// ...
```

}

CWinApp::ParseCommandLine

void ParseCommandLine(CCommandLineInfo& *rCmdInfo*);

参数

rCmdInfo

对CCommandLineInfo对象的引用。

说明

调用这个函数以解析命令行并将参数发送到CCommandLineInfo::ParseParam，每次一个。

当你通过 AppWizard 开始一个新的 MFC 项目时，AppWizard 将会创建一个 Ccommand- LineInfo 的本地实例，然后在 InitInstance 成员函数中调用 ProcessShellCommand 和 ParseCommandLine。命令行的过程描述如下：

1. 在 InitInstance 中被创建之后，CCommandLineInfo 对象被传递给 ParseCommandLine。
2. 随后 ParseCommandLine 反复调用 CCommandLineInfo::ParseParam，每次解析出一个参数。

3. ParseParam 填充 CCommandLineInfo 对象，随后该对象被传递给 ProcessShellCommand。
4. ProcessShellCommand 处理命令行参数和标志。

注意如果需要的话，你可以直接调用 ParseCommandLine。
有关命令行标志的描述参见 CCommandLineInfo::m_nShellCommand。

请参阅 CCommandLineInfo, CWinApp::InitInstance, CCommandLineInfo::ParseParam, CWinApp::ProcessShellCommand, CCommandLineInfo::m_nShellCommand

CWinApp::PreTranslateMessage

virtual BOOL PreTranslateMessage(MSG* *pMsg*);

返回值

如果消息在 PreTranslateMessage 中被完全处理并且不需要进一步处理，则返回非零值。如果消息还需要按照通常方式处理，则返回零。

参数

pMsg
指向一个 MSG 结构的指针，其中包含了要处理的消息。

说明

如果要在消息被分派到Windows函数TranslateMessage和DispatchMessage之前过滤窗口消息，则应重载这个函数。缺省的实现执行加速键的转换，因此你可以在重载函数中调用CWinApp::PreTranslateMessage成员函数。

请参阅 ::DispatchMessage, ::TranslateMessage

CWinApp::ProcessMessageFilter

virtual BOOL ProcessMessageFilter(int *code*, LPMSG *lpMsg*);

返回值

如果消息被处理，则返回非零值；否则返回0。

参数

code
  指定了钩子代码。这个成员函数使用这些代码以确定如何处理*lpMsg*。

lpMsg
  指向一个Windows的MSG结构的指针。

说明

框架的钩子函数调用这个成员函数以过滤和响应特定的Windows消息。钩子函数在事件被发送到应用程序的正常消息处理过程之前处理这些事件。

如果你重载了这个高级特性，确保你调用了基类版本以维护框架的钩子处理。

请参阅 MessageProc, WH_MSGFILTER

CWinApp::ProcessShellCommand

BOOL ProcessShellCommand(CCommandLineInfo& *rCmdInfo*);

返回值

如果成功地处理了外壳命令，则返回非零值。如果InitInstance返回了FALSE，则返回0。

参数

rCmdInfo

对CCommandLineInfo对象的引用。

说明

这个成员函数被InitInstance调用，用以接收rCmdInfo所标识的CCommandLineInfo对象传递的参数，并执行指定的动作。

当你通过AppWizard开始一个MFC的新项目时，AppWizard将创建CCommandLineInfo的一个本地实例，然后在InitInstance成员函数中调用ProcessShellCommand和ParseCommandLine。命令行按照下面描述的路线传递：

1. 在InitInstance中被创建以后，CCommandLineInfo对象将它传递给ParseCommand-Line。
2. 随后ParseCommandLine反复调用CCommandLineInfo::ParseParam，每次解析一个参数。
3. ParseParam填充CCommandLineInfo对象，然后将之传递给ProcessShellCommand。
4. ProcessShellCommand处理命令行参数和标志。

CCommandLineInfo对象中用CCommandLineInfo::m_nShellCommand标识的数据成员属于下面的枚举类型，它在CCommandLineInfo类中定义。

```
enum{
FileNew,
FileOpen,
FilePrint,
FilePrintTo,
FileDDE,
```

};

有关这些值的简要描述参见CCommandLineInfo::m_nShellCommand。

请参阅　CWinApp::ParseCommandLine, CCommandLineInfo, CCommandLineInfo::ParseParam, CCommandLineInfo::m_nShellCommand

CWinApp::ProcessWndProcException

virtual LRESULT ProcessWndProcException(CException* *e*, const MSG* *pMsg*);

返回值

这个返回值必须返回给Windows。通常，对于Windows消息，返回值为0L，对于命令消息，返回值为1L（TRUE）。

参数

e

指向没有被捕获的异常。

pMsg

一个MSG结构，其中包含了有关导致框架出现异常的Windows消息的信息。

说明

每当应用程序不能捕获应用程序的消息处理函数或命令处理函数出现的异常时，框架就调用这个成员函数。

不要直接调用这个函数。

这个成员函数的缺省实现创建一个消息框。如果这个未被捕获的异常源于一条菜单、工具条或加速键命令失败，则消息框显示一条“Command failed”消息；否则，显示一条“Internal application error”消息。

如果要为异常提供全局处理，则应重载这个成员函数。若你需要显示该消息框，则应调用基类的函数。

请参阅 CWnd::WindowProc, CException

CWinApp::RegisterShellFileTypes

```
void RegisterShellFileTypes( BOOL bCompat = FALSE );
```

参数

bCompat

如果为TRUE，则为外壳命令Print和Print To加入注册表入口，使用户能够从外壳直接打印文件，或者是将文件拖动到打印机对象上。同时它也

加入一个DefaultIcon键。缺省情况下，为了向后的兼容性，这个参数为FALSE。

说明

调用这个函数在Windows的文件管理器中注册应用程序的所有文档类型。这就使用户能够通过在文件管理器内双击而打开应用程序创建的数据文件。对于应用程序中的每个文档模板，在调用AddDocTemplate之后调用RegisterShellFileTypes。当你调用RegisterShellFile- Types的时候，同时也调用EnableShellOpen成员函数。

RegisterShellFileTypes在应用程序维护的CDocTemplate对象列表中反复，并且，对于每个文档模板，在Windows维护的注册表数据库中加入入口。当用户双击文件的时候，文件管理器利用这个入口打开数据文件。这减小了随应用程序发放.REG文件的必要性。

如果注册表数据库中已经将指定的文件扩展名与其它文件类型相关联了，则不会创建新的关联。有关注册信息时所用的字符串格式，请参见CDocTemplate类。

请参阅 CDocTemplate, CWinApp::EnableShellOpen, CWinApp::AddDocTemplate

CWinApp::Run

virtual int Run( );

返回值

WinMain返回的整数值。

说明

这个函数提供了缺省的消息循环。Run函数接收并分派Windows消息，直到应用程序接收到一个WM_QUIT消息。如果应用程序的消息队列当前不包含任何消息，Run就调用OnIdle以执行空闲时处理。接收到的消息先进入PreTranslateMessage成员函数以进行特殊处理，然后发送到Windows函数TranslateMessage，进行标志的键盘转换，最后，调用Windows函数DispatchMessage。

Run很少被重载，但是你也可以重载它以提供特殊的功能。

请参阅

CWinApp::PreTranslateMessage, M_QUIT, ::DispatchMessage, ::TranslateMessage

CWinApp::RunAutomated

BOOL RunAutomated( );

返回值

如果找到了该选项，则返回非零值；否则返回0。

说明

调用这个函数以确定是否出现了“/Automation”或“-Automation”选项，这表明服务器程序是否是由客户应用程序启动的。如果有，则从命令行中清除这个选项。有关OLE自动化的更多信息，参见“Visual C++ 程序员指南”中的“自动化服务器”。

请参阅 CWinApp::RunEmbedded

CWinApp::RunEmbedded

BOOL RunEmbedded( );

返回值

如果发现了这个选项，则返回非零值；否则返回0。

说明

调用这个选项以确定是否出现了“/Embedding”或“-Embedding”选项，这表明服务器应用程序是否是由客户应用程序启动的。如果有，则从命令行中清除这个选项。有关嵌入的更多信息，参见“Visual C++ 程序员指南”中的“服务器：实现服务器”。

请参阅 CWinApp::RunAutomated

CWinApp::SaveAllModified

```
virtual BOOL SaveAllModified( );
```

返回值

如果能够安全地结束应用程序，则返回非零值；如果不安全，则返回0。

说明

当应用程序的主框架窗口要被关闭时，或者接收到WM_QUERYENDSESSION消息时，框架调用这个函数以保存所有的文档。

这个成员函数的缺省实现为应用程序中所有被修改的文档调用CDocument::SaveModified成员函数。

CWinApp::SelectPrinter

void SelectPrinter(HANDLE *hDevNames*, HANDLE *hDevMode*, BOOL *bFreeOld* = TRUE);

参数

hDevNames
　　指向DEVNAMES结构的句柄，标识了指定打印机的驱动程序、设备和输出端口名。

hDevMode
　　指向DEVMODE结构的句柄，指定了有关设备初始化和打印机环境的信息。

bFreeOld
　　释放以前选择的打印机。

说明

调用这个成员函数以选择指定的打印机，并释放先前在Print对话框中选择的打印机。

如果hDevMode和hDevNames都是NULL，则SelectPrinter使用当前的缺省打印机。

请参阅　CPrintDialog, DEVMODE, DEVNAMES

CWinApp::SetDialogBkColor

void SetDialogBkColor(COLORREF *clrCtlBk* = RGB(192, 192, 192), COLORREF *clrCtlText* = RGB(0, 0, 0));

参数

clrCtlBk
　　应用程序的对话框的背景颜色。

clrCtlText
　　应用程序的对话框控件的颜色。

说明

在InitInstance成员函数内调用这个成员函数以设置应用程序中对话框和消息框的缺省背景色和文本颜色。

示例

```
BOOL CMyApp::InitInstance()
{
// 标准的初始化
SetDialogBkColor();            // 将对话框背景颜色设为灰
```

```
LoadStdProfileSettings();   // 载入标准的 INI 文件选项（包括 MRU）
// ...
}
```

CWinApp::SetRegistryKey

```
void SetRegistryKey( LPCTSTR lpszRegistryKey );
void SetRegistryKey( UINT nIDRegistryKey );
```

参数

lpszRegistryKey

字符串指针，包含了键的名字。

nIDRegistryKey

注册表中键的ID/索引。

说明

这个函数将应用程序的设置保存在注册表而不是INI文件中。这个函数设置m_pszRegistry Key，它被CWinApp的成员函数GetProfileInt，GetProfileString，WriteProfileInt和WriteProfileString使用。如果调用了这个函数，最近使用（MRU）的文件也被保存到注册表中。通常注册表的键为公司的名字。它保存在如下形式的键中：

HKEY_CURRENT_USER\Software\<company name>\<application name>\<section name>\<value name>.

请参阅

CWinApp::InitInstance, CWinApp::GetProfileInt, CWinApp::GetProfileString, CWinApp::WriteProfileInt, CWinApp::WriteProfileString

CWinApp::WinHelp

virtual void WinHelp(DWORD *dwData*, UINT *nCmd* = HELP_CONTEXT);

参数

dwData
指定了附加数据。这些值依赖于*nCmd*参数的值。

nCmd
指定了请求的帮助类型。可能取值的列表以及它们影响*dwData*参数的方式参见Windows函数WinHelp。

说明

调用这个函数以激活WinHelp应用程序。框架也会调用这个函数以激活WinHelp

应用程序。

当你的应用程序终止时，框架会自动关闭 WinHelp 应用程序。

示例

```
// 头文件： HELPIDS.H
//
// 这个例子的头文件被包含了两次：
// (1) 它被.CPP 文件包含，将 DWORD 类型的上下
// 文 ID 传递给 CWinApp::WinHelp。
// (2) 它被包含在.HPJ 文件中的[MAP]部分，将帮
// 助上下文字符串“ HID_MYTOPIC ”与帮助上下
// 文 ID101 关联起来。
// 帮助上下文字符串“ HID_MYTOPIC ”标识了帮助
// 的.RTF 源文件中的帮助主题，带有“ # ”脚注：
//         # HID_MYTOPIC
//
// 没有必要用这种方式管理在 RESOURCE.H 文件中定义
//的与命令对象或用户界面对象相关的帮助主题的帮助
//上下文 ID。你可以使用 MAKEHM 工具，或者是
//AppWizard 的上下文帮助选项所生成的 MAKEHELP.BAT
//文件来为这些 ID 生成帮助映射（.HM）文件。仅对那些
```

```
//不与命令对象或用户界面对象相关的帮助主题才有必要
//按照这里演示的方式管理帮助上下文 ID。
#define HID_MYTOPIC 101
// 显示了在帮助的.RTF 文件中具有上下文字符串
// “HID_MYTOPIC”的自定义帮助主题，它被映
// 射到 HELPIDS.H 文件中的 DWORD 型 ID 值 HID_MYTOPIC。
AfxGetApp()->WinHelp(HID_MYTOPIC);
// 下面是 MAKEHM 工具生成的帮助映射（.HM）文件
// 中的一行代码，随后它被 AppWizard 的上下文帮助
// 选项所生成的 MAKEHELP.BAT 文件所调用。
// MAKETM 工具读出应用程序的 RESOURCE.H 文件中
// 的#define 语句：
//      #define ID_MYCOMMAND 0x08004
// 然后加上一个帮助 ID 的偏移量 0x10000 以创建帮助上
// 下文的 DWORD 值 0x18004。有关帮助 ID 偏移量的更
// 多信息参见 MFC 的技术注释 28。
HID_MYCOMMAND                              0x18004
// 你很少需要利用帮助上下文 ID 为命令或用户界面对象
// 直接调用 WinHelp。例如，当焦点位于 My Command 菜
// 单项上时，如果用户按下 F1 键，框架就会自动调用 WinHelp。
// 但是，如果你想要为与命令相关的帮助主题直接调用 WinHelp，
// 下面是调用的方法：
```

```
AfxGetApp()->WinHelp(0x10000 + ID_MYCOMMAND);
```

请参阅 CWinApp::OnContextHelp, CWinApp::OnHelpUsing, CWinApp::OnHelp, CWinApp::OnHelpIndex, ::WinHelp

CWinApp::WriteProfileInt

BOOL WriteProfileInt(LPCTSTR *lpszSection*, LPCTSTR *lpszEntry*, int *nValue*);

返回值

如果成功，则返回非零值；否则返回0。

参数

lpszSection

指向一个以null**结尾的字符串，指定了包含入口的部分。如果这个部分不存在，就创建它。这个部分的名字对大小写不敏感，字符串可以是大写字符和小写字符的任意组合。**

lpszEntry

指向一个以null**结尾的字符串，指定了包含要写入值的入口的部分。如果在指定的部分中不存在这个入口，就创建它。**

nValue

包含了要写入的值。

说明

调用这个成员函数以把指定的值写入应用程序的注册表或INI文件的指定部分。这些入口按如下方式保存：

- 在Windows NT中，该值被保存在注册表中。
- 在Windows 3.X中，该值被保存在WIN.INI文件中。
- 在Windows 95中，该值被保存在WIN.INI的缓存版本中。

示例

```
CString strSection       = "My Section";
CString strStringItem    = "My String Item";
CString strIntItem       = "My Int Item";
CWinApp* pApp = AfxGetApp();
pApp->WriteProfileString(strSection, strStringItem, "test");
CString strValue;
strValue = pApp->GetProfileString(strSection, strStringItem);
ASSERT(strValue == "test");
pApp->WriteProfileInt(strSection, strIntItem, 1234);
int nValue;
```

```
nValue = pApp->GetProfileInt(strSection, strIntItem, 0);
ASSERT(nValue == 1234);
```

请参阅 CWinApp::GetProfileInt, CWinApp::WriteProfileString

CWinApp::WriteProfileString

BOOL WriteProfileString(LPCTSTR *lpszSection*, LPCTSTR *lpszEntry*, LPCTSTR *lpszValue*);

返回值

如果成功，则返回非零值；否则返回0。

参数

lpszSection

指向一个以null结尾的字符串，指定了包含入口的部分。如果这个部分不存在，就创建它。这个部分的名字对大小写不敏感，字符串可以是大写字符和小写字符的任意组合。

lpszEntry

指向一个以null结尾的字符串，指定了包含要写入值的入口的部分。如果在指定的部分中不存在这个入口，就创建它。

lpszValue

指向要写入的字符串。如果这个参数为NULL，则将删除*lpszEntry*指定的键。

说明

调用这个函数将指定的字符串写入应用程序的注册表或INI文件的指定部分。

这些入口按如下方式保存：

- 在 Windows NT 中，该值被保存在注册表中。
- 在Windows 3.x中，该值被保存在WIN.INI文件中。
- 在Windows 95中，该值被保存在WIN.INI的缓存版本中。

示例

```
CString strSection        = "My Section";
CString strStringItem     = "My String Item";
CString strIntItem        = "My Int Item";
CWinApp* pApp = AfxGetApp();
pApp->WriteProfileString(strSection, strStringItem, "test");
CString strValue;
strValue = pApp->GetProfileString(strSection, strStringItem);
ASSERT(strValue == "test");
pApp->WriteProfileInt(strSection, strIntItem, 1234);
```

```
int nValue;
nValue = pApp->GetProfileInt(strSection, strIntItem, 0);
ASSERT(nValue == 1234);
```

请参阅 CWinApp::GetProfileString, CWinApp::WriteProfileInt, ::WritePrivateProfileStringCWinApp::SetRegistryKey

数据成员

CWinApp::m_bHelpMode

说明

如果应用程序处于帮助上下文模式（通常由SHIFT+F1激活），则为TRUE，否则为FALSE。在帮助上下文模式中，光标变为问号，用户可以在屏幕上移动它。如果你希望在帮助模式下实现特殊处理，则检查这个标志。m_bHelpMode是BOOL类型的公有变量。

CWinApp::m_hInstance

说明

对应Windows传递给WinMain的hInstance参数。m_hInstance数据成员是指向运行在Windows环境中的应用程序当前实例的句柄。它是由全局函数AfxGetInstanceHandle返回的。m_hInstance是HINSTANCE类型的公有变量。

示例

```
// 通常你不用直接把应用程序的 hInstance 传递给
// Windows 的 API，因为等效的 MFC 成员函数传递
// 了 hInstance。下面的例子不太典型。
HCURSOR hCursor;
hCursor = ::LoadCursor(AfxGetApp()->m_hInstance,
   MAKEINTRESOURCE(IDC_MYCURSOR));
// 获得应用程序的 hInstance 的更直接的方式是调用 AfxGetInstanceHandle：
hCursor = ::LoadCursor(AfxGetInstanceHandle(),
   MAKEINTRESOURCE(IDC_MYCURSOR));
// 如果你在载入资源的时候需要 hInstance，最好
// 调用 AfxGetResourceHandle，而不是 AfxGetInstanceHandle。
hCursor = ::LoadCursor(AfxGetResourceHandle(),
```

```
    MAKEINTRESOURCE(IDC_MYCURSOR));
// 载入光标资源的更好方式是调用 CWinApp::LoadCursor
hCursor = AfxGetApp()->LoadCursor(IDC_MYCURSOR);
```

CWinApp::m_hPrevInstance

说明

对应于Windows传递给WinMain的hPrevInstance参数。
在Win32应用程序中，m_hPrevInstance数据成员总是被设为NULL。如果要找到应用程序的先前实例，则应使用CWnd::FindWindow。

CWinApp::m_lpCmdLine

说明

对应于Windows传递给WinMain的lpCmdLine参数。指向一个以null结尾的字符串，指定了应用程序的命令行。用m_lpCmdLine可以访问当应用程序启动时用户输入的命令行参数。m_lpCmdLine是LPSTR类型的公有变量。

示例

```
BOOL CMyApp::InitInstance()
```

```
{
// ...
if (m_lpCmdLine[0] == '\0')
{
   // 创建一个新（空的）文档
   OnFileNew();
}
else
{
   // 打开作为第一个命令行参数传递的文件
   OpenDocumentFile(m_lpCmdLine);
}
// ...
}
```

CWinApp::m_nCmdShow

说明

对应于Windows**传递给**WinMain**的**nCmdShow**参数。当你为应用程序的主窗口调用** CWnd::ShowWindow **函数的时候，可以将** m_nCmdShow **作为参数传递。**

m_nCmdShow是int类型的公有变量。

示例

```
BOOL CMyApp::InitInstance()
{
// ...
// 创建 MDI 主框架窗口
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
   return FALSE;
// 保存主框架窗口的指针。这是框架识别主框架窗口的唯一途径。
m_pMainWnd = pMainFrame;
// 根据 nCmdShow 参数显示主窗口，这个参数是当应用
// 程序第一次启动的时候传递给它的。
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();
// ...
}
```

CWinApp::m_pActiveWnd

说明

利用这个数据成员来保存OLE容器应用程序的主窗口指针，它使你的OLE服务器应用程序能够现场激活。如果这个数据成员为NULL，则应用程序不能被现场激活。
当框架窗口被OLE容器应用程序现场激活时，框架就设置这个成员变量。

请参阅　AfxGetMainWnd, CWinThread::m_pMainWnd

CWinApp::m_pszAppName

说明

指定应用程序的名字。应用程序可以从传递给CWinApp的构造函数的参数中得到，如果其中没有指定名字，则是ID为AFX_IDS_APP_TITLE的资源字符串。如果在资源中找不到应用程序的名字，那么它来自程序的可执行文件名。
全局函数AfxGetAppName返回该值。m_pszAppName是const char* 类型的公有变量。

注意　如果你为 m_pszAppName 分配了一个值，它必须是在堆中动态分配

的。CWinApp 的析构函数利用这个指针调用 free( )。你可以使用运行库函数_tcsdup( )来分配内存。同时，在为它分配一个新值之前，先释放与当前指针相关的内存。例如：

```
// 首先释放在 CWinApp 启动的时候 MFC 分配的字符串
// 字符串是在调用 InitInstance 之前分配的
free((void*)m_pszAppName);
// 改变应用程序文件的名字
// CWinApp 的析构函数将释放内存
m_pszAppName=_tcsdup(_T("d:\\somedir\\myapp.exe"));
```

示例

```
CWnd* pWnd;
// 将 pWnd 设为某个 CWnd 对象的指针，它的窗口已经创建了。
// 后面对 CWnd::MessageBox 的调用使用应用
// 程序的名字作为消息框的标题。
pWnd->MessageBox("Some message", AfxGetApp()->m_pszAppName);
// 获得应用程序名字的更直接的方式是调用 AfxGetAppName
pWnd->MessageBox("Some message", AfxGetAppName());
// 用应用程序的名字作为消息框标题来显示消息框
// 的更简单的方式是调用 AfxMessageBox。
AfxMessageBox("Some message");
```

CWinApp::m_pszExeName

说明

该变量包含了应用程序的可执行文件名，不包括扩展名。与m_pszAppName**不同，这个名字中不包含空白。**m_pszExeName**是**const char* **类型的公有变量。**

> **注意** 如果你为 m_pszExeName 分配了一个值，它必须是在堆中动态分配的。CWinApp 的析构函数利用这个指针调用 free( )。你可以使用运行库函数_tcsDup( )来分配内存。同时，在为它分配一个新值之前，先释放与当前指针相关的内存。例如：

```
// 首先释放在 CWinApp 启动的时候 MFC 分配的字符串
// 字符串是在调用 InitInstance 之前分配的
free((void*)m_pszExeName);
// 改变.EXE 文件的名字
// CWinApp 的析构函数将释放内存
m_pszExeName=_tcsdup(_T("d:\\somedir\\myapp"));
```

CWinApp::m_pszHelpFilePath

说明

包含了应用程序的帮助文件的路径。在缺省情况下，框架将m_pszHelpFilePath初始化为应用程序的名字，并加上“.HLP”。如果要改变帮助文件的名字，则应将m_pszHelpFilePath设为包含所需的帮助文件的完整名字的字符串。完成这个操作的合适的位置是在应用程序的InitInstance函数中。m_pszHelpFilePath是const char*类型的公有变量。

注意　如果你为m_pszHelpFilePath分配了一个值，它必须是在堆中动态分配的。CWinApp的析构函数利用这个指针调用free( )。你可以使用运行库函数_tcsDup( )来分配内存。同时，在为它分配一个新值之前，先释放与当前指针相关的内存。例如：

```
// 首先释放在CWinApp启动的时候MFC分配的字符串
// 字符串是在调用InitInstance之前分配的
free((void*)m_pszHelpFilePath);
// 改变.HLP文件的名字
// CWinApp的析构函数将释放内存
m_pszHelpFilePath=_tcsdup(_T("d:\\somedir\\myapp.hlp"));
```

CWinApp::m_pszProfileName

说明

包含了应用程序的.INI**文件的名字。**m_pszProfileName**是**const char* **类型的公有变量。**

> **注意** 如果你为m_pszProfileName分配了一个值，它必须是在堆中动态分配的。CWinApp的析构函数利用这个指针调用free( )。你可以使用运行库函数_tcsDup( )来分配内存。同时，在为它分配一个新值之前，先释放与当前指针相关的内存。例如：

```
// 首先释放在CWinApp启动的时候MFC分配的字符串
// 字符串是在调用InitInstance之前分配的
free((void*)m_pszProfileName);
// 改变.INI文件的名字
// CWinApp的析构函数将释放内存
m_pszProfileName=_tcsdup(_T("d:\\somedir\\myini.ini"));
```

请参阅 CWinApp::GetProfileString, CWinApp::GetRofileInt, CWinApp::WriteProfileInt, CWinApp::WriteProfileString。

CWinApp::m_pszRegistryKey

LPCTSTR m_pszRegistryKey;

说明

用于确定应用程序的配置信息保存在注册表或INI文件的什么位置。通常，这个数据成员被当作是只读的。
注册表入口按如下方式保存：

- **在Windows NT中，该值被保存在注册表中。应用程序配置的名字被加在下述注册表键的后面：** HKEY_CURRENT_USER/SoftWare/LocalAppWizard-Generated/
- **在Windows 3.x中，该值被保存在INI文件中。**
- **在Windows 95中，该值被保存在INI文件的缓存版本。**

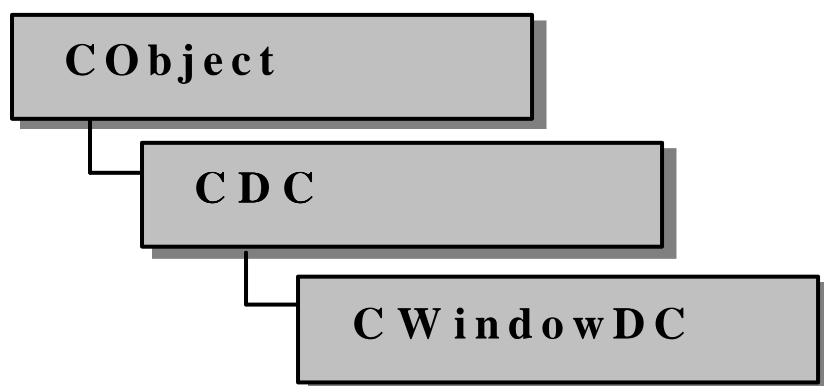
如果你为m_pszRegistryKey分配了一个值，它必须是在堆中动态分配的。CWinApp的析构函数利用这个指针调用free( )。你可以使用运行库函数_tcsDup( )来分配内存。同时，在为它分配一个新值之前，先释放与当前指针相关的内存。例如：

```
// 首先释放在CWinApp启动的时候MFC分配的字符串
// 字符串是在调用InitInstance之前分配的
free((void*)m_pszRegistryKey);
// 改变注册 表键的名字
```

```
// CWinApp的析构函数将释放内存
m_pszRegistryKey=_tcsdup(_T("HKEY_CURRENT_USER\\SoftWare\\mycompany\\myapp\\thissection\\this-value"));
```

请参阅　CWinApp::SetRegistryKey

CWindowDC



CWindowDC类是从CDC继承的。它在构造的时候调用Windows函数GetWindowDC，在销毁的时候调用ReleaseDC。这意味着CWindowDC对象可以访问CWnd的全部屏幕区域（包括客户区和非客户区）。

有关使用CWindowDC的更多信息参见“Visual C++ 程序员指南”中的“设备环境”。

#include <afxwin.h>

请参阅　CDC

CWindowDC 类成员

Construction	
CWindowDC	构造一个 CWindowDC 对象

Data Members	
m_hWnd	与这个 CWindowDC 相关联的 HWND 句柄

成员函数

CWindowDC::CWindowDC

CWindowDC(CWnd* *pWnd*);
throw(CResourceException);

参数

pWnd

窗口指针，设备环境对象将访问其客户区域。

说明

构造一个CWindowDC对象，它可以访问pWnd指向的CWnd对象的整个屏幕区域（包括客户区和非客户区）。构造函数调用Windows函数GetWindowDC。

如果对Windows函数GetWindowDC的调用失败了,将出现异常(属于Cresource-Exception类型）。如果Windows已经分配了所有可用的设备环境，则可能不能访问设备环境。在Windows下，在任何给定时刻应用程序只能使用5个公共显示环境之一。

请参阅　CDC, CClientDC, CWnd

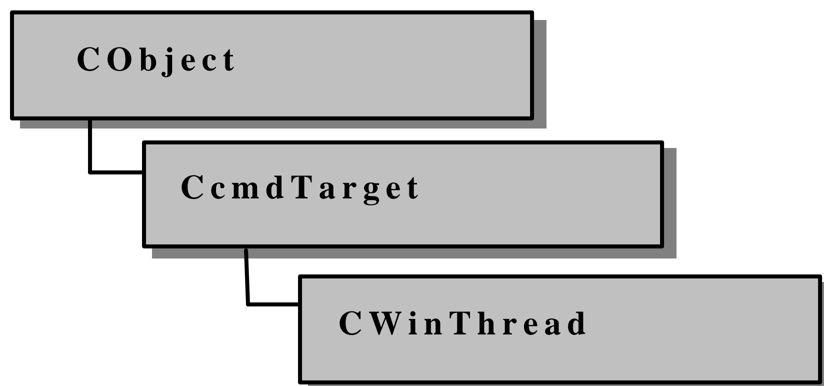
数据成员

CWindowDC::m_hWnd

说明

CWnd指针的HWND句柄被用于构造CWindowDC对象。m_hWnd是HWND类型的保护变量。

CWinThread



CWinThread 对象代表在一个应用程序内运行的线程。运行的主线程通常由 CWinApp 的派生类提供；CWinApp 由 CWinThread 派生。另外，CWinThread 对象允许一给定的应用程序拥有多个线程。

CWinThread 支持两种线程类型：工作者线程和用户界面线程。工作者线程没有收发消息的功能：例如，在电子表格应用程序中进行后台计算的线程。用户界面线程具有收发消息的功能，并处理从系统收到的消息。CWinApp 及其派生类是用户界面线程的例子。其它用户界面线程也可由 CWinThread 直接派生。

CWinThread 类的对象存在于线程的生存期。如果你希望改变这个特性，将

m_bAutoDelete 设为 FALSE。

要使你的代码和 MFC 是完全线程安全的，CWinThread 类是完全必要的。框架使用的用来维护与线程相关的信息的线程局部数据由 CWinThread 对象管理。由于依赖 CWinThread 来处理线程局部数据，任何使用 MFC 的线程必须由 MFC 创建。例如，由运行时函数_beginthreadex 创建的线程不能使用任何 MFC API。

为了创建一个线程，调用 AfxBeginThread 函数。根据你需要工作者线程还是用户界面线程，有两种调用 AfxBeginThread 的格式。如果你需要用户界面线程，则将指向你的 CWinThread 派生类的 CRuntimeClass 的指针传递给 AfxBeginThread。如果你需要创建工作者线程，则将指向控制函数的指针和控制函数的参数传递给 AfxBeginThread。对于工作者线程和用户界面线程，你可以指定可选的参数来修改优先级，堆栈大小，创建标志和安全属性。AfxBeginThread 线程将返回指向新的 CWinThread 对象的指针。

与调用 AfxBeginThread 相反，你可以构造一个 CWinThread 派生类的对象，然后调用 CreateThread。如果你需要在连续创建和终止线程的执行之间重复使用 CWinThread 对象，这种两步构造方法非常有用。

有关 CWinThread 的进一步信息，参见“Visual C++程序员指南”中的文章“用 C++和 MFC 实现多线程”，“多线程：创建用户界面线程”，“多线程：创建工作者线程”和“多线程：如何使用同步”。

请参阅　CWinApp,CCmdTarget

CWinThread 类成员

Data Members

m_bAutoDelete	指定线程结束时是否要销毁对象
m_hThread	当前线程的句柄
m_nThreadID	当前线程的 ID
m_pMainWnd	保存指向应用程序的主窗口的指针
m_pActiveWnd	指向容器应用程序的主窗口，当一个 OLE 服务器被现场激活时

Construction

CwinThread	构造一个 CWinThread 对象
CreateThread	开始一个 CWinThread 对象的执行

Operations

GetMainWnd	查询指向线程主窗口的指针
GetThreadPriority	获取当前线程的优先级
PostThreadMessage	向另外的 CWinThread 对象传递一条消息
ResumeThread	减少一个线程的挂起计数
SetThreadPriority	设置当前线程的优先级
SuspendThread	增加一个线程的挂起计数

Overridables	
ExitInstance	重载以进行线程终止时的清理工作
InitInstance	重载以实现线程实例的初始化
OnIdle	重载以进行线程特定的空闲操作
PreTranslateMessage	在消息被发送到 Windows 函数 TranslateMessage 和 DispatchMessa- ge 之前过滤消息
IsIdleMessage	检测特定的消息
ProcessWndProcException	截获线程消息和命令处理函数出现的所有未处理的异常
ProcessMessageFilter	在特定的消息到达应用程序之前截获消息
Run	线程的具有消息收发功能的控制函数，可重载以定制缺省的消息循环

成员函数

CWinThread::CreateThread

BOOL CreateThread(DWORD *dwCreateFlags* = 0, UINT *nStackSize* = 0, LPSECURITY_ATTRIBUTES *lpSecurityAttrs* = NULL);

返回值

若成功地创建了线程，返回值为非零；否则，返回值为 0。

参数

dwCreateFlags

指定控制线程的创建的附加标志。该标志可以是下列两个值之一：

- CREATE_SUSPENDED 启动线程时将挂起计数置为一。直到调用 ResumeThread 线程才执行。
- 创建后立即启动线程

nStackSize

指定以字节数计的新线程的堆栈大小。如果为 0，堆栈的大小缺省为与此过程的主线程的堆栈大小相同。

lpSecurityAttrs

指向一个 SECURITY_ATTRIBUTES 结构，此结构指定线程的安全属性。

说明

此成员函数创建一个在调用过程的地址空间中运行的线程。使用 AfxBeginThread 可以一步创建线程对象并运行之。如果你需要在连续创建和线

程执行的终止之间重复使用线程对象，应使用 CreateThread。

请参阅 AfxBeginThread, CWinThread::CWinThread, ::CreateThread

CWinThread::CWinThread

```
CWinThread( );
```

说明

此函数构造一个 CWinThread 对象。要执行线程，请调用 CreateThread 成员函数。你通常通过调用 AfxBeginThread 来创建线程，AfxBeginThread 将调用此构造函数和 CreateThread 函数。

请参阅 CWinThread::CreateThread

CWinThread::ExitInstance

```
virtual int ExitInstance( );
```

返回值

是线程的退出码；值为 0 表示没有错误，值大于 0 表示有错误发生。通过调用 ::GetExitCode- Thread.可以查询到该值。

说明

框架通过很少被重载的 Run 成员函数调用此函数以退出线程的这个实例；或者当调用 InitInstance 失败时，调用此函数。

除了在 Run 成员函数内之外，不得在任何地方调用此成员函数。此成员函数仅被用户界面线程使用。

当 m_bAutoDelete 为真时，此函数的缺省实现删除 CWinThread 对象。如果你希望当线程终止时执行额外的清除工作，请重载此函数。当你的程序代码被执行之后，你的 ExitInstance 实现应调用基类的 ExitInstance 函数。

请参阅　CWinApp::ExitInstance

CWinThread::GetMainWnd

```
virtual CWnd * GetMainWnd( );
```

返回值

此函数返回指向一个窗口的指针，这个窗口为两类窗口中的一种。如果你的线程是一个 OLE 服务器的一部分并且拥有一个位于活动容器中的现场激活的对象，此函数返回 CWinThread 对象的 CWinApp::m_pActiveWnd 数据成员。

如果没有位于容器中的现场激活的对象或者你的应用程序不是 OLE 服务器，此函数返回线程对象的 m_pMainWnd 数据成员。

说明

如果你的应用程序为一个 OLE 服务器，调用此函数以得到应用程序的活动主窗口的指针，而不是直接引用应用程序对象的 m_pmainWnd 成员。对于用户界面线程，调用此函数等价于引用应用程序对象的 m_pActiveWnd 成员。

如果你的应用程序不是一个 OLE 服务器，则调用此函数等价于直接引用应用程序对象的 m_pMainWnd 成员。

重载此函数以修正缺省的行为。

请参阅 AfxGetMainWnd

CWinThread::GetThreadPriority

```
int GetThreadPriority( );
```

返回值

此函数返回当前线程在其优先级类中的优先级。此返回值应是下列从高到低列出的优先级值中的一个：

- THREAD_PRIORITY_TIME_CRITICAL
- THREAD_PRIORITY_HIGHEST
- THREAD_PRIORITY_ABOVE_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_LOWEST
- THREAD_PRIORITY_IDLE

关于这些优先级的进一步信息，参阅“Win32 SDK 程序员参考大全”第4卷中的函数::SetThreadPriority。

说明

此函数用于得到线程的当前线程优先级。

请参阅 CWinThread::SetThreadPriority, ::GetThreadPriority

CWinThread::InitInstance

virtual BOOL InitInstance( );

返回值

若初始化成功，返回值为非零；不成功则返回 0。

说明

InitInstance 必须被重载以初始化每个用户界面线程的新实例。统称，你重载 InitInstance 函数来执行当线程首次被创建时必须完成的任务。

此成员函数仅在用户界面线程中使用。工作者线程的初始化在传递给 AfxBeginThread 的控制函数中完成。

请参阅 CWinApp::InitInstance

CWinThread::IsIdleMessage

virtual BOOL IsIdleMessage(MSG* *pMsg*);

返回值

如果在处理消息之后调用了 OnIdle，则返回非零值；否则返回 0。

参数

pMsg

指向当前被处理的消息。

说明

如果避免在产生指定的消息以后调用 OnIdle 函数，则应重载这个函数。当产生重复的鼠标消息和闪烁光标消息之后，这个函数的缺省实现并不调用 OnIdle。

如果应用程序创建了一个短的定时器，OnIdle 将被频繁调用，导致性能上的问题。为了改善这种应用程序的性能，重载应用程序的 CWinApp 派生类中的 IsIdleMessage 消息，按照如下方式检测 WM_TIMER 消息：

```
BOOL CMyApp::IsIdleMessage( MSG* pMsg )
{
	if (!CWinApp::IsIdleMessage( pMsg ) ||
		pMsg->message == WM_TIMER)
		return FALSE;
	else
		return TRUE;
}
```

按照这种方式处理 WM_TIMER 消息可以改善那些使用短定时器的应用程序的

性能。

CWinThread::OnIdle

virtual BOOL OnIdle(LONG *lCount*);

返回值

如果需要更多的空闲处理时间，则返回非零值；如果不需要，则返回0。

参数

lCount

该参数是一个计数值，当应用程序的消息队列为空，OnIdle函数被调用时，该计数值就增加1。每当一条新消息被处理时，该计数值就被复位为0。你可以使用lCount参数来确定应用程序不处理消息时空闲时间的相对长度。

说明

如果要执行空闲时处理，则重载这个成员函数。当应用程序的消息队列为空时，OnIdle就在缺省的消息循环中被调用。你可以用重载函数来调用自己的后台空闲处理任务。

OnIdle应返回0以表明不需要更多的空闲处理时间。当消息队列为空时，OnIdle每被调用一次lCount参数就增加，而每处理一条新消息lCount就被复位为0。你可以根据这个计数值调用不同的空闲处理例程。

这个成员函数的缺省实现从内存中释放临时对象和不用的动态链接库。

这个成员函数仅在用户界面线程中使用。
由于应用程序在 OnIdle 返回之后才能处理消息，因此不应在这个函数中执行较长的操作。

请参阅 CWinApp::OnIdle

CWinThread::PostThreadMessage

BOOL PostThreadMessage(UINT message , WPARAM *wParam*, LPARAM *lParam*);

返回值

如果成功，则返回非零值；否则返回 0。

参数

message

用户自定义消息的 ID。

wParam

第一个消息参数。

lParam

第二个消息参数。

说明

调用这个函数以向其它 CWinThread 对象发送一个用户自定义消息。发送的消息通过消息映射宏 ON_THREAD_MESSAGE 被映射到适当的消息处理函数。

请参阅 ON_THREAD_MESSAGE

CWinThread::PreTranslateMessage

virtual BOOL PreTranslateMessage(MSG **pMsg*);

返回值

如果消息在 PreTranslateMessage 中已经被完全处理，不需要再进一步处理，则返回非零值。如果消息需要按通常方式进一步处理，则返回零。

参数

pMsg

指向包含了要处理的消息的 MSG 结构。

说明

如果要在消息被分派到 Windows 函数::TranslateMessage 和::DispatchMessage 之前过滤 Windows 消息，则应重载这个函数。

这个函数仅在用户界面线程中使用。

请参阅 CWinApp::PreTranslateMessage

CWinThread::ProcessMessageFilter

virtual BOOL ProcessMessageFilter(int *code*, LPMSG *lpMsg*);

返回值

如果消息被处理了，则返回非零值；否则返回 0。

参数

code

指定了钩子代码。这个成员函数使用这些代码来确定如何处理 lpMsg。

lpMsg

指向 Windows 的 MSG 结构的指针。

说明

框架的钩子函数调用这个函数以过滤并响应特定的 Windows 函数。钩子函数在事件被发送到应用程序的正常消息处理机制之前处理这些事件。

如果你改变了这种高级特性，确保你调用了基类的相应函数以维持框架的钩子处理。

请参阅 MessageProc, WH_MSGFILTER

CWinThread::ProcessWndProcException

virtual LRESULT ProcessWndProcException(CException **e*, const MSG **pMsg*);

返回值

如果产生了一个WM_CREATE异常，则返回-1；否则返回0。

参数

e

指向没有被处理的异常。

pMsg

指向一个MSG结构，其中包含了与导致框架出现异常的Windows消息有关的信息。

说明

每当处理函数没有捕获你的线程消息或命令处理函数所抛出的异常时，框架就调用这个成员函数。

不要直接调用这个成员函数。

这个成员函数的缺省实现仅处理下列消息产生的异常：

命令	动作
WM_CREATE	失败
WM_PAINT	使涉及的窗口有效，防止产生另一个WM_PAINT消息

重载这个成员函数以提供对异常的全局处理。仅当你希望执行缺省的动作时才调用基类的函数。

这个成员函数仅在具有消息收发功能的线程中使用。

请参阅 CWinApp::ProcessWndProcException

CWinThread::ResumeThread

```
DWORD ResumeThread( );
```

返回值

如果成功，则返回线程的原挂起计数值；否则返回 0xFFFFFFFF。如果返回值为零，则表示当前线程没有被挂起。如果返回值为 1，线程被挂起，但是即将重新启动。任何大于 1 的返回值都表明线程将继续挂起。

说明

调用这个函数以使被 SuspendThread 成员函数所挂起的线程恢复执行，或者使用 CREATE_SUSPENDED 标志创建的线程恢复执行。当前线程的挂起计数被减小 1。如果挂起计数被减小到 0，线程将恢复执行；否则线程继续被挂起。

请参阅 CWinThread::SuspendThread, ::ResumeThread

CWinThread::Run

virtual int Run( );

返回值

线程返回的一个整数值。这个值可以通过调用::GetExitCodeThread来获得。

说明

这个函数为用户界面线程提供了缺省的消息循环。Run接收并分派Windows消息，直到它接收到一个WM_QUIT消息。如果线程的当前消息队列中不包含消息，Run就调用OnIdle以执行空闲处理。接收到的消息被送到PreTranslateMessage成员函数以进行特殊处理，然后被发送到Windows函数::TranslateMessage以进行标志键盘转换。最后，调用Windows函数::DispatchMessage。

Run函数很少被重载，但是你可以重载它以提供特殊的功能。

这个成员函数仅在用户界面线程中使用。

请参阅　CWinApp::Run

CWinThread::SetThreadPriority

BOOL SetThreadPriority(int nPriority);

返回值

如果这个函数成功地执行，则返回非零值；否则返回 0。

参数

nPriority

指定了线程在其优先权类中的新优先权级。这个参数必须是下列值之一，从最高优先权到最低：

- THREAD_PRIORITY_TIME_CRITICAL
- THREAD_PRIORITY_HIGHEST
- THREAD_PRIORITY_ABOVE_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_LOWEST
- THREAD_PRIORITY_IDLE

有关这些优先权的更多信息参见“ Win32 SDK **程序员参考”第四卷中**

的::SetThreadPriority。

说明

这个函数设置当前线程在它所属的优先权类中的优先权级。它只能在CreateThread成功返回之后调用。

请参阅 CWinThread::GetThreadPriority, ::SetThreadPriority

CWinThread::SuspendThread

DWORD SuspendThread( );

返回值

如果成功，则返回线程原来的挂起计数值；否则返回0xFFFFFFFF。

说明

增加当前线程的挂起计数。如果线程的挂起计数大于零，则该线程将不被执行。线程可以通过调用ResumeThread成员函数恢复执行。

请参阅 CWinThread::ResumeThread, ::SuspendThread

数据成员

CWinThread::m_bAutoDelete

参数

指定当线程结束的时候，是否要自动删除 CWinThread **对象。**m_bAutoDelete **是** BOOL **类型的公有变量。**

CWinThread::m_hThread

说明

与 CWinThread **相关联的线程的句柄。**m_hThread **数据成员是** HANDLE **类型的公有变量。它仅当线程存在时才有效。**

CWinThread::m_nThreadID

说明

与 CWndThread 相关联的线程的句柄。m_nThreadID 数据成员是 DWORD 类型的公有变量。它仅当线程存在时才有效。

CWinThread::m_pActiveWnd

说明

使用这个数据成员保存线程的当前活动窗口对象的指针。当 m_pActiveWnd 所代表的窗口被关闭时，微软基础类库将自动终止你的线程。如果该线程是应用程序的主线程，则应用程序将结束。如果这个数据成员为 NULL，则应用程序的 CWinApp 对象的活动窗口将被继承。m_pActiveW nd 是 CWnd*类型的公有变量。

通常，你在重载 InitInstance 的时候设置这个成员变量。在工作者线程中，这个数据成员的值是从它的父线程中继承来的。

请参阅 CWinThread::InitInstance, CWinThread::m_pMainWnd

CWinThread::m_pMainWnd

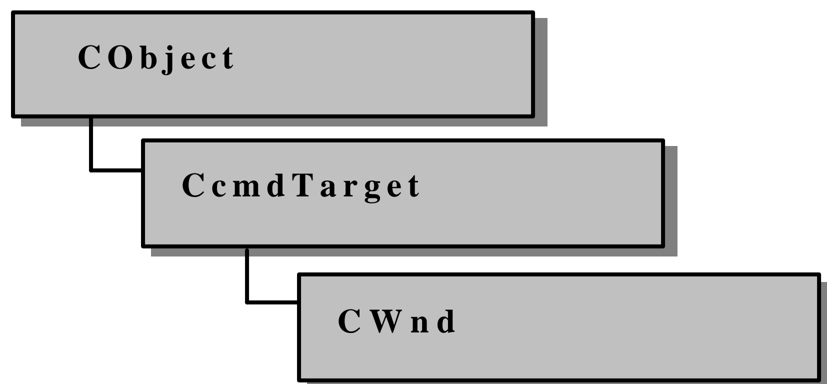
说明

使用这个数据成员来保存你的线程的主窗口对象的指针。当 m_pMainWnd 所代表的窗口被关闭时，微软基础类库将自动终止线程。如果该线程是应用程序的主线程，则应用程序将结束。如果这个数据成员为 NULL，则应用程序的 CWinApp 对象的主窗口将被用来确定什么时候终止线程。m_pMainWnd 是 CWnd* 类型的公有变量。

通常，你在重载 InitInstance 的时候设置这个成员变量。在工作者线程中，这个数据成员的值是从它的父线程继承的。

请参阅 CWinThread::InitInstance

CWnd



CWnd 类提供了微软基础类库中所有窗口类的基本功能。

CWnd 对象与 Windows 的窗口不同，但是两者有紧密联系。CWnd 对象是由 CWnd 的构造函数和析构函数创建或销毁的。另一方面，Windows 的窗口是 Windows 的一种内部数据结构，它是由 CWnd 的 Create 成员函数创建的，而由 CWnd 的虚拟析构函数销毁。DestroyWindow 函数销毁 Windows 的窗口，但是不销毁对象。

CWnd 类和消息映射机制隐藏了 WndProc 函数。接收到的 Windows 通知消息通过消息映射被自动发送到适当的 CWnd OnMessage 成员函数。你可以在派生

类中重载OnMessage成员函数以处理成员的特定消息。

CWnd类同时还使你能够为应用程序创建Windows的子窗口。先从CWnd继承一个类，然后在派生类中加入成员变量以保存与你的应用程序有关的数据。在派生类中实现消息处理成员函数和消息映射，以指定当消息被发送到窗口时应该如何动作。

你可以经过两个步骤来创建一个子窗口。首先，调用构造函数CWnd以创建一个CWnd对象，然后调用Create成员函数以创建子窗口并将它连接到CWnd对象。

当用户关闭你的子窗口时，应销毁CWnd对象，或者调用DestroyWindow成员函数以清除窗口并销毁它的数据结构。

在微软基础类库中，从CWnd派生了许多其它类以提供特定的窗口类型。这些类中有许多，包括CFrameWnd，CMDIFrameWnd，CMDIChildWnd，CView和CDialog，被用来进一步派生。从CWnd派生的控件类，如CButton，可以被直接使用，也可以被进一步派生出其它类来。

```
#include <afxwin.h>
```

请参阅　CFrameWnd, CView

CWnd 类成员

初始化	对话框项函数	初始化消息处理函数
窗口状态函数	数据绑定函数	系统消息处理函数
窗口大小和位置	菜单函数	一般消息处理函数
窗口访问函数	工具提示函数	控件消息处理函数
更新/绘图函数	定时器函数	输入消息处理函数
坐标映射函数	警告函数	非客户区消息处理函数
窗口文本函数	窗口消息函数	MDI 消息处理函数
滚动函数	剪贴板函数	剪贴板消息处理函数
拖放函数	OLE 控件	菜单循环通知
插字符号函数	可重载函数	

Data Members

m_hWnd	指明与这个 CWnd 对象相关联的 HWND 句柄

Construction/Destruction

CWnd	构造一个 CWnd 对象
DestroyWindow	销毁相关联的 Windows 窗口

Initialization	
Create	创建并初始化与 CWnd 对象相关联的子窗口
PreCreateWindow	在与 CWnd 对象相关联的窗口被创建之前调用
CalcWindowRect	调用这个函数以计算窗口客户区的矩形
GetStyle	返回当前的窗口风格
GetExStyle	返回窗口的扩展风格
Attach	将 Windows 句柄与 CWnd 对象相连接
Detach	将一个 Windows 句柄从 CWnd 对象上分离并返回这个句柄
PreSubclassWindow	在调用 SubclassWindow 之前，允许其它必要的子类化工作
SubclassWindow	将窗口与 CWnd 对象相连接，并使它通过 CWnd 的消息映射转发消息
UnsubclassWindow	将窗口与 CWnd 对象分离
FromHandle	当给定一个窗口的句柄时，返回 CWnd 对象的指针。如果没有 CWnd 对象与这个句柄相连接，则创建一个临时的 CWnd 对象并与之相连接
FromHandlePermanent	当给定一个窗口的句柄时，返回 CWnd 对象的指针。如果没有 CWnd 对象与这个句柄相连接，则返回 NULL
DeleteTempMap	CWinApp 的空闲处理函数自动调用这个函数，清除由 FromHandle 创建的任何临时 CWnd 对象

续表

GetSafeHwnd	返回m_hWnd，如果该指针为NULL，则返回NULL
CreateEx	创建一个Windows的可重叠窗口、弹出窗口或子窗口，并把它连接到一个CWnd对象上
CreateControl	创建一个OLE控件，该控件在MFC程序中由一个CWnd对象代表

Window State Funtions

IsWindowEnabled	确定一个窗口是否允许鼠标和键盘输入
EnableWindow	允许或禁止鼠标和键盘输入
GetActiveWindow	获得激活的窗口
SetActiveWindow	激活窗口
GetCapture	获得捕获鼠标的CWnd
SetCapture	使随后的鼠标输入都被发送到这个CWnd
GetFocus	获得当前具有输入焦点的CWnd
SetFocus	要求输入焦点
GetDesktopWindow	获得Windows的桌面窗口
GetForegroundWindow	返回前台窗口的指针（顶层窗口，用户正在其中工作）
SetForegroundWindow	使创建窗口的线程变为前台并激活窗口
GetIcon	获得图标的句柄

续表

SetIcon	设置指定图标的句柄
GetWindowContextHelpId	获得帮助上下文的标识符
SetWindowContextHelpId	设置帮助上下文的标识符
ModifyStyle	修改当前的窗口风格
ModifyStyleEx	修改窗口的扩展风格

WindowSize and Position

GetWindowPlacement	获得窗口的显示状态以及正常（还原）的、最大化和最小化的位置
SetWindowPlacement	设置窗口的显示状态以及正常（还原）的、最大化和最小化的位置
GetWindowRgn	获得窗口的窗口区域的一个拷贝
SetWindowRgn	设置窗口的区域
IsIconic	确定 CWnd 是否被最小化（图标化）
IsZoomed	确定 CWnd 是否被最大化
MoveWindow	改变 CWnd 的位置和大小
SetWindowPos	改变子窗口、弹出窗口和顶层窗口的大小、位置以及顺序
ArrangeIconicWindows	排列所有最小化（图标化）的子窗口
BringWindowToTop	使 CWnd 到达重叠窗口堆栈的顶部
GetWindowRect	获得 CWnd 的屏幕坐标

续表

GetClientRect	获得 CWnd 客户区域的大小
WindowAccess Functions	
ChildWindowFromPoint	确定哪个子窗口包含指定的点，如果有的话
FindWindow	返回由窗口名或窗口类标识的窗口的句柄
GetNextWindow	返回窗口管理器列表中的下一个（或前一个）窗口
GetOwner	获得 CWnd 的拥有者的指针
SetOwner	改变 CWnd 的拥有者
GetTopWindow	返回 CWnd 所属的第一个子窗口
GetWindow	返回与窗口有特定关系的窗口
GetLastActivePopup	确定 CWnd 拥有的弹出窗口是否最近被激活
IsChild	指明 CWnd 是指定窗口的一个子窗口还是其它子对象
GetParent	获得 CWnd 的父窗口（如果有）
GetSafeOwner	获得给定窗口的安全拥有者
SetParent	改变父窗口
WindowFromPoint	标明包含给定点的窗口
GetDlgItem	获得指定的对话框中具有指定 ID 的控件
GetDlgCtrlID	如果 CWnd 是一个子窗口，则用这个函数来返回它的 ID 值

续表

SetDlgCtrlID	为窗口（可以是任意子窗口，不仅是对话框中的控件）设置窗口 ID 或控件 ID
GetDescendantWindow	搜索所有子窗口并返回具有指定 ID 的窗口
GetParentFrame	获得 CWnd 对象的父框架窗口
SendMessageToDescendants	将消息发送给窗口的所有子窗口
GetTopLevelParent	获得窗口的顶层父窗口
GetTopLevelOwner	获得顶层窗口
GetParentOwner	返回子窗口的父窗口的指针
GetTopLevelFrame	获得窗口的顶层框架窗口
UpdateDialogControls	调用这个函数以更新对话框按钮和其它控件的状态
UpdateData	初始化对话框或获得对话框中的数据
CenterWindow	将窗口设置到父窗口的中央

Update/Painting Functions

BeginPaint	为绘图准备 CWnd
EndPaint	标志着绘图结束
Print	在指定的设备环境中画出当前窗口
PrintClient	在指定的设备环境（通常是打印机设备环境）中画出任何窗口
LockWindowUpdate	禁止或恢复在给定窗口中的绘图

续表

UnlockWindowUpdate	解锁用 CWnd::LockWindowUpdate 锁定的窗口
GetDC	获得客户区的设备环境
GetDCEx	获得客户区的显示设备环境，允许在绘图的时候进行剪裁
RedrawWindow	更新客户区中的指定矩形或区域
GetWindowDC	获得整个窗口的显示环境，包括标题条、菜单和滚动条
ReleaseDC	是否客户和窗口设备环境，使其它应用程序能够使用它们
UpdateWindow	更新客户区
SetRedraw	使 CWnd 的变化能够被重画，或者禁止重画发生的变化
GetUpdateRect	获得完全封闭了 CWnd 中更新区域的最小矩形的坐标
GetUpdateRgn	获得 CWnd 的更新区域
Invalidate	使整个客户区无效
InvalidateRect	在当前的更新区域中加入给定的矩形，使客户区的给定矩形无效
InvalidateRgn	在当前的更新区域中加入给定的区域，使客户区的给定区域无效

续表

ValidateRect	在当前的更新区域中删除给定的矩形，使客户区的给定矩形有效
ValidateRgn	在当前的更新区域中删除给定的区域，使客户区的给定区域有效
ShowWindow	显示或隐藏窗口
IsWindowVisible	确定窗口是否可见
ShowOwnedPopups	显示或隐藏该窗口拥有的所有弹出窗口
EnableScrollBar	允许或禁止滚动条的一个或两个箭头

CoordinateMapping Functions

MapWindowPoints	将一些坐标从 CWnd 的坐标空间转换（映射）到其它窗口的坐标空间
ClientToScreen	将给定点或显示器上矩形的客户区坐标转换为屏幕坐标
ScreenToClient	将给定点或显示器上矩形的屏幕坐标转换为客户坐标

Window Text Functions

SetWindowText	将窗口的文本或标题文字（如果有）设为指定的文本
GetWindowText	返回窗口的文本或标题文字（如果有）

续表

GetWindowTextLength	返回窗口文本或标题文字的长度
SetFont	设置当前字体
GetFont	获得当前字体

Scrolling Functions	
GetScrollPos	获得滚动块的当前位置
GetScrollRange	复制指定滚动条的最大和最小滚动条位置
ScrollWindow	滚动客户区的内容
ScrollWindowEx	滚动客户区的内容，与 ScrollWindow 类似，还有其它功能
GetScrollInfo	获得 SCROLLINFO 结构维护的滚动条信息
GetScrollLimit	获得滚动条的限制
SetScrollInfo	设置滚动条信息
SetScrollPos	设置滚动块的当前位置，如果指定，并重画滚动条以反映新的位置
SetScrollRange	设置指定滚动条的最小和最大位置值
ShowScrollBar	显示或隐藏滚动条
EnableScrollBarCtrl	允许或禁止一个滚动条控件
GetScrollBarCtrl	返回滚动条控件
RepositionBars	重新设定客户区中控制条的位置

Drag-Drop Functions

DragAcceptFiles	指明该窗口将接收拖来的文件

Caret Functions

CreateCaret	为系统插字符创建一个新的形状并获得它的所有权
CreateSolidCaret	为系统插字符创建一个实心块并获得它的所有权
CreateGrayCaret	为系统插字符创建一个灰色块并获得它的所有权
GetCaretPos	获得插字符当前位置的客户区坐标
SetCaretPos	将插字符移动到指定的位置
HideCaret	通过将插字符移出显示屏幕而隐藏它
ShowCaret	在插字符的当前位置显示插字符。一旦被显示，插字符就会自动闪烁

Dialog-BoxItem Functions

CheckDlgButton	在按钮控件旁放置或者清除检查标记
CheckRadioButton	选中指定的单项按钮并清除指定的按钮组中其它所有单项按钮的检查标记
GetCheckedRadioButton	返回一组按钮中当前选中的单项按钮的 ID
DlgDirList	用文件或目录列表填充一个列表框
DlgDirListComboBox	用文件或目录列表填充一个组合框中的列表框
DlgDirSelect	获得列表框的当前选择
DlgDirSelectComboBox	获得组合框中列表框的当前选择

续表

GetDlgItemInt	将给定对话框中控件的文本转换为整数
GetDlgItemText	获得与控件相关的标题或文本
GetNextDlgGroupItem	在一组控件内搜索下一个（或上一个）控件
GetNextDlgTabItem	获得指定的控件之后（或之前）具有WS_TABSTOP风格的第一个控件
IsDlgButtonChecked	确定按钮控件是否有检查标记
IsDialogMessage	确定给定的消息是否是送往无模式对话框的，如果是，则处理它
SendDlgItemMessage	向指定的控件发送一条消息
SetDlgItemInt	将控件的文本设为代表一个整数的字符串
SetDlgItemText	设置指定的对话框中的控件的标题或文本
SubclassDlgItem	将一个Windows控件与CWnd对象连接，然后使它通过CWnd的消息映射转发消息
ExecuteDlgInit	初始化对话框资源
RunModalLoop	获得、转换或分派来自模式窗口的消息
ContinueModal	继续一个窗口的模式状态
EndModalLoop	结束一个窗口的模式状态

Data-Binding Functions

BindDefaultProperty	像类型库中标记的那样，将调用对象的缺省的简单移动属性和与数据源控件相关的游标绑定在一起
BindProperty	将一个游标移动属性与数据移动控件绑定在一起，然后在 MFC 的绑定管理器中注册这种联系
GetDSCCursor	获得游标的指针，该游标是通过数据源控件的 DataSource、UserName、Password 和 SQL 属性定义的

Menu Functions

GetMenu	获得指定菜单的指针
SetMenu	将菜单设为指定的菜单
DrawMenuBar	重画菜单条
GetSystemMenu	允许应用程序访问控制菜单，用于拷贝和修改
HiliteMenuItem	加亮显示一个顶层（菜单条）菜单项，或者取消加亮显示

ToolTip Functions

EnableToolTips	允许工具提示控件
CancelToolTips	禁止工具提示控件

续表

FilterToolTipMessage	获得与对话框中控件相关的标题和文本
OnToolHitTest	确定一个点是否位于指定工具的边界矩形之中并获得工具的信息

Timer Functions

SetTimer	安装一个系统定时器，当它被激活时，发送一个WM_TIMER消息
KillTimer	销毁一个系统定时器

Alert Functions

FlashWindow	使窗口闪烁一次
MessageBox	创建并显示一个窗口，其中包含了应用程序提供的消息和标题

WindowMessage Functions

GetCurrentMessage	返回这个窗口当前处理的消息的指针。只应该在OnMessage消息处理函数内部调用
Default	调用缺省的窗口过程，它提供了对应用程序没有处理的任何窗口消息的缺省处理
PreTranslateMessage	在消息被发送到Windows函数TranslateMessage和DispatchMessage

续表

	之前，CWinApp 使用这个函数来过滤窗口消息
SendMessage	向 CWnd 对象发送一个消息，直到这条消息被处理之后才返回
PostMessage	将一条消息放入应用程序的消息队列，然后不等窗口处理这条消息直接返回
SendNotifyMessage	将一条消息发送到窗口并尽快返回，返回的速度取决于该窗口是否是由调用线程所创建

Clipboard Functions

ChangeClipboardChain	将 CWnd 从剪贴板观察器的链中清除
SetClipboardViewer	将 CWnd 加入一个窗口链，每当剪贴板的内容发生变化时，就会通知这些窗口
OpenClipboard	打开剪贴板。其它应用程序将不能修改剪贴板，直到调用 Windows 的 CloseClipboard 函数
GetClipboardOwner	获得指向剪贴板的当前拥有者的指针
GetOpenClipboardWindow	获得当前打开剪贴板的窗口的指针
GetClipboardViewer	获得剪贴板观察器链中的第一个窗口的指针

OLE Controls

SetProperty	设置 OLE 控件的属性
OnAmbientProperty	实现周围属性值
GetControlUnknown	获得指向未知 OLE 控件的指针
GetProperty	获得 OLE 控件的属性
InvokeHelper	激活 OLE 控件的方法或属性

Overridables

WindowProc	为 CWnd 对象提供了窗口过程。缺省的窗口过程通过消息映射分派消息
DefWindowProc	调用缺省的窗口过程，它提供了对应用程序没有处理的任何窗口消息的缺省处理
PostNcDestroy	这个虚拟函数在窗口被销毁以后被缺省的 OnNcDestroy 函数所调用
OnNotify	框架调用这个函数以通知父窗口，在它的一个控件发生了一个事件，或该控件需要消息
OnChildNotify	父窗口调用这个函数，给被通知的控件一个响应控件通知消息的机会
DoDataExchange	用于对话框数据交换和校验。由 UpdateData 调用

Initialization Message Handlers	
OnInitMenu	当菜单要被激活时调用这个函数
OnInitMenuPopup	当弹出菜单要被激活时调用这个函数

System Message Handlers	
OnSysChar	当一次击键被转换为系统字符消息时调用这个函数
OnSysCommand	当用户从控制菜单中选择命令，或者当用户选择了最大化或最小化按钮时，调用这个函数
OnSysDeadChar	当一次击键被转换为系统死键（例如重音字符）消息时调用这个函数
OnSysKeyDown	当用户按住 ALT 键并按下其它键时调用这个函数
OnSysKeyUp	当用户放开一个键，而此时 ALT 键被按下，则调用这个函数
OnCompacting	当 Windows 检测到系统内存很少时，就调用这个函数
OnDevModeChange	当用户改变了设备模式设置时，就为顶层窗口调用这个函数
OnFontChange	当字体资源池发生变化时调用这个函数

续表

OnPaletteIsChanging	当应用程序将要实现其逻辑调色板时通知其它应用程序
OnPaletteChanged	调用这个函数时使用调色板的所有窗口能够实现它们的逻辑调色板并更新它们的客户区
OnSysColorChange	当系统颜色设置发生改变时为所有的顶层窗口调用这个函数
OnWindowPosChanging	由于调用了 SetWindowPos 函数或其它的窗口管理函数，因而窗口的大小、位置和次序将要发生变化时，就调用这个函数
OnWindowPosChanged	由于调用了 SetWindowPos 函数或其它的窗口管理函数，因而当窗口的大小、位置和次序发生了变化时，就调用这个函数
OnDropFiles	当用户在注册为可以接收拖放文件的窗口上方释放鼠标左键时，这个函数就被调用
OnSpoolerStatus	每当一个作业被加入或移出打印管理器的队列时，打印管理器就调用这个函数
OnTimeChange	在系统时间改变之后，为所有的顶层窗口调用这个函数
OnWinIniChange	在 Windows 的初始化文件 WIN.INI 改变之后为所有的顶层窗口调用这个函数

General Message Handlers	
OnCommand	当用户选择了一个命令时调用这个函数
OnActivate	当 CWnd 要被激活或退出激活状态时调用这个函数
OnActivateApp	当应用程序要被激活或退出激活状态时调用这个函数

General Message Handlers	
OnCancelMode	调用这个函数以允许 CWnd 取消任何内部模式，比如鼠标捕获状态
OnChildActivate	当 CWnd 的大小和位置发生变化或者 CWnd 被激活时，就为多文档界面（MDI）的子窗口调用这个函数
OnClose	调用这个函数，作为关闭 CWnd 的信号
OnCopyData	从一个应用程序复制数据到另一个应用程序
OnCreate	作为窗口创建过程的一部分来调用
OnCtlColor	如果 CWnd 是一个控件的父窗口，当控件要被重画时就调用这个函数
OnDestroy	当 CWnd 要被销毁时就调用这个函数
OnEnable	当 CWnd 被允许或禁止时调用这个函数
OnEndSession	当会话将要结束时调用这个函数

续表

OnEnterIdle	调用这个函数以通知应用程序的主窗口过程，模式对话框或菜单正在进入空闲状态
OnEraseBkgnd	当需要擦除窗口的背景时调用这个函数
OnGetMinMaxInfo	每当 Windows 需要知道最大化的位置和大小，或者最小或最大跟踪尺寸时，就调用这个函数
OnIconEraseBkgnd	当 CWnd 被最小化（图标化），并且在画出图标之前，必须填充图标的背景时调用这个函数
OnKillFocus	当 CWnd 失去输入焦点时立即调用这个函数
OnMenuChar	当用户按下一个菜单助记字符，但是不能与当前菜单中任何预定义的助记符相匹配时，就调用这个函数
OnMenuSelect	当用户选择了一个菜单项时就调用这个函数
OnMove	当 CWnd 的位置发生变化时调用这个函数
OnMoving	指明用户正在移动 CWnd 对象
OnDeviceChange	通知应用程序或设备驱动程序，设备或计算机的硬件配置发生了变化
OnStyleChanged	指明 Windows 的::SetWindowLong 函数已经改变了一个或多个窗口风格

续表

OnStyleChanging	指明Windows的::SetWindowLong函数将要改变一个或多个窗口风格
OnPaint	调用这个函数以重画窗口的一部分
OnParentNotify	当创建或销毁一个子窗口，或者当用户在子窗口上方点击了鼠标键时调用这个函数
OnQueryDragIcon	当一个最小化（图标化）的CWnd要被用户拖拉时就调用这个函数
OnQueryEndSession	当用户选择结束Windows会话时就调用这个函数
OnQueryNewPalette	通知CWnd它将接收输入焦点
OnQueryOpen	当CWnd是一个图标并且用户请求打开这个图标时就调用这个函数

General Message Handlers	
OnSetFocus	当CWnd获得输入焦点时调用这个函数
OnShowWindow	当CWnd被隐藏或显示时调用这个函数
OnSize	当CWnd的大小被改变以后调用这个函数
OnSizing	指明用户正在改变矩形的大小
OnStyleChanged	指明窗口的一个或多个风格已经被改变
OnStyleChanging	指明窗口的一个或多个风格将被改变

Control Message Handlers

OnCharToItem	这个函数被具有 LBS_WANTKEYBOARDINPUT 风格的子列表框调用，用以响应 WM_CHAR 消息
OnCompareItem	调用这个函数以确定排序的自画组合框或列表框中新项的相对位置
OnDeleteItem	当一个自画子列表框或组合框中将被销毁时，或者从控件中删除项时调用这个函数
OnDrawItem	当自画子按钮控件、组合框控件、列表框控件或菜单的可视部分需要被画出时调用这个函数
OnDSCNotify	在响应数据源控件引发的事件时调用，该事件是当与数据源控件绑定的控件修改或将要修改游标时产生的
OnGetDlgCode	为控件调用这个函数，使控件能够自己处理输入的箭头键和 TAB 键
OnMeasureItem	当控件被创建时为自画子组合框，列表框或菜单项调用这个函数。CWnd 通知 Windows 该控件的大小
SendChildNotifyLastMsg	提供了从父窗口到子窗口的通知消息，使子窗口能够处理一个任务
ReflectChildNotify	将消息向它的来源反映的帮助函数

续表

OnWndMsg	指明一个窗口消息是否已被处理
ReflectLastMsg	将上一个消息反映到子窗口
OnVKeyToItem	由 CWnd 所拥有的列表框调用，用于响应 WM_KEYDOWN 消息

Input Message Handlers

OnChar	当一次击键被转换为非系统字符时调用这个函数
OnDeadChar	当一次击键被转换为非系统死键（例如重音字符）时调用这个函数
OnHScroll	当用户点击了 CWnd 的水平滚动条时调用这个函数
OnKeyDown	当按下了一个非系统键时调用这个函数
OnKeyUp	当放开一个非系统键时调用这个函数
OnLButtonDblClk	当用户双击鼠标左键时调用这个函数

Input Message Handlers

OnLButtonDown	当用户按下鼠标左键时调用这个函数
OnLButtonUp	当用户放开鼠标左键时调用这个函数
OnMButtonDblClk	当用户双击鼠标中键时调用这个函数
OnMButtonDown	当用户按下鼠标中键时调用这个函数

续表

OnMButtonUp	当用户放开鼠标中键时调用这个函数
OnMouseActivate	当鼠标位于非活动窗口，并且用户按下鼠标键时调用这个函数
OnMouseMove	当鼠标光标移动时调用这个函数
OnMouseWheel	当用户旋转鼠标轮时调用这个函数。使用 WindowsNT4.0 的消息处理
OnRegisteredMouseWheel	当用户旋转鼠标轮的时候调用这个函数。使用 Windows 95 和 Windows NT 3.51 的消息处理
OnRButtonDblClk	当用户双击鼠标右键时调用这个函数
OnRButtonDown	当用户按下鼠标右键时调用这个函数
OnRButtonUp	当用户放开鼠标右键时调用这个函数
OnSetCursor	如果没有捕获鼠标输入并且鼠标导致光标在窗口内移动时，就调用这个函数
OnTimer	当达到 SetTimer 指定的时间间隔时调用这个函数
OnVScroll	当用户点击窗口的垂直滚动条时调用这个函数
OnCaptureChanged	向失去鼠标捕获的窗口发送一条消息

Nonclient-Area Message Handlers

OnNcActivate	当需要改变非客户区以指明活动或非活动状态时调用这个函数

续表

OnNcCalcSize	当需要计算非客户区的大小和位置时调用这个函数
OnNcCreate	在 OnCreate 之前，当要创建非客户区时调用这个函数
OnNcDestroy	当非客户区要被销毁的时候调用这个函数
OnNcHitTest	如果 CWnd 中包含了光标，或者用 SetCapture 捕获了鼠标输入时，每当鼠标移动时，Windows 调用这个函数
OnNcLButtonDblClk	当光标位于 CWnd 的非客户区，用户双击鼠标左键时，就调用这个函数
OnNcLButtonDown	当光标位于 CWnd 的非客户区，用户按下鼠标左键时，就调用这个函数
OnNcLButtonUp	当光标位于 CWnd 的非客户区，用户放开鼠标左键时，就调用这个函数
OnNcMButtonDblClk	当光标位于 CWnd 的非客户区，用户双击鼠标中键时，就调用这个函数
OnNcMButtonDown	当光标位于 CWnd 的非客户区，用户按下鼠标中键时，就调用这个函数
OnNcMButtonUp	当光标位于 CWnd 的非客户区，用户放开鼠标中键时，就调用这个函数

Nonclient-Area Message Handlers

OnNcMouseMove	当光标在 CWnd 的非客户区中移动时就调用这个函数
OnNcPaint	当非客户区需要重画时调用这个函数
OnNcRButtonDblClk	当光标位于 CWnd 的非客户区，用户双击鼠标右键时，就调用这个函数
OnNcRButtonDown	当光标位于 CWnd 的非客户区，用户按下鼠标右键时，就调用这个函数
OnNcRButtonUp	当光标位于 CWnd 的非客户区，用户放开鼠标右键时，就调用这个函数

MDI Message Handlers

OnMDIActivate	当 MDI 子窗口被激活或失去活动状态时调用这个函数

Clipboard Message Handlers

OnAskCbFormatName	当剪贴板的拥有者将显示剪贴板内容时，剪贴板观察程序就调用这个函数
OnChangeCbChain	通知指定的窗口将从链中删除
OnDestroyClipboard	当通过 Windows 的 EmptyClipboard 函数清空剪贴板时调用这个函数
OnDrawClipboard	当内容变化时调用这个函数

续表

OnHScrollClipboard	当剪贴板的拥有者要滚动剪贴板的图像、使适当的部分无效以及更新滚动条值的时候调用这个函数
OnPaintClipboard	当剪贴板观察器的客户区需要重画的时候调用这个函数
OnRenderAllFormats	当拥有者应用程序将被销毁且需要提交它的所有格式时调用这个函数
OnRenderFormat	当一种延迟提交的格式需要被提交时为剪贴板拥有者调用这个函数
OnSizeClipboard	当剪贴板观察器窗口的客户区大小发生变化时调用这个函数
OnVScrollClipboard	当拥有者要滚动剪贴板的图像、使适当的部分无效以及更新滚动条值的时候调用这个函数

Menu Loop Notification

OnEnterMenuLoop	进入一个菜单模式的循环时调用该函数
OnExitMenuLoop	当退出一个菜单模式的循环时调用该函数

成员函数

CWnd::ArrangeIconicWindows

UINT ArrangeIconicWindows( );

返回值

如果这个函数执行成功，则返回一行图标的高度；否则返回0。

说明

排列所有最小化（图标化）的子窗口。

这个成员函数还排列桌面窗口中的图标，这些图标覆盖了整个屏幕。GetDesktopWindow成员函数可以返回桌面窗口对象的图标。

如果要排列MDI客户窗口中的MDI子窗口，则调用CMDIFrameWnd::MDIIconArrange。

请参阅 CWnd::GetDesktopWindow，CMDIFrameWnd::MDIIconArrange，::ArrangeIconicWindows

CWnd::Attach

BOOL Attach(HWND *hWndNew*);

返回值

如果成功，则返回非零值；否则返回 0。

参数

hWndNew

指定了 Windows 窗口的句柄。

说明

将一个 Windows 窗口与 CWnd 对象相连接。

请参阅 CWnd::Detach, CWnd::m_hWnd, CWnd::SubclassWindow

CWnd::BeginPaint

CDC* BeginPaint(LPPAINTSTRUCT *lpPaint*);

返回值

标识了 CWnd 的设备环境。这个指针可能是临时的，不应在 EndPaint 之外保存。

参数

lpPaint

执行 PAINTSTRUCT 结构，用于获取绘图信息。

说明

为绘图准备 CWnd 并用与绘图有关的信息填充 PAINTSTRUCT 数据结构。

绘图结构中包含了一个 RECT 数据结构，它包含了完全封闭更新区域的最小矩形以及一个标志，指明背景是否需要擦除。

更新区域是由 Invalidate、InvalidateRect 或 InvalidateRgn 成员函数设置的，并且在更新区域改变大小、移动、创建、滚动或执行其它会影响客户区的操作后由系统设置。如果更新区域被标记为需要擦除，则 BeginPaint 发送一个 WM_ONERASEBKGND 消息。

除非是在响应 WM_PAINT 消息的时候，否则不要调用 BeginPaint 成员函数。每个对 BeginPaint 成员函数的调用都必须有对应的对 EndPaint 成员函数的调用。如果在这个区域中的插字符需要被重画，那么 BeginPaint 成员函数自动隐

藏插字符以免被擦除。

请参阅 CWnd::EndPaint, CWnd::Invalidate, CWnd::InvalidateRgn, ::BeginPaint, CPaintDC

CWnd::BindDefaultProperty

void BindDefaultProperty(DISPID *dwDispID*, VARTYPE *vtProp*, LPCTSTR *szFieldName*, CWnd * *pDSCWnd*);

参数

dwDispID
    指定要与数据源控件绑定的数据绑定控件的属性的 DISPID。

vtProp
    指定要绑定的属性的类型 ---- 例如，VT_BSTR，VT_VARIANT 等等。

szFieldName
    指定要与属性绑定的字段的名字，位于数据源控件提供的游标中。

pDSCWnd
    指向拥有数据源控件的窗口，属性将与该窗口绑定。利用 DCS 的宿主窗口的资源 ID 调用 GetDlgItem 以获取这个指针。

说明

如类型库中标记的那样将调用对象缺省的简单绑定属性（比如编辑控件）与游标绑定起来，该游标是通过数据源控件的 DataSource、UserName、Password **和** SQL **属性定义的。你调用这个函数的** CWnd **对象必须是一个数据绑定对象。** BindDefaultProperty **必须在下面的上下文中使用：**

```
BOOL CMyDlg::OnInitDialog()
{
    ...
    CWnd* pDSC = GetDlgItem(IDC_REMOTEDATACONTROL);
    CWnd* pList = GetDlgItem(IDC_DBLISTBOX);
    pList->BindDefaultProperty(0x2,
        VT_BSTR, _T("CourseID"), pDSC);
    CWnd* pEdit = GetDlgItem(IDC_MASKEDBOX);
    pEdit->BindDefaultProperty(0x16,
        VT_BSTR, _T("InstuctorID"), pDSC);
    ...
    return TRUE;
}
```

请参阅　CWnd::GetDSCCursor, CWnd::BindProperty

CWnd::BindProperty

void BindProperty(DISPID *dwDispID*, CWnd * *pWndDSC*);

参数

dwDispID

指定了要与数据源控件绑定的数据绑定控件的属性的 DISPID。

pWndDSC

指向拥有数据源控件的窗口，属性将与该窗口绑定。利用 DCS 的宿主窗口的资源 ID 调用 GetDlgItem 以获取这个指针。

说明

将数据绑定控件（如网格控件）的游标绑定属性与数据源控件绑定起来，并且在 MFC 绑定管理器中注册这种联系。BindProperty 必须在下面的上下文中使用：

```
BOOL CMyDlg::OnInitDialog()
{
	...
	CWnd* pDSC = GetDlgItem(IDC_REMOTEDATACONTROL);
	CWnd* pList= GetDlgItem(IDC_DBLISTBOX);
	pList.BindProperty(0x9, pDSC);
```

```
    ...
    return TRUE;
}
```

请参阅 CWnd::GetDSCCursor, CWnd::BindDefaultProperty

CWnd::BringWindowToTop

void BringWindowToTop( );

说明

这个函数将 CWnd **放到重叠窗口堆栈的顶部。另外，**BringWindowToTop **还激活弹出窗口、顶层窗口以及** MDI **子窗口。**BringWindowToTop **成员函数应当被用来把任何被重叠窗口部分或完全遮住的窗口显露出来。**

调用这个函数类似于调用 SetWindowPos **函数来改变窗口在** Z **轴方向上的位置。**BringWindowToTop **函数并不将窗口的风格改变为桌面上的顶层窗口。**

请参阅 ::BringWindowToTop

CWnd::CalcWindowRect

virtual void CalcWindowRect(LPRECT *lpClientRect*, UINT *nAdjustType* =

adjustBorder);

参数

lpClientRect

指向一个 RECT **结构或** CRect **对象，其中包含了窗口矩形的值。**

nAdjustType

用于现场编辑的枚举类型。它可以具有以下的值：CWnd::adjustBorder = 0**，意味着在计算时不考虑滚动条大小；或者** CWnd::adjustBorder = 1**，意味着它们将被加入最终的矩形大小。**

说明

调用这个成员函数以根据所需的客户矩形大小计算窗口矩形的大小。随后算出的窗口矩形（保存在 lpClientRect**）中可以被传递到** Create **成员函数以创建一个窗口，其客户区大小就是要求的大小。**

框架在创建窗口之前确定窗口的大小。

客户矩形是完全封闭客户区的最小矩形。窗口矩形是完全封闭窗口的最小矩形。

请参阅　::AdjustWindowRectEx

CWnd::CancelToolTips

static void PASCAL CancelToolTips(BOOL *bKeys* = FALSE);

参数

bKeys

如果为 TRUE，当按下键时取消工具提示，并将状态条文本设为缺省值；否则为 FALSE。

说明

如果当前显示了工具提示，则调用这个函数以从屏幕上清除工具提示。

注意：这个成员函数对你的代码管理的工具提示不起作用。它只影响CWnd::Enable ToolTips管理的工具提示控件。

请参阅 EnableToolTips, TTM_ACTIVATE

CWnd::CenterWindow

void CenterWindow(CWnd* *pAlternateOwner* = NULL);

参数

pAlternateOwner
指向一个窗口的指针，本窗口将被定位到该窗口（而不是其它的父窗口）的中央。

说明

这个函数将一个窗口定位到它的父窗口的中央。通常在 CDialog::OnInitDialog 中调用，用于将对话框定位到应用程序主窗口的中央。在缺省情况下，这个函数将子窗口定位到它们的父窗口的中央，而将弹出窗口定位到拥有者的中央。如果弹出窗口没有拥有者，它将被定位到屏幕中央。如果要使窗口根据不是父窗口也不是拥有者的窗口来定位，则可以将 pAlternateOwner 参数可以被设为一个有效的窗口。如果要强迫相对于屏幕定位，则应在 pAlternateOwner 参数中传递 CWnd::GetDesktopWindow 返回的值。

请参阅 CWnd::GetDesktopWindow, CDialog::OnInitDialog

CWnd::ChangeClipboardChain

BOOL ChangeClipboardChain(HWND *hWndNext*);

返回值

如果成功，则返回一个非零值；否则返回 0。

参数

hWndNext

  标识了剪贴板观察器链中跟在 CWnd **后面的窗口。**

说明

从剪贴板观察器的链中移去 CWnd**，并且使** hWndNext **指定的窗口称为链中** CWnd **的后继者。**

请参阅 CWnd::SetClipboardViewer, ::ChangeClipboardChain

CWnd::CheckDlgButton

void CheckDlgButton(int nIDButton, UINT *nCheck*);

参数

nIDButton

  指定要修改的按钮。

nCheck

指定了要执行的动作。如果 *nCheck* 为非零值，则 CheckDlgButton 成员函数将一个检查标记放在按钮旁边；如果为 0，则清除检查标记。对于三态按钮，如果 *nCheck* 为 2，则按钮的状态不确定。

说明

选择（置检查标记）或清除（清除检查标记）一个按钮，或者改变三态按钮的状态。

CheckDlgButton 函数向指定的按钮发送一条 BM_SETCHECK 消息。

请参阅 CWnd::IsDlgButtonChecked, CButton::SetCheck, ::CheckDlgButton

CWnd::CheckRadioButton

void CheckRadioButton(int *nIDFirstButton*, int *nIDLastButton*, int *nIDCheckButton*);

参数

nIDFirstButton

指定组中第一个单选按钮的整数标识符。

nIDLastButton

指定组中最后一个单选按钮的整数标识符。

nIDCheckButton

指定了要选中的单选按钮的整数标识符。

说明

从一组按钮中选择（加入检查标记）一个单选按钮并清除（清除检查标记）同组中其它单选按钮。

CheckRadioButton 函数向指定的单选按钮发送一条 BM_SETCHECK 消息。

请参阅 CWnd::GetCheckedRadioButton, CButton::SetCheck, ::CheckRadioButton

CWnd::ChildWindowFromPoint

CWnd* ChildWindowFromPoint(POINT *point*) const;
CWnd* ChildWindowFromPoint(POINT *point*, UINT *nFlags*) const;

返回值

标识了包含指定点的子窗口。如果给定的点位于客户区之外，则返回 NULL。如果给定点位于客户区之内，但是不在任何子窗口之内，则返回 CWnd。

此成员函数将返回包含了指定点的被隐藏或禁止的子窗口。

可能有多于一个子窗口包含了给定的点。但是，这个函数仅返回包含该点的第一个窗口的 CWnd* 指针。

返回的 CWnd* 有可能是临时的，不能保存以供将来使用。

参数

point

指定了要被测试的点的客户区坐标。

nflags

指定了要跳过哪个子窗口。这个参数可以是如下值的组合：

值	含义
CWP_ALL	不跳过任何子窗口
CWP_SKIPINVISIBLE	跳过不可见的子窗口
CWP_SKIPDISABLED	跳过被禁止的子窗口
CWP_SKIPTRANSPARENT	跳过透明的子窗口

说明

确定 CWnd 的哪个子窗口包含了指定的点。

请参阅 CWnd::WindowFromPoint, ::ChildWindowFromPoint

CWnd::ClientToScreen

void ClientToScreen(LPPOINT *lpPoint*) const;
void ClientToScreen(LPRECT *lpRect*) const;

参数

lpPoint
　　指向一个 POINT **结构或** CPoint **对象，其中包含了要转换的客户区坐标。**
lpRect
　　指向一个 RECT **结构或** CRect **对象，其中包含了要转换的客户区坐标。**

说明

将显示器上给定点或矩形的客户区坐标转换为屏幕坐标。ClientToScreen **成员函数使用** lpPoint **或** lpRect **指向的** POINT **或** RECT **结构或者** CPoint **或** CRect **对象的客户区坐标来计算新的屏幕坐标，然后它将结构中的坐标替换为新坐标。新的屏幕坐标是相对于系统显示器的左上角的。**

ClientToScreen **成员函数假定给定的点或矩形使用的是客户区坐标。**

请参阅　CWnd::ScreenToClient, ::ClientToScreen

CWnd::ContinueModal

BOOL ContinueModal( );

返回值

如果要继续模式循环，则返回非零值；如果调用了 EndModalLoop**，则返回** 0**。**

说明

这个成员是由 RunModalLoop **调用的，用于确定何时退出模式状态。在缺省情况下，它返回一个非零值，直到调用了** EndModalLoop**。**

请参阅 RunModalLoop, EndModalLoop

CWnd::Create

virtual BOOL Create(LPCTSTR *lpszClassName*, LPCTSTR *lpszWindowName*, DWORD *dwStyle*, const RECT& *rect*, CWnd* *pParentWnd*, UINT *nID*, CCreateContext* *pContext* = NULL);

返回值

如果成功，则返回非零值；否则返回 0。

参数

lpszClassName

指向一个以 null 结尾的字符串，它命名了一个 Windows 的窗口类（一个 WNDCLASS 结构）。类名可以是用全局函数 AfxRegisterWndClass 注册的任何名字，也可以是任何预定义的控制类名。如果该参数为 NULL，则使用缺省的 CWnd 属性。

lpszWindowName

指向一个 null 结尾的字符串，其中包含了窗口名。

dwStyle

指定了窗口风格属性。不能使用 WS_POPUP。如果你想要创建一个弹出窗口，则应使用 CWnd::CreateEx。

rect

窗口的位置和大小，使用 *pParentWnd* 的客户区坐标。

pParentWnd

父窗口。

nID

子窗口的 ID。

pContext

窗口的创建上下文。

说明

创建一个 Windows 的子窗口，并将它连接到 CWnd 对象上。

你可以经过两步构造一个子窗口。首先，调用构造函数，创建一个 CWnd 对象。然后调用 Create，创建一个 Windows 的子窗口，并将它连接到 CWnd。Create 函数初始化窗口的类名、窗口名，并为它的风格、父窗口和 ID 注册值。

请参阅 CWnd::CWnd, CWnd::CreateEx

CWnd::CreateCaret

void CreateCaret(CBitmap* *pBitmap*);

参数

pBitmap

标识了一个位图，定义了插入字符的形状。

说明

为系统插入字符创建一个新的形状，并声明对插入字符的所有权。

这个位图必须是先前用 CBitmap::CreateBitmap **成员函数、**Windows **的** CreateDIBitmap **函数或** CBitmap::LoadBitmap **成员函数创建的。**

CreateCaret **自动销毁原来的插字符形状，并不考虑哪个窗口拥有这个插字符。在被创建之后，插字符最初是隐藏的。要显示插字符，必须调用** ShowCaret **成员函数。**

系统插字符是一种共享资源。CWnd **只应在它具有输入焦点或者活动的时候才创建插字符。在它失去输入焦点或变为非活动之前，它应该销毁插字符。**

请参阅 CBitmap::CreateBitmap, ::CreateDIBitmap, ::DestroyCaret, CBitmap::LoadBitmap, CWnd::ShowCaret, ::CreateCaret

CWnd::CreateControl

BOOL CWnd::CreateControl(LPCTSTR *lpszClass*, LPCTSTR *lpszWindowName*, DWORD *dwStyle*, const RECT& *rect*, CWnd* *pParentWnd*, UINT *nID*, CFile* *pPersist* = NULL, BOOL *bStorage* = FALSE, BSTR *bstrLicKey* = NULL);
BOOL CWnd::CreateControl(REFCLSID *clsid*, LPCTSTR *lpszWindowName*, DWORD *dwStyle*, const RECT& *rect*, CWnd* *pParentWnd*, UINT *nID*, CFile*

pPersist = NULL, BOOL *bStorage* = FALSE, BSTR *bstrLicKey* = NULL);

返回值

如果成功，则返回非零值；否则返回 0。

参数

lpszClass

这个字符串可能包含了该类的 OLE **的"短名"（**ProgID**），例如，**"CIRC3.Circ3Ctrl.1"**。这个名字应该与控件注册的名字相匹配。或者，这个字符串可能包含了** CLSID **的字符串形式，包括在大括号内，例如，**"{9DBAFCCF-592F-101B-85CE-00608CEC297B}"**。在其它情况下，**CreateControl **将该字符串转换为对应的类** ID。

lpszWindowName

指向要显示在控件中的文本的指针。设置了控件的标题或文本属性（如果有）的值。如果该指针为 NULL，**则不改变控件的标题或文本属性。**

dwStyle

Windows **风格。可能的取值在说明部分列出。**

rect

指定了控件的大小和位置。它可以是一个 CRect **对象，也可以是一个** RECT

结构。

pParentWnd

指定了控件的父窗口。它不能为 NULL。

nID

指定了控件的 ID。

pPersist

指向一个 CFile 对象的指针，其中包含了控件的永久状态。缺省值为 NULL，表明控件在初始化自己的时候并不读任何永久性的存储。如果该参数不是 NULL，它必须是一个 CFile 派生类对象的指针，其中包含了控件的永久数据，可以是流的形式，也可以是存储的形式。这些数据必须是在客户以前的活动中保存的。CFile 对象中还可以包含其它数据，但是当调用 CreateControl 的时候，它的读写指针必须定位在永久数据的第一个字节。

bStorage

指明 *pPersist* 中的数据是被解释为 IStorage 数据还是 IStream 数据。如果 *pPersist* 中的数据是一种存储，则 *bStorage* 应该为 TRUE。如果 *pPersist* 中的数据是一个流，则 *bStorage* 应该是 FALSE。缺省值为 FALSE。

bstrLicKey

可选的许可键数据。这个数据仅在创建需要运行时许可的控件时才需要。

如果该控件支持许可，要成功地创建控件，你必须提供许可键。缺省的值为 NULL。

clsid

控件的唯一的类 ID。

说明

使用这个成员函数来创建一个 OLE 控件，在 MFC 程序中，它用一个 CWnd 对象来代表。CreateControl 与 CWnd::Create 函数类似，CWnd::Create 为 CWnd 创建一个窗口。CreateControl 创建一个 OLE 控件，而不是其它的普通窗口。

CreateControl 仅支持 Windows 的 dwStyle 风格的一个子集：

- WS_VISIBLE　创建一个最初可见的窗口。如果你希望该控件立即可见，向普通窗口一样，则需要这个风格。
- WS_DISABLED　创建一个最初被禁止的窗口。被禁止的窗口不能接收用户的输入。如果控件具有 Enable 属性，则可以设置这个风格。
- WS_BORDER　创建一个带有细边框的窗口。如果控件具有 BorderStyle 属性，则可以设置这个风格。
- WS_GROUP　指定了一组控件中的第一个控件。用户可以在组中使用方向键来把键盘焦点从一个控件转移到另一个控件。在第一个控件之后所有用 WS_GROUP 风格定义的控件都属于同一组。下一个具有 WS_GROUP 风格的控件将结束这个组并开始一个新组。

- WS_TABSTOP 指明当用户按下 TAB 键时，控件可以接收键盘焦点。按下 TAB 键时键盘焦点转移到具有 WS_TABSTOP 风格的下一个控件。

CWnd::CreateEx

BOOL CreateEx(DWORD dwExStyle, LPCTSTR *lpszClassName*, LPCTSTR *lpszWindowName*, DWORD *dwStyle*, int *x*, int *y*, int *nWidth*, int *nHeight*, HWND *hwndParent*, HMENU *nIDorHMenu*, LPVOID *lpParam* = NULL);
BOOL CreateEx(DWORD *dwExStyle*, LPCTSTR *lpszClassName*, LPCTSTR *lpszWindowName*, DWORD *dwStyle*, const RECT& *rect*, CWnd* *pParentWnd*, UINT *nID*, LPVOID *lpParam* = NULL);

返回值

如果成功，则返回非零值；否则返回 0。

参数

dwExStyle

指定了要创建的 CWnd **的扩展风格。对窗口应用任何扩展的窗口风格。**

lpszClassName

指向一个以 null **结尾的字符串，命名了** Windows **的窗口类（一个**

WNDCLASS结构）。类名可以是用全局函数AfxRegisterWndClass注册的任何名字，或者是任何预定义的控制类名。它不能是NULL。

lpszWindowName

指向一个用null结尾的字符串，包含了窗口的名字。

dwStyle

指定了窗口风格属性。对可能取值的描述参见“窗口风格”和CWnd::Create。

x

指定了CWnd窗口的初始x轴位置。

y

指定了CWnd窗口的初始y轴位置。

nWidth

指定了CWnd窗口的宽度（设备单位）。

nHeight

指定了CWnd窗口的高度（设备单位）。

hwndParent

标识了要创建的CWnd窗口的父窗口或拥有者窗口。对顶层窗口，使用NULL。

nIDorHMenu

标识了菜单或子窗口的标识符。其含义依赖于窗口的风格。

lpParam

指向 CREATESTRUCT 结构的 lpCreateParams 字段所代表的数据。

rect

窗口的大小和位置，使用 *pParentWnd* 的客户区坐标。

pParentWnd

父窗口。

nID

子窗口的 ID。

说明

这个函数使用 dwExStyle 所指定的扩展风格创建一个重叠式、弹出式或子窗口。

CreateEx 的参数指定了 WNDCLASS、窗口标题、窗口风格以及（可选）窗口的初始位置和大小。CreateEx 还指定了窗口的父窗口（如果有）和 ID。

当 CreateEx 执行的时候，Windows 向窗口发送 WM_GETMINMAXINFO、WM_NCCREA- TE、WM_NCCALCSIZE 和 WM_CREATE 消息。

要扩展缺省的消息处理，应从 CWnd 继承一个类，在新类中加入消息映射，并

为以上的消息提供成员函数。例如，可以重载 OnCreate 为新类提供需要的初始化功能。
重载其它 OnMessage 消息处理函数，为你的派生类提供进一步的功能。

如果给定了 WS_VISIBLE 风格，Windows 将向窗口发送激活和显示窗口所需的所有消息。如果窗口风格中指定了标题条，则 lpszWindowName 参数中指定的窗口标题将显示在标题条上。

dwStyle 参数可以是窗口风格的任意组合。

请参阅 CWnd::Create, ::CreateWindowEx

CWnd::CreateGrayCaret

void CreateGrayCaret(int *nWidth*, int *nHeight*);

参数

nWidth

指定了插字符的宽度（逻辑单位）。如果这个参数为 0，则宽度被设为系统定义的窗口边框宽度。

nHeight

指定了插字符的高度（逻辑单位）。如果这个参数为 0，则高度被设为

系统定义的窗口边框高度。

说明

这个函数为系统插字符创建一个灰色的矩形，并声明对插字符的所有权。插字符的形状可以是线条，也可以是方块。

参数 nWidth 和 nHeight 指定了插字符的宽度和高度（逻辑单位）；实际的宽度和高度（以像素为单位）依赖于映射模式。

使用 SM_CXBORDER 和 SM_CYBORDER 索引调用 Windows 的 GetSystemMetrics 函数可以获得系统的窗口边框宽度或高度。使用窗口边框的宽度和高度以确保插字符在高分辨率显示中能够看得见。

CreateGrayCaret 成员函数自动销毁原来的插字符形状，如果有的话，而并不考虑哪个窗口拥有这个插字符。在被创建之后，插字符是隐藏的。要显示插字符，必须调用 ShowCaret 成员函数。
系统插字符是一种共享资源。CWnd 只应在它具有输入焦点或处于活动状态时才创建插字符。在它失去输入焦点或变为非活动以前，它应当销毁插字符。

请参阅 ::DestroyCaret, ::GetSystemMetrics, CWnd::ShowCaret, ::CreateCaret

CWnd::CreateSolidCaret

void CreateSolidCaret(int *nWidth*, int *nHeight*);

参数

nWidth

指定了插字符的宽度（逻辑单位）。如果这个参数为 0，则宽度被设为系统定义的窗口边框宽度。

nHeight

指定了插字符的高度（逻辑单位）。如果这个参数为 0，则高度被设为系统定义的窗口边框高度。

说明

这个函数为系统插字符创建一个实心矩形，并声明对插字符的所有权。插字符的形状可以是线条，也可以是方块。

参数 nWidth 和 nHeight 指定了插字符的宽度和高度（逻辑单位）；实际的宽度和高度（以像素为单位）依赖于映射模式。

使用 SM_CXBORDER 和 SM_CYBORDER 索引调用 Windows 的 GetSystemMetrics 函数可以获得系统的窗口边框宽度或高度。使用窗口边框的

宽度和高度以确保插字符在高分辨率显示中能够看得见。

CreateSolidCaret 成员函数自动销毁原来的插字符形状，如果有的话，而并不考虑哪个窗口拥有这个插字符。在被创建之后，插字符是隐藏的。要显示插字符，必须调用 ShowCaret 成员函数。

系统插字符是一种共享资源。CWnd 只应在它具有输入焦点或处于活动状态时才创建插字符。在它失去输入焦点或变为非活动以前，它应当销毁插字符。

请参阅　::DestroyCaret, ::GetSystemMetrics, CWnd::ShowCaret, ::CreateCaret

CWnd::CWnd

CWnd( );

说明

构造一个 CWnd 对象。必须在 CreateEx 或 Create 处于函数被调用以后才会创建 Windows 的窗口并与之连接。

请参阅　CWnd::CreateEx, CWnd::Create

CWnd::Default

LRESULT Default( );

返回值

依赖于发出的消息。

说明

调用缺省的窗口过程。缺省的窗口过程提供了对应用程序没有处理的任何窗口消息的缺省处理。这个处理函数确保每个消息都被处理。

请参阅 CWnd::DefWindowProc, ::DefWindowProc

CWnd::DefWindowProc

virtual LRESULT DefWindowProc(UINT *message*, WPARAM *wParam*, LPARAM *lParam*);

返回值

依赖于发送的消息。

参数

message

指定了要处理的 Windows 消息。

wParam

指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

说明

这个函数调用缺省的窗口过程，提供了对应用程序没有处理的任何窗口消息的缺省处理。这个成员函数确保每个消息都被处理。它必须用与窗口过程接收到的参数相同的参数来调用。

请参阅 CWnd::Default, ::DefWindowProc

CWnd::DeleteTempMap

static void PASCAL DeleteTempMap( );

说明

这个函数被 CWinApp 对象的空闲时间处理函数自动调用。删除 FromHandle 成员函数所产生的任何临时 CWnd 对象。

请参阅 CWnd::FromHandle

CWnd::DestroyWindow

```
virtual BOOL DestroyWindow( );
```

返回值

如果销毁了窗口，则返回非零值；否则返回 0。

说明

这个函数销毁一个与 CWnd 对象相连接的 Windows 窗口。DestroyWindow 成员函数向窗口发送一个适当的消息，以使该窗口变为非激活的并移去输入焦点。它还销毁窗口的菜单，清除应用程序的队列，销毁定时器，清除剪贴板拥有权，并且如果 CWnd 对象位于剪贴板观察器链的顶部，还打断剪贴板观察器链。它向窗口发送 WM_DESTROY 消息和 WM_NCDESTROY 消息。它不销毁 CWnd 对象。

DestroyWindow 是一个用于执行清除工作的占位符。因为 DestroyWindow 是一个虚拟函数，在 ClassWizard 中，它显示在任何 CWnd 的派生类中。但是即使你在自己的 CWnd 派生类中重载了这个函数，也不必调用 DestroyWindow。如果在 MFC 代码中没有调用 DestroyWindow，并且你希望调用它的话，必须在自己的代码中调用它。

例如，假定你在 CView 的派生类中重载了 DestroyWindow。由于 MFC 的源代码在任何 CFrameWnd 的派生类中都没有调用 DestroyWindow，因此除非你调用了它，否则你重载的 DestroyWindow 不会被调用。

如果该窗口是其它窗口的父窗口，当父窗口被销毁时，这些子窗口将被自动销毁。DestroyWindow 成员函数首先销毁子窗口，然后销毁本窗口。

DestroyWindow 成员函数也销毁 CDialog::Create 创建的无模式对话框。
如果要被销毁的 CWnd 是一个子窗口并且没有设置 WS_EX_NOPARENTNOTIFY 风格，则 WM_PARENTNOTIFY 消息将被发送到父窗口。

请参阅 CWnd::OnDestroy, CWnd::Detach, ::DestroyWindow

CWnd::Detach

HWND Detach( );

返回值

指向 Windows 对象的 HWND 句柄。

说明

将一个 Windows 的句柄从 CWnd 对象上分离并返回这个句柄。

请参阅 CWnd::Attach

CWnd::DlgDirList

int DlgDirList(LPTSTR *lpPathSpec*, int *nIDListBox*, int *nIDStaticPath*, UINT *nFileType*);

返回值

如果函数成功，则返回非零值；否则返回 0。

参数

lpPathSpec

指向一个以 null 结尾的字符串，其中包含了路径或文件名。DlgDirList 修改这个字符串，它必须足够长，能够容纳改变后的内容。更多的信息

参见后面的说明部分。

nIDListBox

指定了列表框的标识符。如果 *nIDListBox* 为 0，DlgDirList 假定不存在列表框，因此不作填充。

nIDStaticPath

指定了用于显示当前驱动器和目录的静态文本控件的标识符。如果 *nIDStaticPath* 为 0，则 DlgDirList 假定没有这样的文本控件。

nFileType

指定了要显示的文件的属性。它可以是下列值的组合：

- DDL_READWRITE　可读写的数据文件，没有其它属性。
- DDL_READONLY　只读文件
- DDL_HIDDEN　隐藏文件
- DDL_SYSTEM　系统文件
- DDL_DIRECTORY　目录
- DDL_ARCHIVE　档案
- DDL_POSTMSGS　LB_DIR 标志。如果设置了 LB_DIR 标志，Windows 将 DlgDirList 产生的消息放入应用程序的队列，否则，它们被直接发送到对话框过程。

- DDL_DRIVES 驱动器。如果设置了 DDL_DRIVES 标志，就会自动设置 DDL_EXCLUSIVE 标志。因此，要创建包括驱动器和文件的目录列表，你必须两次调用 DlgDirList：第一次使用 DDL_DRIVES 标志，第二次使用列表中其它标志。
- DDL_EXCLUSIVE 不相容的位。如果设置了这个位，则只列出指定类型的文件；否则列出普通文件和指定类型的文件。

说明

这个函数用文件或目录列表来填充一个列表框。DlgDirList 向列表框发送 LB_RESETCON- TENT 和 LB_DIR 消息。它用与 lpPathSpec 指定的路径匹配的所有文件的名字填充 nIDListBox 指定的列表框。

lpPathSpec 参数具有如下形式：

[drive:] [[\u]directory[\idirectory]...\u] [filename]

在这个例子中，drive 是驱动器字母，directory 是有效的目录名，filename 是有效的文件名，它必须至少包含一个通配符。通配符是一个问号（？），它意味着与任何字符匹配，还有星号（*），它意味着与任意个数的字符匹配。

如果你为 lpPathSpec 指定了一个 0 长度的字符串，或者你只指定了目录名却没有包含任何文件描述，则字符串将变为“*.*”。

如果 lpPathSpec 中包括了驱动器和/或目录名，则在填充列表框之前，当前驱动

器和目录将被改变到设定的驱动器和目录。由 nIDStaticPath 指定的文本控件也被更新为新的驱动器和/或目录名。

在列表框被填充以后，lpPathSpec 将被更新为去掉驱动器和/或目录部分的路径。

请参阅 CWnd::DlgDirListComboBox, ::DlgDirList

CWnd::DlgDirListComboBox

int DlgDirListComboBox(LPTSTR *lpPathSpec*, int *nIDComboBox*, int *nIDStaticPath*, UINT *nFileType*);

返回值

指明了该函数的结果。如果建立了一个列表框，即使是空的列表框，则返回一个非零值。返回值为 0 则意味着输入字符串中不包含有效的查找路径。

参数

lpPathSpec

指向一个以 null 结尾的字符串，其中包含了路径或文件名。DlgDirList 修改这个字符串，它必须足够长，能够容纳改变后的内容。更多的信息

参见后面的说明部分。

nIDComboBox

指定了对话框中组合框的标识符。如果 *nIDComboBox* 为 0，DlgDirListComboBox 假定不存在组合框，因此不作填充。

nIDStaticPath

指定了用于显示当前驱动器和目录的静态文本控件的标识符。如果 *nIDStaticPath* 为 0，则 DlgDirListComboBox 假定没有这样的文本控件。

nFileType

指定了要显示的文件的属性。它可以是下列值的组合：

- DDL_READWRITE 可读写的数据文件，没有其它属性。
- DDL_READONLY 只读文件
- DDL_HIDDEN 隐藏文件
- DDL_SYSTEM 系统文件
- DDL_DIRECTORY 目录
- DDL_ARCHIVE 档案
- DDL_POSTMSGS CB_DIR 标志。如果设置了 CB_DIR 标志，Windows 将 DlgDirListComboBox 产生的消息放入应用程序的队列，否则，它们被直接发送到对话框过程。
- DDL_DRIVES 驱动器。如果设置了 DDL_DRIVES 标志，就会自动设置 DDL_EXCLUSIVE 标志。因此，要创建包括驱动器和文件的目录

列表，你必须两次调用 DlgDirListComboBox：第一次使用 DDL_DRIVES 标志，第二次使用列表中其它标志。

- DDL_EXCLUSIVE 不相容的位。如果设置了这个位，则只列出指定类型的文件；否则列出普通文件和指定类型的文件。

说明

这个函数用文件或目录列表来填充一个组合框中的列表框。DlgDirListComboBox 向组合框发送 CB_RESETCONTENT 和 CB_DIR 消息。它用与 lpPathSpec 指定的路径匹配的所有文件的名字填充 nIDComboBox 指定的组合框。

lpPathSpec 参数具有如下形式：

[drive:] [[\u]directory[\idirectory]...\u] [filename]

在这个例子中，drive 是驱动器字母，directory 是有效的目录名，filename 是有效的文件名，它必须至少包含一个通配符。通配符是一个问号（？），它意味着与任何字符匹配，还有星号（*），它意味着与任意个数的字符匹配。

如果你为 lpPathSpec 指定了一个 0 长度的字符串，或者你只指定了目录名却没有包含任何文件描述，则字符串将变为“*.*”。

如果 lpPathSpec 中包括了驱动器和/或目录名，则在填充列表框之前，当前驱动器和目录将被改变到设定的驱动器和目录。由 nIDStaticPath 指定的文本控件也

被更新为新的驱动器和/或目录名。

在列表框被填充以后，lpPathSpec 将被更新为去掉驱动器和/或目录部分的路径。

请参阅　CWnd::DlgDirList, CWnd::DlgDirSelect, ::DlgDirListComboBox

CWnd::DlgDirSelect

BOOL DlgDirSelect(LPTSTR *lpString*, int *nIDListBox*);

返回值

如果成功，则返回非零值；否则返回 0。

参数

lpString
　　指向一个缓冲区，用于获取列表框的当前选择。

nIDListBox
　　指定了对话框中列表框的整数 ID。

说明

这个函数获取列表框的当前选择。它假定列表框已经被 DlgDirList 成员函数所填充并且选择项是一个驱动器字母，一个文件或一个目录名。

DlgDirSelect 成员函数将选择项拷贝到 lpString 所指定的缓冲区。如果没有选择项，则不改变 lpString。

DlgDirSelect 向列表框发送 LB_GETCURSEL 和 LB_GETTEXT 消息。
不允许从列表框返回多于一个文件名。列表框不能是多选列表框。

请参阅 CWnd::DlgDirList, CWnd::DlgDirListComboBox, CWnd::DlgDirSelectComboBox, ::DlgDirSelectEx

CWnd::DlgDirSelectComboBox

BOOL DlgDirSelectComboBox(LPTSTR *lpString*, int *nIDComboBox*);

返回值

如果成功，则返回非零值；否则返回 0。

参数

lpString

指向一个缓冲区，用于获取选择的路径。

nIDComboBox

指定了对话框中组合框的整数 ID。

说明

这个函数获取组合框中列表框的当前选择。它假定列表框已经被 DlgDirListComboBox **成员函数所填充并且选择项是一个驱动器字母、一个文件或一个目录名。**

DlgDirSelectComboBox **成员函数将选择项拷贝到** lpString **所指定的缓冲区。如果没有选择项，则不改变** lpString。

DlgDirSelectComboBox **向组合框发送** CB_GETCURSEL **和** CB_GETTEXT **消息。**

不允许从组合框返回多于一个文件名。

请参阅 CWnd::DlgDirListComboBox, ::DlgDirSelectComboBoxEx

CWnd::DoDataExchange

virtual void DoDataExchange(CDataExchange* *pDX*);

参数

pDX

指向 CDataExchange 对象的指针。

说明

框架调用这个函数以交换并校验对话框数据。

永远不要直接调用这个函数。它是由 UpdateData 成员函数所调用的。可调用 UpdateData 函数以初始化对话框控件或从对话框获取数据。

当你从 CDialog 继承应用程序特有的对话框类时，如果你想要利用框架的自动数据交换和校验功能，你需要重载这个成员函数。ClassWizard 将为你编写这个成员函数的重载版本，包含了对话框数据交换（DDX）和校验（DDV）全局函数调用所需的“数据映射”。

要自动生成这个函数的重载版本，首先用对话框编辑器创建一个对话框资源，然后继承一个应用程序特有的对话框类。然后调用 ClassWizard 并用它来把变量、数据和校验范围与新对话框的不同控件关联起来。ClassWizard 将写入重载

的 DoDataExchange，其中包含了数据映射。下面是 ClassWizard 生成的 DDX/DDV 代码块的例子：

```
void CPenWidthsDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPenWidthsDlg)
        DDX_Text(pDX, IDC_THIN_PEN_WIDTH, m_nThinWidth);
        DDV_MinMaxInt(pDX, m_nThinWidth, 1, 20);
        DDX_Text(pDX, IDC_THICK_PEN_WIDTH, m_nThickWidth);
        DDV_MinMaxInt(pDX, m_nThickWidth, 1, 20);
    //}}AFX_DATA_MAP
}
```

ClassWizard 将维护\\{{和\\}}分解符之内的代码。你不应该修改这些代码。

重载的 DoDataExchange 成员函数必须在源文件的宏语句之前。

有关 ClassWizard 生成的 DDX_和 DDV_宏的更多信息参见技术注释 26。

请参阅 CWnd::UpdateData

CWnd::DragAcceptFiles

void DragAcceptFiles(BOOL *bAccept* = TRUE);

参数

bAccept

指明是否接收拖放文件的标志。

说明

使用 CWnd 指针，在应用程序的 CWinApp::InitInstance 函数中，在一个窗口的内部调用这个成员函数，以指明你的主窗口和所有的子窗口是否接收从 Windows 的文件管理器拖放的文件。

只有将 bAccept 参数设为 TRUE 并调用了 DragAcceptFiles 函数的窗口才将它标识为能够处理 Windows 的 WM_DROPFILES 消息。例如，在 MDI 应用程序中，如果在调用 DragAcceptFiles 函数的时候使用了 CMDIFrameWnd 窗口指针，则只有 CMDIFrameWnd 窗口得到 WM_DROPFILES 消息。这个消息将不会被发送到所有打开的 CMDIChildWnd 窗口。要使 CMDIChildWnd 窗口能够接收这个消息，你必须用 CMDIChildWnd 指针调用 DragAcceptFiles 函数。

要终止接收拖放文件，则调用这个成员函数并令 bAccept 等于 FALSE。

请参阅　::DragAcceptFiles, WM_DROPFILES

CWnd::DrawMenuBar

void DrawMenuBar( );

说明

重画菜单条。如果在Windows创建窗口以后菜单条发生了改变，则应调用这个函数以画出改变了的菜单条。

请参阅 ::DrawMenuBar

CWnd::EnableScrollBar

BOOL EnableScrollBar(int *nSBFlags*, UINT *nArrowFlags* = ESB_ENABLE_BOTH);

返回值

如果箭头被允许或禁止，如指定的那样，则返回非零值。否则为0，指明箭头已经处于要求的状态，或者发生了错误。

参数

nSBFlags

指定了滚动条类型。可以是下列值之一：

- SB_BOTH 允许或禁止窗口的水平和垂直滚动条的箭头。
- SB_HORZ 允许或禁止窗口的水平滚动条的箭头。

- SB_VERT　**允许或禁止窗口的垂直滚动条的箭头。**

nArrowFlags

指定滚动条箭头是被允许还是禁止，以及哪个箭头将被允许或禁止。可以是下列值之一：

- ESB_ENABLE_BOTH　**允许滚动条的两个箭头（缺省值）。**
- ESB_DISABLE_LTUP　**禁止水平滚动条的左箭头或垂直滚动条的上箭头。**
- ESB_DISABLE_RTDN　**禁止水平滚动条的右箭头或垂直滚动条的下箭头。**
- ESB_DISABLE_BOTH　**禁止滚动条的两个箭头。**

说明

允许或禁止滚动条的一个或两个箭头。

请参阅　CWnd::ShowScrollBar, CScrollBar::EnableScrollBar

CWnd::EnableScrollBarCtrl

void EnableScrollBarCtrl(int *nBar*, BOOL *bEnable* = TRUE);

参数

nBar

滚动条标识符。

bEnable

指定滚动条是要被禁止还是允许。

说明

调用这个函数以允许或禁止窗口的滚动条。如果窗口拥有同类的滚动条控件，则该滚动条被使用；否则将使用窗口自己的滚动条。

请参阅 CWnd::GetScrollBarCtrl

CWnd::EnableToolTips

BOOL EnableToolTips(BOOL *bEnable*);

返回值

如果允许工具提示，则返回 TRUE；否则返回 FALSE。

参数

bEnable

指定工具提示控件是要被允许还是禁止。如果是 TRUE 则允许该控件；如果是 FALSE 则禁止该控件。

说明

调用这个成员函数以允许给定窗口的工具提示。重载 OnToolHitTest 函数，为窗口提供 TOOLINFO 结构。

注意　有些窗口，如 CToolBar，提供 OnToolHitTest 的内建实现。

有关这个结构的更多信息参见 Win32 SDK 程序员参考中的 TOOLINFO。

要为你的子控件显示工具提示，仅调用 EnableToolTips 是不够的，除非父窗口是从 CFrameWnd 继承的。这是因为 CFrameWnd 提供了 TTN_NEEDTEXT 通知的缺省处理函数。如果你的父窗口不是从 CFrameWnd 继承的，这意味着，如果它是一个对话框或格式视图，则你的子控件的工具提示将不会正确显示，除非你提供了工具提示通知 TTN_NEEDTEXT 的缺省处理函数。

用 EnableToolTips 为窗口提供的缺省工具提示没有相关的文本。为了获得用于显示工具提示的文本，在显示工具提示窗口之前，将向工具提示控件的父窗口发送一个 TTN_NEEDTEXT 通知。如果这个消息没有处理函数，为 TOOLTIPTEXT 结构的 pszText 成员分配一些值，则将不会为工具提示显示文

本。

请参阅 CWnd::CancelToolTips, CWnd::OnToolHitTest, CToolBar, TOOLINFO

CWnd::EnableWindow

BOOL EnableWindow(BOOL *bEnable* = TRUE);

返回值

指明了调用 EnableWindow 成员函数之前窗口的状态。如果窗口原来是禁止的，则返回非零值。如果窗口原来是允许的，或者发生了错误，则返回 0。

参数

bEnable

指定是把窗口允许还是禁止。如果这个参数为 TRUE，则窗口被允许。如果参数为 FALSE，则窗口将被禁止。

说明

允许或禁止鼠标和键盘输入。当禁止输入时，类似鼠标点击和击键之类的输入将被忽略。当允许输入时，窗口将处理所有输入。

如果允许状态发生变化，则在函数返回之前将发送 WM_ENABLE 消息。

如果被禁止，则所有的子窗口都被禁止。尽管没有向它们发送 WM_ENABLE 消息。

在窗口被激活之前必须允许窗口。例如，如果应用程序正显示一个无模式对话框，并且禁止了它的主窗口，则在对话框被销毁之前，主窗口必须被允许。否则，另一个窗口将获得输入焦点并被激活。如果子窗口被禁止，当 Windows 试图确定哪个窗口将得到鼠标消息时，它将被忽略。

在缺省情况下，窗口在被创建的时候是允许的。应用程序可以在 Create 或 CreateEx 成员函数中指定 WS_DISABLED 风格以创建一个最初就禁止的窗口。在窗口被创建以后，应用程序也可以利用 EnableWindow 成员函数来允许或禁止窗口。

应用程序可以用这个函数来允许或禁止对话框中的控件。被禁止的控件不能接收输入焦点，用户也不能访问它。

请参阅 ::EnableWindow, CWnd::OnEnable

CWnd::EndModalLoop

void EndModalLoop(int *nResult*);

参数

nResult

　　包含了要返回到 RunModalLoop **的调用者的值。**

说明

调用这个成员函数以结束对 RunModalLoop **的调用。**nResult **参数被传给** RunModalLoop **的返回值。**

请参阅　CWnd::RunModalLoop, CWnd::ContinueModal

CWnd::EndPaint

void EndPaint(LPPAINTSTRUCT *lpPaint*);

参数

lpPaint

　　指向一个 PAINTSTRUCT **结构，其中包含了** BeginPaint **成员函数接收到的绘图信息。**

说明

标记了给定窗口的绘图过程的结束。每个被调用的 BeginPaint 成员函数都需要有一个对对应的 EndPaint 成员函数，仅在完成绘图操作以后。

如果 BeginPaint 成员函数隐藏了插字符，则 EndPaint 在屏幕上恢复插字符。

请参阅 CWnd::BeginPaint, ::EndPaint, CPaintDC

CWnd::ExecuteDlgInit

BOOL ExecuteDlgInit(LPCTSTR *lpszResourceName*);
BOOL ExecuteDlgInit(LPVOID *lpResource*);

返回值

如果执行了一个对话框资源，则返回 TRUE；否则返回 FALSE。

参数

lpszResourceName
指向一个以 null 结尾的字符串的指针，指定了资源的名字。

lpResource

指向资源的指针。

说明

调用这个成员函数以初始化对话框资源。

ExecuteDlgInit 将使用与执行模块绑在一起的资源，或者是其它来源的资源。要实现这个目的，ExecuteDlgInit 通过调用 AfxFindResourceHandle 来找到一个资源句柄。如果你的 MFC 应用程序没有使用共享 DLL（MFCx0[U][D].DLL），那么 AfxFindResourceHandle 调用 AfxGetResourceHandle，该函数返回可执行程序中当前资源的句柄。如果你的 MFC 应用程序使用了 MFCx0[U][D].DLL，AfxFindResourceHandle 遍历整个共享的 CDynLinkLibrary 对象列表以及扩展 DLL 以查找正确的资源句柄。

请参阅 CDialog::OnInitDialog, ::WM_INITDIALOG

CWnd::FilterToolTipMessage

void FilterToolTipMessage(MSG* *pMsg*);

参数

pMsg

执行工具提示消息的指针。

说明

这个成员函数被框架调用，用以显示与工具条上的按钮相关的工具提示信息。通常它是在 PreTranslateMessage 函数内调用的。

当框架没有为你调用这个函数时，主动调用它。

请参阅 CWnd::OnToolHitTest

CWnd::FindWindow

static CWnd* PASCAL FindWindow(LPCTSTR *lpszClassName*, LPCTSTR *lpszWindowNa- me*);

返回值

标识了具有指定的类名或窗口名的窗口。如果没有找到这样的窗口，则返回 NULL。

返回的 CWnd*值可能是临时的，不能被保存以供将来使用。

参数

lpszClassName

指向一个以null结尾的字符串，指定了窗口类（一个WNDCLASS结构）的名字。如果*lpClassName*为NULL，则所有的类名都匹配。

lpszWindowName

指向一个以null结尾的字符串，指定了窗口的名字（窗口的标题）。如果*lpWindowName*为NULL，所有的窗口名都匹配。

说明

返回顶层的CWnd，其窗口类是由lpszClassName，其窗口名或标题是lpszWindowName给定的。这个函数不搜索子窗口。

请参阅 ::FindWindow

CWnd::FlashWindow

BOOL FlashWindow(BOOL *bInvert*);

返回值

如果在调用FlashWindow成员函数之前窗口是激活的，则返回非零值；否则返

回 0。

参数

bInvert

指定 CWnd 是要闪烁还是返回它的原始状态。如果 *bInvert* 为 TRUE，则 CWnd 从一种状态闪烁到另一种状态。如果 *bInvert* 为 FALSE，则窗口返回它的原始状态（可以是活动或非活动的）。

说明

使给定窗口闪烁一次。要实现连续的闪烁，则应该创建一个系统定时器，并反复调用 FlashWindow。使 CWnd 闪烁意味着改变它的标题条的外观，就像 CWnd 从非活动状态改变到活动状态，或反之（非活动状态的标题条改变到活动标题条；活动标题条改变到非活动标题条）。

通常，窗口被闪烁以通知用户它需要被注意，但是它当前不具有输入焦点。

仅当窗口得到了输入焦点并且不再需要闪烁时，bInvert 才应被设为 FALSE，在等待输入焦点的时候，应当在对这个函数的连续调用中将它设为 TRUE。

对于最小化窗口，这个函数总是返回非零值。如果窗口是最小化的，FlashWindow 只是简单地闪烁窗口的图标，对于最小化窗口 bInvert 会被忽略。

请参阅 ::FlashWindow

CWnd::FromHandle

static CWnd* PASCAL FromHandle(HWND *hWnd*);

返回值

当给定了窗口的句柄时，这个函数返回一个 CWnd **对象的指针。如果** CWnd **对象没有与这个句柄相连接，则创建一个临时的** CWnd **对象并将它与句柄连接。这个指针可能是临时的，不能被保存以供将来使用。**

参数

hWnd

一个 Windows **窗口的** HWND **句柄。**

请参阅 CWnd::DeleteTempMap

CWnd::FromHandlePermanent

static CWnd* PASCAL FromHandlePermanent(HWND *hWnd*);

返回值

指向一个CWnd对象的指针。

参数

hWnd

　　一个Windows窗口的HWND句柄。

说明

当给定了一个窗口的句柄时，这个函数返回指向一个CWnd对象的指针。如果CWnd对象没有与这个句柄连接，则返回NULL。

这个函数与FromHandle不同，它不生成临时对象。

请参阅　CWnd::FromHandle

CWnd::GetActiveWindow

static CWnd* PASCAL GetActiveWindow( );

返回值

返回活动窗口，如果在被调用时没有活动窗口，则返回 NULL。这个指针可能是临时的，不能被保存以供将来使用。

说明

这个函数获得活动窗口的指针。活动窗口或者是拥有当前输入焦点的窗口，或者是用 SetActiveWindow 成员函数激活的窗口。

请参阅 CWnd::SetActiveWindow, ::GetActiveWindow

CWnd::GetCapture

```
static CWnd* PASCAL GetCapture( );
```

返回值

标识了捕获鼠标的窗口。如果没有窗口捕获鼠标则返回 NULL。

返回值可能是临时的，不能被保存以供将来使用。

说明

这个函数获得捕获了鼠标的窗口。在一个给定的时刻只能有一个窗口捕获鼠标。当调用了 SetCapture 成员函数之后，窗口将接收鼠标捕获。这个窗口将接收所有的鼠标输入，不管光标是否位于它的边界内。

请参阅 CWnd::SetCapture, ::GetCapture

CWnd::GetCaretPos

```
static CPoint PASCAL GetCaretPos( );
```

返回值

返回一个 CPoint 对象，包含了插字符位置的坐标。

说明

这个函数获得插字符当前位置的客户区坐标，并且通过 CPoint 返回。

插字符的位置是用 CWnd 窗口的客户区坐标给出的。

请参阅 ::GetCaretPos

CWnd::GetCheckedRadioButton

int GetCheckedRadioButton(int *nIDFirstButton*, int *nIDLastButton*);

返回值

被选中的单选按钮的 ID，如果没有选中任何项，则返回 0。

参数

nIDFirstButton
指定了按钮组中第一个单选按钮的整数标识符。

nIDLastButton
指定了按钮组中最后一个单选按钮的整数标识符。

说明

获得指定的按钮组中当前被选中的单选按钮的 ID。

请参阅 CWnd::CheckRadioButton

CWnd::GetClientRect

void GetClientRect(LPRECT *lpRect*) const;

参数

lpRect

指向一个 RECT 结构或一个 CRect 对象，用于接收客户区坐标。left 和 top 成员将被设为 0。right 和 bottom 成员将包含了窗口的宽度和高度。

说明

这个函数将 CWnd 的客户区的客户坐标拷贝到 lpRect 指向的结构中。客户坐标指定了客户区的左上角和右下角。由于客户坐标是相对于 CWnd 客户区的左上角的，因此左上角的坐标是（0,0）。

请参阅 CWnd::GetWindowRect, ::GetClientRect

CWnd::GetClipboardOwner

static CWnd* PASCAL GetClipboardOwner( );

返回值

如果这个函数执行成功，则返回拥有剪贴板的窗口。否则，返回 NULL。
返回的指针可能是临时的，不能被保存以供将来使用。

说明

这个函数获得剪贴板的当前拥有者。
即使剪贴板当前没有拥有者，它也可以包含数据。
请参阅 CWnd::GetClipboardViewer, ::GetClipboardOwner

CWnd::GetClipboardViewer

```
static CWnd* PASCAL GetClipboardViewer( );
```

返回值

如果这个函数执行成功，则返回当前负责显示剪贴板的窗口；否则返回 NULL（例如，如果现在没有观察器）。
返回的指针可能是临时的，不能被保存以供将来使用。

说明

这个函数获得剪贴板观察器链中的第一个窗口。

请参阅 CWnd::GetClipboardOwner, ::GetClipboardViewer

CWnd::GetControlUnknown

LPUNKNOWN GetControlUnknown( );

返回值

返回CWnd对象所代表的OLE控件的IUnknown接口的指针。如果这个对象不代表一个OLE控件，则返回值为NULL。

说明

调用这个函数以获得未知OLE控件的指针。你不应该释放这个IUnknown指针。通常，你必须使用它来访问控件的指定接口。

GetControlUnknown返回的接口指针没有被加入引用计数。不要对指针调用IUnknown::Release，除非你原来对它调用了IUnknown::AddRef。

请参阅 IUnknown::Release, IUnknown::QueryInterface

CWnd::GetCurrentMessage

static const MSG* PASCAL GetCurrentMessage( );

返回值

返回一个指向 MSG 结构的指针，该结构中包含了窗口当前处理的消息。只应在 OnMessage 消息处理函数内部调用。

CWnd::GetDC

CDC* GetDC( );

返回值

如果调用成功，则返回 CWnd 客户区的设备环境；否则，返回 NULL。这个指针可能是临时的，不能被保存以供将来使用。

说明

这个函数获得一个指针，指向一个客户区的公用的、属于类的或者私有的设备环境，依赖于为 CWnd 指定的类风格。对于公用的设备环境，GetDC 每次获得设备环境时都给它赋予缺省值。对于属于类的或者私有的设备环境，GetDC 保

持原来的属性不变。在随后的图形设备接口（GDI）函数中可以使用设备环境以在客户区中绘图。

除非设备环境属于一个窗口类，否则在绘图之后必须调用 ReleaseDC 成员函数以释放设备环境。由于在同一时刻只有五个公用设备环境可供使用，因此如果释放设备环境时失败，可能导致其它应用程序不能访问设备环境。

如果在注册窗口类的时候，在 WNDCLASS 的风格中指定了 CS_CLASSDC，CS_OWNDC 或 CS_PARENTDC，则 GetDC 成员函数将返回属于 CWnd 类的设备环境。

请参阅 CWnd::GetDCEx, CWnd::ReleaseDC, CWnd::GetWindowDC, ::GetDC, CClientDC

CWnd::GetDCEx

CDC* GetDCEx(CRgn* *prgnClip*, DWORD *flags*);

返回值

如果函数调用成功，则返回指定窗口的设备环境；否则返回 NULL。

参数

prgnClip

标识了一个裁剪区域，可能与客户窗口的可视区域组合。

flags

可以是下列值之一：

- DCX_CACHE 由缓存而不是 OWNDC 或 CLASSDC 窗口返回一个设备环境。该标志覆盖 CS_OWNDC 和 CS_CLASSDC。
- DCX_CLIPCHILDREN 不包括此 CWnd 窗口下的所有子窗口的可见区域。
- DCX_CLIPSIBLINGS 不包括此 CWnd 窗口之上的所有兄弟窗口的可见区域。
- DCX_EXCLUDERGN 在返回的设备环境的可见区域中不包括 *prgnClip* 标识的裁剪区域。
- DCX_INTERSECTRGN 返回的设备环境的可见区域与 *prgnClip* 所标识的裁剪区域重叠。
- DCX_LOCKWINDOWUPDATE 即使调用了会影响此窗口的 LockWindowUpdate 也允许在窗口内绘图。这个值用于在跟踪时绘图。
- DCX_PARENTCLIP 使用父窗口的可见区域并忽略父窗口的 WS_CLIPCHILDR- EN 和 WS_PARENTDC 风格。这个值将设备环境的原点设为 CWnd 窗口的左上角。

- DCX_WINDOW 返回与窗口矩形而不是客户矩形相对应的设备环境。

说明

这个函数获得CWnd窗口的设备环境句柄。这个设备环境可以在随后的GDI函数中使用，用于在客户区中绘图。

这个函数是GetDC函数的扩展，使应用程序能够更多地控制一个窗口的设备环境如何或是否被裁剪。

除非设备环境属于一个窗口类，否则在绘图后必须调用ReleaseDC成员函数以释放设备环境。由于在同一时刻仅有五个公用的设备环境可供使用，因此如果释放设备环境时失败，则可能导致其它应用程序不能使用设备环境。

为了获得在缓存中的设备环境，应用程序必须指定DCX_CACHE。如果没有指定DCX_CACHE，并且窗口既不是CS_OWNDC也不是CS_CLASSDC，则这个函数返回NULL。

如果在注册窗口类的时候在WNDCLASS结构中指定了CS_CLASSDC，CS_OWNDC或CS_PARENTDC风格，则GetDCEx将返回具有指定特征的设备环境。

有关这些特征的更多信息参见“Win32 SDK”文档中WNDCLASS结构的描述。

请参阅 CWnd::BeginPaint, CWnd::GetDC, CWnd::GetWindowDC,

CWnd::ReleaseDC,::GetDCEx

CWnd::GetDescendantWindow

CWnd* GetDescendantWindow(int *nID*, BOOL *bOnlyPerm* = FALSE) const;

返回值

指向一个 CWnd 对象的指针，如果没有找到子窗口，则返回 NULL。

参数

nID
  指定了要获取的控件或子窗口的标识符。

bOnlyPerm
  指定了返回的窗口是否可以是临时的。如果为 TRUE，则只能返回永久性的窗口；如果为 FALSE，则该函数可以返回临时窗口。有关临时窗口的更多信息参见技术注释 3。

说明

调用这个函数以找到给定的 ID 所指定的后代窗口。这个成员函数搜索整个子

窗口树，并不仅是直接子窗口。

请参阅 CWnd::GetParentFrame, CWnd::IsChild, CWnd::GetDlgItem

CWnd::GetDesktopWindow

```
static CWnd* PASCAL GetDesktopWindow( );
```

返回值

标识了Windows的桌面窗口。这个指针可能是临时的，因此不能被保存以供将来使用。

说明

这个函数返回Windows的桌面窗口。桌面窗口覆盖整个屏幕，并且所有的图标和其它窗口都画在它上面。

请参阅 ::GetDesktopWindow

CWnd::GetDlgCtrlID

```
int GetDlgCtrlID( ) const;
```

返回值

如果函数调用成功，则返回 CWnd **子窗口的整数标识符；否则返回** 0。

说明

返回任何子窗口的窗口或控件 ID **值，并不仅是对话框的控件。由于顶层窗口没有** ID **值，因此如果** CWnd **是一个顶层窗口，则这个函数的返回值是没有意义的。**

请参见 ::GetDlgCtrlID

CWnd::GetDlgItem

CWnd* GetDlgItem(int *nID*) const;
void CWnd::GetDlgItem(int *nID*, HWND* *phWnd*) const;

返回值

指向给定的控件或子窗口的指针。如果没有控件具有 nID **给出的整数** ID，**则返回** NULL。

返回的指针可能是临时的，不能被保存以供将来使用。

参数

nID

指定了要获取的控件或子窗口的标识符。

phWnd

指向子窗口的指针。

说明

这个函数获得对话框或其它窗口中指定控件或子窗口的指针。返回的指针通常被强制转换为 nID 所标识的控件类型。

请参阅 CWnd::GetWindow, CWnd::GetDescendantWindow, CWnd::GetWindow, ::GetDlgItem

CWnd::GetDlgItemInt

UINT GetDlgItemInt(int *nID*, BOOL* *lpTrans* = NULL, BOOL *bSigned* = TRUE) const;

返回值

指定了对话框项的文本经转换的值。由于 0 是一个有效的返回值,必须用 lpTrans

来检测错误。如果要求返回带符号的值，则将它强制转换为整数类型。

如果经转换的值大于32767（对于带符号数）或65535（对于无符号数），则这个函数返回0。

当有错误发生时，比如遇到了非数字字符或超出了最大数范围，GetDlgItemInt将0拷贝的lpTrans指向未知。如果没有错误发生，则lpTrans接收到一个非零值。如果lpTrans为NULL，GetDlgItemInt不提出错误警告。

参数

nID

指定了要转换的对话框控件的整数标识符。

lpTrans

指向要接收转换标志的布尔。

bSigned

指定要接收的值是否带符号。

说明

这个函数接收nID标识的控件的文本。它将给定对话框中指定控件的文本转换为整数，跳过文本开始部分的空格并转换十进制数字。当它到达文本的末尾或者遇到非数字字符时就停止转换。

如果 bSigned 为 TRUE，GetDlgItemInt 在文本的开始部分检测有没有减号（－），并将文本转换为带符号数。否则，它生成一个无符号值。

它向控件发送一个 WM_GETTEXT 消息。

请参阅　CWnd::GetDlgItemText, ::GetDlgItemInt

CWnd::GetDlgItemText

int GetDlgItemText(int *nID*, LPTSTR *lpStr*, int *nMaxCount*) const;
int GetDlgItemText(int *nID*, CString& *rString*) const;

返回值

指定了被拷贝到缓冲区中的实际字节数，不包括结尾的 null 字符。如果没有拷贝文本，则返回 0。

参数

nID
　　指定了要获取其标题的控件的整数标识符。
lpStr
　　指向要接收控件的标题或文本的缓冲区。

nMaxCount

指定了要拷贝到 lpStr 的字符串的最大长度（以字节为单位）。如果字符串比 nMaxCount 要长，它将被截断。

rString

对一个 CString 对象的引用。

说明

调用这个函数以获得与对话框中的控件相关的标题或文本。GetDlgItemText 成员函数将文本拷贝到 lpStr 指向的位置并返回拷贝的字节的数目。

请参阅 CWnd::GetDlgItem, CWnd::GetDlgItemInt, ::GetDlgItemText, WM_GETTEXT

CWnd::GetDSCCursor

IUnknown * GetDSCCursor( );

返回值

指向数据源控件定义的游标的指针。MFC 会为这个指针调用 AddRef 函数。

说明

调用这个函数以获取游标的指针，该游标是用数据源控件的 DataSource，UserName，Password 和 SQL 属性定义的。可以使用返回的指针来设置复杂数据绑定控件的 ICursor 属性，如数据绑定网格控件。数据源控件将不会被激活，直到第一个绑定控件请求了游标。这种情况可能发生在调用 GetDSCCursor 时，也可能是被 MFC 绑定管理器激活的。在这两种情况下，你都可以调用 GetDSCCursor，然后用返回的 IUnknown 指针调用 Release。被激活后数据源控件会试图与数据源相连接。返回的指针应该在下面的上下文中使用：

```
BOOL CMyDlg::OnInitDialog()
{
	// 找到对话框中的子控件
	CWnd* pDSC = GetDlgItem(IDC_REMOTEDATACONTROL);
```

```
CDBListBox* pList = (CDBListBox*)
GetDlgItem(IDC_DBLISTBOX);
// 告诉 MFC 的绑定管理器我们已经把 DISPID 3
// 与数据源控件相绑定了。
pList->BindProperty(0x3, pDSC);
// 告诉列表框把哪个域作为绑定字段。
pList->SetBoundColumn(_T("CourseID"));
// 告诉列表框把它的内容转移到哪个游标和字段。
pList->SetListField(_T("CourseID"));
IPUNKNOWN *pcursor = pDSC->GetDSCCursor();
...
if (!pcursor)
{
// 指针没有成功地分配，返回 FALSE。
    return FALSE;
}
// 成功地返回了指针
pList->SetRowSource(pcursor);
...
pcursor->Release();
return TRUE;
```

}

请参阅 CWnd::BindDefaultProperty, CWnd::BindProperty

CWnd::GetExStyle

```
DWORD GetExStyle( ) const;
```

返回值

窗口的扩展风格。

请参阅 CWnd::GetStyle, ::GetWindowLong

CWnd::GetFocus

```
static CWnd* PASCAL GetFocus( );
```

返回值

指向拥有当前焦点的窗口的指针，如果没有焦点窗口，则返回 NULL。
这个指针可能是临时的，不能被保存以供将来使用。

说明

这个函数获得指向当前拥有输入焦点的 CWnd 的指针。

请参阅 CWnd::GetActiveWindow, CWnd::GetCapture, CWnd::SetFocus, ::GetFocus

CWnd::GetFont

```
CFont* GetFont( ) const;
```

返回值

指向一个 CFont 对象的指针，其中包含了当前的字体。

这个指针可能是临时的，不能被保存以供将来使用。

说明

获得窗口的当前字体。

请参阅 CWnd::SetFont, WM_GETFONT, CFont

CWnd::GetForegroundWindow

static CWnd* PASCAL GetForegroundWindow( );

返回值

指向前台窗口的指针。这可能是一个临时的 CWnd **对象。**

说明

返回指向前台窗口（用户正在其中工作的窗口）的指针。前台窗口仅适用于顶层窗口（框架窗口或对话框）。

请参阅 CWnd::SetForegroundWindow

CWnd::GetIcon

HICON GetIcon(BOOL *bBigIcon*) const;

返回值

指向一个图标的句柄。如果不成功，则返回 NULL。

参数

bBigIcon

如果为TRUE，则指定了32×32像素的图标；如果为FALSE，则指定了16×16像素的图标。

说明

调用这个函数以获得大图标（32×32）或小图标（16×16）的句柄，由bBigIcon指定。

请参阅 SetIcon

CWnd::GetLastActivePopup

```
CWnd* GetLastActivePopup( ) const;
```

返回值

标识了最近激活的弹出窗口。如果遇到了下列情况，则返回值可能是这个窗口本身：

- 该窗口是最后被激活的
- 这个窗口不拥有任何弹出窗口

- 这个窗口不是顶层窗口或被其它窗口所拥有

这个指针可能是临时的，不能被保存以供将来使用。

说明

这个函数确定 CWnd 拥有的哪个弹出窗口是最后被激活的。

请参阅 ::GetLastActivePopup

CWnd::GetMenu

```
CMenu* GetMenu( ) const;
```

返回值

标识了菜单。如果 CWnd 没有菜单，则返回值为 NULL。如果 CWnd 是子窗口，则返回值没有定义。

返回的指针可能是临时的，不能被保存以供将来使用。

说明

这个函数获得窗口的菜单指针。这个函数能对子窗口使用，因为它们没有菜单。

请参阅 ::GetMenu

CWnd::GetNextDlgGroupItem

CWnd* GetNextDlgGroupItem(CWnd* *pWndCtl*, BOOL *bPrevious* = FALSE) const;

返回值

如果这个成员函数执行成功，则返回一组中前一个（或下一个）控件的指针。返回的指针可能是临时的，不能被保存以供将来使用。

参数

pWndCtl

标识了用作搜索起点的控件。

bPrevious

指定这个函数如何在对话框的控件组中搜索。如果为 TRUE**，则该函数搜索组中的前一个控件；如果为** FALSE**，则搜索组中的下一个控件。**

说明

这个函数在对话框的控件组内搜索前一个（或下一个）控件。一组控件从用WS_GROUP 风格创建的控件开始，而以最后一个没有用 WS_GROUP 风格创建的控件结束。

在缺省情况下，GetNextDlgGroupItem 成员函数返回组中下一个控件的指针。如果 pWndCtl 指定了组中的第一个控件，并且 bPrevious 为 TRUE，则GetNextDlgGroupItem 返回组中最后一个控件的指针。

请参阅 CWnd::GetNextDlgTabItem, ::GetNextDlgGroupItem

CWnd::GetNextDlgTabItem

CWnd* GetNextDlgTabItem(CWnd* *pWndCtl*, BOOL *bPrevious* = FALSE) const;

返回值

如果这个成员函数执行成功，则返回指向具有 WS_TABSTOP 风格的前一个（或下一个）控件。

返回的指针可能是临时的，不应保存以供将来使用。

参数

pWndCtl
　　标识了用作搜索起点的控件。

bPrevious
　　指定这个函数如何在对话框中搜索。如果为 TRUE，则这个函数在对话框中搜索前一个控件；如果为 FALSE，则搜索下一个控件。

说明

这个函数获得用 WS_TABSTOP 风格创建的第一个控件，并且它位于指定的控件之前（或之后）。

请参阅 CWnd::GetNextDlgGroupItem, ::GetNextDlgTabItem

CWnd::GetNextWindow

CWnd* GetNextWindow(UINT *nFlag* = GW_HWNDNEXT) const;

返回值

如果这个成员函数执行成功，则返回窗口管理器中的下一个（或前一个）窗口。返回的指针可能是临时的，不应保存以供将来使用。

参数

nFlag

指定了该函数是返回下一个窗口还是前一个窗口的指针。它可以是GW_HWNDNEXT，表明返回窗口管理器列表中CWnd对象之后的窗口，或者是GW_HWNDPREV，返回窗口管理器列表中的前一个窗口。

说明

这个函数在窗口管理器列表中搜索下一个（或前一个）窗口。窗口管理器的列表中包含了所有顶层窗口、它们的相关子窗口和任意子窗口的子窗口。

如果CWnd是一个顶层窗口，则函数搜索下一个（或前一个）顶层窗口；如果CWnd是一个子窗口，则函数搜索下一个（或前一个）子窗口。

请参阅 ::GetNextWindow

CWnd::GetOpenClipboardWindow

static CWnd* PASCAL GetOpenClipboardWindow( );

返回值

如果函数执行成功，则返回当前打开剪贴板的窗口的句柄；否则返回NULL。

说明

这个函数获得当前打开剪贴板的窗口的句柄。

请参阅 CWnd::GetClipboardOwner, CWnd::GetClipboardViewer, CWnd::OpenClipboard,::GetOpenClipboardWindow

CWnd::GetOwner

CWnd* GetOwner( ) const;

返回值

指向 CWnd 对象的指针。

说明

这个函数获得窗口的拥有者的指针。如果窗口没有拥有者，则缺省地返回父窗口对象的指针。注意在拥有者和被拥有者之间的关系与父子关系在几个重要方面的差别。例如，具有父窗口的窗口被限制在父窗口的客户区内，但是被拥有的窗口可以被画在桌面上的任何位置。

这个函数中的拥有概念与 GetWindow 中的拥有概念不同。

请参阅 CWnd::GetParent, CWnd::SetOwner

CWnd::GetParent

CWnd* GetParent( ) const;

返回值

如果这个成员函数执行成功，则返回父窗口指针；否则返回值为 NULL，表明发生了错误或没有父窗口。

返回的指针可能是临时的，不应保存以供将来使用。

说明

调用这个函数以获得子窗口的父窗口（如果有）的指针。GetParent 函数返回直接父窗口的指针。与此不同，GetParentOwner 函数返回不是子窗口（不具有 WS_CHILD 风格）的最直接父窗口或拥有者窗口的指针。如果你在子窗口中还有子窗口，则 GetParent 和 GetParentOwner 返回不同的结果。

请参阅 CWnd::GetParentOwner, CWnd::GetOwner, CWnd::SetOwner, CWnd::SetParent,::GetParent

CWnd::GetParentFrame

CFrameWnd* GetParentFrame( ) const;

返回值

如果成功，则返回框架窗口的指针；否则返回NULL。

说明

调用这个函数以获得父框架窗口。这个成员函数向上搜索父窗口链直到找到一个CFrameWnd（或其派生类）对象。

请参阅 CWnd::GetDescendantWindow, CWnd::GetParent, CFrameWnd::GetActiveView

CWnd::GetParentOwner

CWnd* GetParentOwner( ) const;

返回值

指向一个CWnd对象的指针。如果CWnd对象没有与一个句柄连接，则创建一

个临时的 CWnd 对象并与之连接。指针可能是临时的，不应保存以供将来使用。

说明

调用这个成员函数以获得子窗口的父窗口或拥有这窗口的指针。GetParentOwner 返回不是子窗口（不具有 WS_CHILD 风格）的最直接父窗口或拥有者窗口的指针。可以用 SetOwner 来设置当前的拥有者窗口。在缺省情况下，一个窗口的父窗口就是它的拥有者。

与此不同，GetParent 函数返回直接父窗口的指针，而不管它是否是一个子窗口。如果你在子窗口内还有子窗口，则 GetParen 和 GetParentOwner 返回不同的结果。

请参阅 CWnd::GetParent, CWnd::GetOwner, CWnd::SetOwner,
CWnd::SetParent, ::GetParent

CWnd::GetProperty

void GetProperty(DISPID *dwDispID*, VARTYPE *vtProp*, void* *pvProp*)const;

参数

dwDispID

标识要获取的属性。这个值通常是由组件廊提供的。

vtProp

　　指定了要获取的属性类型。可能的取值参见 COleDispatchDriver::InvokeHelper 中的说明部分。

pvProp

　　变量的地址，该变量将接收属性值。它必须与 *vtProp* 指定的类型匹配。

说明

调用这个成员函数以获得 *dwDispID* 所指定的 OLE 控件属性。GetProperty 通过 *pvProp* 返回该值。

　　注意　这个函数只能在代表 OLE 控件的 CWnd 对象内调用。

请参阅　CWnd::InvokeHelper, COleDispatchDriver, CWnd::CreateControl

CWnd::GetSafeHwnd

```
HWND GetSafeHwnd( ) const;
```

返回值

返回窗口的句柄。如果 CWnd 对象没有与一个窗口连接或它使用的 CWnd 指针为 NULL，则返回 NULL。

CWnd::GetSafeOwner

CWnd* PASCAL GetSafeOwner(CWnd* *pParent*, HWND* *pWndTop*);

返回值

指向给定窗口的安全拥有者的指针。

参数

pParent

指向父窗口的 CWnd **对象的指针。**

pWndTop

指向当前位于顶部的窗口的指针。可能为 NULL**。**

说明

调用这个成员函数以获得拥有者窗口，可以被用于对话框或其它模式窗口。安全拥有者是 pParentWnd **的第一个不是子窗口的父窗口。如果** pParentWnd **为** NULL**，则线程的主窗口（可以通过** AfxGetMainWnd **获得）被用于找到一个拥有者。**

注意　框架本身利用这个函数以确定没有指定拥有者的对话框和属性表

的正确拥有者窗口。

请参阅 AfxGetMainWnd

CWnd::GetScrollBarCtrl

virtual CScrollBar* GetScrollBarCtrl(int *nBar*) const;

返回值

一个子滚动条控件。如果没有，则返回 NULL。

参数

nBar
指定了滚动条的类型。这个参数可以取如下值：

- SB_HORZ 获得水平滚动条的位置。
- SB_VERT 获得垂直滚动条的位置。

说明

调用这个成员函数以获取指定的子滚动控件或分隔窗口的指针。

如果滚动条是在创建窗口的时候指定了 WS_HSCROLL 或 WS_VSCROLL 位而

创建的，则这个成员函数不对滚动条进行操作。CWnd 对这个函数的实现仅简单地返回 NULL。派生类，例如 CView，实现了描述的功能。

请参阅 CWnd::EnableScrollBarCtrl

CWnd::GetScrollInfo

BOOL GetScrollInfo(int nBar, LPSCROLLINFO *lpScrollInfo*, UINT *nMask* = SIF_ALL);

返回值

如果这个消息获得了任何值，则返回值为 TRUE；否则返回值为 FALSE。

参数

nBar

指定了滚动条是一个控件还是窗口的非客户区的一部分。如果它是非客户区的一部分，则 *nBar* 还指明滚动条是水平的、垂直的，还是都有。它必须是下列值之一：

- SB_BOTH　指定了窗口的水平和垂直滚动条。
- SB_HORZ　指定了窗口的水平滚动条。

• SB_VERT 指定了窗口的垂直滚动条。

lpScrollInfo

指向一个 SCROLLINFO 结构的指针。有关这个结构的更多信息参见 Win32 SDK 程序员参考。

nMask

指定了要获取的滚动条参数。缺省值指定了 SIF_PAGE，SIF_POS，SIF_TRACKPOS 和 SIF_RANGE 的组合。有关 *nMask* 取值的更多信息参见 SCROLLINFO。

说明

调用这个函数以获得 SCROLLINFO 结构为滚动条维护的信息。GetScrollInfo 使应用程序能够使用 32 位的滚动位置值。

SCROLLINFO 结构中包含了有关滚动条的信息，包括最小和最大滚动位置，页面大小以及滚动块的位置。有关改变这个结构的缺省值的信息参见《Win32 SDK 程序员参考》中的 SCROLLINFO 结构主题。

指明滚动条位置的 MFC Windows 消息处理函数 CWnd::OnHScroll 和 CWnd::OnVScroll 只提供位置数据中的 16 位。GetScrollInfo 和 SetScrollInfo 提供了滚动条位置数据的 32 位值。因此，应用程序在处理 CWnd::OnHScroll 或 CWnd::OnVScroll 的时候可以调用 GetScrollInfo 以获得 32 位的滚动条位置数

据。

请参阅 CScrollBar::GetScrollInfo, CWnd::SetScrollInfo, CWnd::SetScrollPos, CWnd::OnVScroll, CWnd::OnHScroll, SCROLLINFO

CWnd::GetScrollLimit

int GetScrollLimit(int *nBar*);

返回值

如果函数成功，则返回指定了滚动条的最大位置的值；否则返回 0。

参数

nBar

指定了滚动条的类型。这个参数可以是下列值之一：

- SB_HORZ 获得水平滚动条的滚动限制值。
- SB_VERT 获得垂直滚动条的滚动限制值。

说明

调用这个成员函数以获得滚动条的最大滚动位置。

请参阅 CScrollBar::GetScrollLimit

CWnd::GetScrollPos

int GetScrollPos(int *nBar*) const;

返回值

如果成功，则返回滚动条中滚动块的当前位置；否则返回 0。

参数

nBar

指定了要检查的滚动条。这个参数可以是下列值之一：

- SB_HORZ **获取水平滚动条的位置。**
- SB_VERT **获取垂直滚动条的位置。**

说明

这个函数获得滚动条的滚动块的当前位置。当前位置是一个相对值，依赖于当前的滚动范围。例如，如果滚动范围是 50 **到** 100，**并且滚动块位于滚动条的中间，则当前位置为** 75。

请参阅 ::GetScrollPos, CScrollBar::GetScrollPos

CWnd::GetScrollRange

void GetScrollRange(int *nBar*, LPINT *lpMinPos*, LPINT *lpMaxPos*) const;

参数

nBar

指定了要检查的滚动条。这个参数可以是下列值之一：

- SB_HORZ **获得水平滚动条的当前位置。**
- SB_VERT **获得垂直滚动条的当前位置。**

lpMinPos

指向一个整数变量，该变量将接收最小位置。

lpMaxPos

指向一个整数变量，该变量将接收最大位置。

说明

这个函数将指定的滚动条的当前最大和最小滚动位置拷贝到 lpMinPos **和** lpMaxPos **所指向的位置。如果** CWnd **没有滚动条，则** GetScrollRange **成员函数**

将0拷贝到lpMinPos和lpMaxPos。

标准滚动条的范围是0到100。滚动条控件的缺省范围是空的（两个值都是0）。

请参阅　::GetScrollRange

CWnd::GetStyle

DWORD GetStyle( ) const;

返回值

窗口的风格。

请参阅　::GetWindowLong

CWnd::GetSystemMenu

CMenu* GetSystemMenu(BOOL *bRevert*) const;

返回值

如果*bRevert*为FALSE，则标识了控制菜单的一个拷贝；如果为TRUE，则返回值没有定义。

返回的指针可能是临时的，不能被保存以供将来使用。

参数

bRevert

指定要采取的动作。如果 *bRevert* 为 FALSE，则 GetSystemMenu 返回当前使用的控制菜单的一个拷贝的句柄。这个拷贝最初与控制菜单一样，但是可以被修改。如果 *bRevert* 为 TRUE，GetSystemMenu 将控制菜单复位到原来的状态。以前控制菜单可能发生的变化都被销毁。这时返回值没有定义。

说明

这个函数允许应用程序访问控制菜单，用于拷贝和修改。

任何不使用 GetSystemMenu 以生成它自己的控制菜单拷贝的窗口将接收标准的控制菜单。

GetSystemMenu 成员函数返回的指针可以在 CMenu::AppendMenu，CMenu::InsertMenu 或 CMenu::ModifyMenu 中使用，用于改变控制菜单。

控制菜单中最初包含了用不同的 ID 标识的项，如 SC_CLOSE，SC_MOVE 和 SC_SIZE。控制菜单中的项产生 WM_SYSCOMMAND 消息。所有的预定义控制菜单项都具有大于 0xF000 的 ID 值。如果应用程序在控制菜单中加入了项，

则必须使用小于 F000 的 ID 值。

Windows 可能自动使标准控制菜单上的项变灰。CWnd 可以通过响应 WM_INITMENU 消息来执行它自己的选中或变灰操作，这个消息是在任何菜单要被显示之前发送的。

请参阅 CMenu::AppendMenu, CMenu::InsertMenu, CMenu::ModifyMenu, ::GetSystemMenu

CWnd::GetTopLevelFrame

CFrameWnd* GetTopLevelFrame( ) const;

返回值

标识了窗口的顶层框架窗口。
返回的指针可能是临时的，不应保存以供将来使用。

说明

调用这个成员函数以获得窗口的顶层框架窗口，如果有的话。如果 CWnd 没有与之相连的窗口，或者它的顶层父窗口不是 CFrameWnd 的派生类对象，则这个函数返回 NULL。

请参阅 CWnd::GetTopLevelOwner, CWnd::GetTopLevelParent

CWnd::GetTopLevelOwner

```
CWnd* GetTopLevelOwner( ) const;
```

返回值

标识了顶层窗口。返回的指针可能是临时的，不能被保存以供将来使用。

说明

调用这个函数以获得顶层窗口。顶层窗口是桌面的子窗口。如果 CWnd **没有与之相连的窗口，则这个函数返回** NULL。

请参阅 CWnd::GetTopLevelFrame, CWnd::GetTopLevelParent

CWnd::GetTopLevelParent

```
CWnd* GetTopLevelParent( ) const;
```

返回值

标识了窗口的顶层父窗口。
返回的指针可能是临时的，不应保存以供将来使用。

说明

调用这个函数以获得窗口的顶层父窗口。GetTopLevelParent 与 GetTopLevelFrame 和 GetTopLevelOwner 类似，但是，它忽略被设为当前拥有者窗口的值。

请参阅 CWnd::GetTopLevelOwner, CWnd::GetTopLevelFrame, CWnd::GetOwner, CWnd::SetOwner

CWnd::GetTopWindow

CWnd* GetTopWindow( ) const;

返回值

标识了 CWnd 的子窗口链表中的顶层子窗口。如果没有子窗口，则返回值为 NULL。

返回值可能是临时的，不应保存以供将来使用。

说明

这个函数搜索属于 CWnd 的顶层子窗口。如果 CWnd 没有子窗口，这个函数返回 NULL。

请参阅　::GetTopWindow

CWnd::GetUpdateRect

BOOL GetUpdateRect(LPRECT *lpRect*, BOOL *bErase* = FALSE);

返回值

指定了更新区域的状态。如果更新区域不为空，则返回值为非零值；否则为 0。如果 *lpRect* 参数被设为 NULL，且存在更新区域，则返回非零值；否则为 0。

参数

lpRect

指向一个 CRect 对象或 RECT 结构，将被用于接收包含更新区域的客户坐标。

将这个参数设为 NULL 以确定在 CWnd 中是否存在更新区域。如果 lpRect 为 NULL，且存在更新区域，则 GetUpdateRect 成员函数返回非零值；如果不存在，则返回 0。这就提供了一种方法，用来确定 WM_PAINT 是否是一个无效区域引起的。在 Windows 3.0 或更早的版本中不要将这个参数设为 NULL。

bErase

指定更新区域中的背景是否要被擦除。

说明

这个函数获得完全封闭更新区域的最小矩形的坐标。如果 CWnd 是用 CS_OWNDC 创建的，并且映射模式不是 MM_TEXT，则 GetUpdateRect 成员函数用逻辑坐标给出该矩形；否则 GetUpdateRect 用客户坐标给出矩形。如果不存在更新区域，则 GetUpdateRect 将矩形设为空（所有的坐标都被设为 0）。

bErase 成员指定了 GetUpdateRect 是否要擦除更新区域的背景。如果 bErase 为 TRUE，并且更新区域不为空，则背景将被擦除。为了擦除背景，GetUpdateRect 发送一条 WM_ERASEBKGND 消息。

BeginPaint 成员函数获得的更新矩形与 GetUpdateRect 成员函数获得的矩形相同。

BeginPaint 成员函数自动使更新区域有效，因此任何在 BeginPaint 之后立即调

用的 GetUpdateRect 都返回一个空的更新区域。

请参阅 CWnd::BeginPaint, ::GetUpdateRect, CWnd::OnPaint, CWnd::RedrawWindow

CWnd::GetUpdateRgn

int GetUpdateRgn(CRgn* *pRgn*, BOOL *bErase* = FALSE);

返回值

指定了一个短整数标志，标识了结果区域的类型。该值可能为下列值之一：

- SIMPLEREGION 区域没有重叠边界。
- COMPLEXREGION 区域具有重叠边界。
- NULLREGION 区域中是空的。
- ERROR 没有创建区域。

参数

pRgn

标识了更新区域。

bErase

指定了背景是否要被擦除，以及是否要画出子窗口的非客户区。如果该值为 FALSE，则不作绘图操作。

说明

这个函数在 pRgn 标识的区域中获得更新区域。区域的坐标是相对于左上角（客户坐标）的。

BeginPaint 成员函数自动使更新区域有效，因此任何紧接在 BeginPaint 后面的 GetUpdateRgn 调用接收到空的更新区域。

请参阅　CWnd::BeginPaint, ::GetUpdateRgn

CWnd::GetWindow

CWnd* GetWindow(UINT *nCmd*) const;

返回值

返回要求的窗口指针；如果没有，则返回 NULL。

返回的指针可能是临时的，不应保存以供将来使用。

参数

nCmd

指定了CWnd和返回的窗口之间的关系。可以取下列值之一：

- GW_CHILD　标识了CWnd的第一个子窗口。
- GW_HWNDFIRST　如果CWnd是一个子窗口，则返回它的第一个兄弟窗口；否则返回列表中的第一个顶层窗口。
- GW_HWNDLAST　如果CWnd是一个子窗口，则返回最后一个兄弟窗口；否则返回列表中的最后一个顶层窗口。
- GW_HWNDNEXT　返回窗口管理器中的下一个窗口。
- GW_HWNDPREV　返回窗口管理器中的前一个窗口。
- GW_OWNER　标识了CWnd的拥有者。

请参阅　CWnd::GetParent, CWnd::GetNextWindow, ::GetWindow

CWnd::GetWindowContextHelpId

```
DWORD GetWindowContextHelpId( ) const;
```

返回值

帮助上下文标识符。如果窗口没有帮助上下文，则返回0。

说明

调用这个函数以获得与窗口相关的帮助上下文标识符，如果有的话。

CWnd::GetWindowDC

CDC* GetWindowDC( );

返回值

如果这个函数成功，则返回给定窗口的显示环境；否则返回 NULL。

返回的指针可能是临时的，不应保存以供将来使用。在每次成功地调用了 GetWindowDC 之后，必须调用 ReleaseDC。

说明

这个函数获得整个窗口的显示环境，包括标题条、菜单和滚动条。窗口的显示环境允许程序在 CWnd 的任何地方绘图，因为该环境的原点是在 CWnd 的左上角，而不是客户区的左上角。

每次获得环境的时候都给它赋以缺省的属性。以前的设置将会丢失。

GetWindowDC 用于在 CWnd 的非客户区实现特殊的绘图效果。不推荐在任何

窗口的非客户区绘图。

可以利用 Windows 的 GetSystemMetrics 函数来获得非客户区的不同部分的大小，如标题条、菜单和滚动条。

在绘图结束以后，必须调用 ReleaseDC 成员函数以释放显示环境。如果没有成功地释放显示环境，则可能会严重影响应用程序要求的绘图，因为在同一时刻能打开的显示设备环境的数目是有限的。

请参阅 ::GetSystemMetrics, CWnd::ReleaseDC, ::GetWindowDC, CWnd::GetDC, CWindowDC

CWnd::GetWindowPlacement

BOOL GetWindowPlacement(WINDOWPLACEMENT* *lpwndpl*) const;

返回值

如果函数执行成功，则返回非零值；否则返回 0。

参数

lpwndpl

指向一个 WINDOWPLACEMENT 结构，用于接收显示状态和位置信息。

说明

这个函数获得窗口的显示状态和正常（复原的）、最小化和最大化的位置。

这个函数获得的WINDOWPLACEMENT结构中的flags成员总是0。如果CWnd是最大化的，则WINDOWPLACEMENT的showCmd成员为SW_SHOWMAXIMIZED。如果窗口是最小化的，则为SW_SHOWMINIMIZED；否则它为SW_SHOWNORMAL。

请参阅 CWnd::SetWindowPlacement, ::GetWindowPlacement

CWnd::GetWindowRect

```
void GetWindowRect( LPRECT lpRect ) const;
```

参数

lpRect

指向一个CRect对象或RECT结构，用于接收左上角和右下角的屏幕坐标。

说明

这个函数将CWnd对象的边界矩形的大小拷贝到lpRect所指向的结构中。大小

是用相对于显示器屏幕左上角的屏幕坐标给出的，其中包括了标题条，边框和滚动条的大小，如果有的话。

请参阅 CWnd::GetClientRect, CWnd::MoveWindow, CWnd::SetWindowPos, ::GetWindowRect

CWnd::GetWindowRgn

int GetWindowRgn(HRGN *hRgn*)const;

返回值

返回值指定了该函数获得的区域的类型。可以是下列值之一：

- NULLREGION 区域为空。
- SIMPLEREGION 区域是一个简单的矩形。
- COMPLEXREGION 区域包括多于一个矩形。
- ERROR 发生了错误，区域没有受到影响。

参数

hRgn

窗口区域的句柄。

说明

调用这个成员函数以获得窗口的窗口区域。窗口区域确定了操作系统允许画出窗口的区域。操作系统不会在窗口区域之外显示窗口的任何部分。

窗口的窗口区域的坐标是相对于窗口的的左上角的，不是窗口的客户区域。

要设置窗口的窗口区域，调用 CWnd::SetWindowRgn。

请参阅 CWnd::SetWindowRgn

CWnd::GetWindowText

int GetWindowText(LPTSTR *lpszStringBuf*, int *nMaxCount*) const;
void GetW indowText(CString& *rString*) const;

返回值

指定了要拷贝的字符串的长度,以字节为单位,不包括结尾的空字符。如果 CWnd **没有标题或标题为空，则为** 0。

参数

lpszStringBuf

指向要接收窗口标题的复制字符串的缓冲区。

nMaxCount

指定了要拷贝的缓冲区的最大字符数目。如果字符串比 nMaxCount 指定的数目还要长，则被截断。

rString

用于接收窗口标题的复制字符串的 CString 对象。

说明

这个函数将 CWnd 的标题（如果有）拷贝到 lpszStringBuf 指向的缓冲区或者目的字符串 rString。如果 CWnd 对象是一个控件，则 GetWindowText 成员函数将拷贝控件内的文本（而不是控件的标题）。

这个成员函数会向 CWnd 对象发送一个 WM_GETTEXT 消息。

请参阅 CWnd::SetWindowText, WM_GETTEXT, CWnd::GetWindowTextLength

CWnd::GetWindowTextLength

int GetWindowTextLength( ) const;

返回值

指定了文本的长度，不包括任何结尾的空字符。如果不存在任何文本，则返回0。

说明

返回 CWnd 的标题对象的长度。如果 CWnd 是一个控件，则 GetWindowTextLength 成员函数返回控件内文本的长度（而不是标题的长度）。

这个成员函数会向 CWnd 对象发送一个 WM_GETTEXTLENGTH 消息。

请参阅 ::GetWindowTextLength, WM_GETTEXTLENGTH, CWnd::GetWindowText

CWnd::HideCaret

```
void HideCaret( );
```

说明

隐藏插字符或将它从屏幕上清除。尽管插字符不再可见，但是它还可以用 ShowCaret 成员函数再次显示。隐藏插字符不会销毁它的当前形状。

隐藏是累积的。如果 HideCaret 已经被调用了五次，在插字符能够被显示之前，必须调用 ShowCaret 成员函数五次。

请参阅 CWnd::ShowCaret, ::HideCaret

CWnd::HiliteMenuItem

BOOL HiliteMenuItem(CMenu* *pMenu*, UINT *nIDHiliteItem*, UINT *nHilite*);

返回值

指定了是否要加亮显示菜单项。如果要加亮菜单项，则返回非零值；否则返回 0。

参数

pMenu

标识了包含要加亮的菜单项的顶层菜单。

nIDHiliteItem

指定了要加亮的菜单项，依赖于 nHilite 参数的值。

nHilite

指定了是要加亮指定的菜单项，还是要清除其加亮显示状态。它可以是

MF_HILITE 或 MF_UNHILITE 与 MF_BYCOMMAND 或 MF_BYPOSITION 的组合。这些值可以用位或操作符 OR 组合起来。这些值具有如下含义：

- MF_BYCOMMAND 将 *nIDHiliteItem* 解释为菜单项 ID（缺省解释）。
- MF_BYPOSITION 将 *nIDHiliteItem* 解释为零基偏移的菜单项。
- MF_HILITE 加亮显示该项。如果没有给定这个值，则清除菜单项的加亮状态。
- MF_UNHILITE 清除菜单项的加亮显示状态。

说明

加亮显示一个顶层（菜单条）菜单项或清除其加亮显示状态。MF_HILITE 和 MF_UNHILITE 标志仅能被用于这个成员函数；它们不能与 CMenu::Modify- Menu 成员函数一起使用。

请参阅 CMenu::ModifyMenu, ::HiliteMenuItem

CWnd::Invalidate

void Invalidate(BOOL *bErase* = TRUE);

参数

bErase

指定是否要擦除更新区域内的背景。

说明

使 CWnd **的整个客户区无效。当产生下一个** WM_PAINT **消息时，客户区被标记为需要重画。也可以在产生** WM_PAINT **消息之前用** ValidateRect **或** ValidateRgn **成员函数使区域有效。**

bErase **参数指定了在处理更新区域的时候是否要擦除更新区域内的背景。如果** bErase **为** TRUE**，则当调用** BeginPaint **的时候，将擦除背景。如果** bErase **为** FALSE**，则背景保持不变。如果对于更新区域的任何部分** bErase **为** TRUE**，则整个区域的背景都会被擦除，而不仅是给定的部分。**

每当 CWnd **的更新区域不为空，并且在应用程序的窗口消息队列中没有其它消息时，**Windows **就发送一条** WM_PAINT **消息。**

请参阅 CWnd::BeginPaint, CWnd::ValidateRect, CWnd::ValidateRgn, ::InvalidateRect

CWnd::InvalidateRect

void InvalidateRect(LPCRECT *lpRect*, BOOL *bErase* = TRUE);

参数

lpRect

指向一个 CRect 对象或 RECT 结构，其中包含了要被加入更新区域的矩形（客户区坐标）。如果 *lpRect* 为 NULL，则整个客户区都被加入更新区域。

bErase

指定更新区域内的背景是否要被擦除。

说明

这个函数将给定的客户区矩形加入 CWnd 更新区域，使该区域无效。无效的矩形与更新区域内的其它区域一起被标记为在发送下一条 WM_PAINT 消息时需要重画。无效的区域在更新区域内累积，直到发生下一次 WM_PAINT 调用，这个区域被处理为止，或者直到这个区域被 ValidateRect 或 ValidateRgn 成员函数标为有效为止。

bErase 参数指定了在处理更新区域的时候是否要擦除更新区域内的背景。如果 *bErase* 为 TRUE，则当调用 BeginPaint 函数的时候，将擦除背景。如果 *bErase*

为 FALSE，则背景保持不变。如果对于更新区域的任何部分 *bErase* 为 TRUE，则整个区域的背景都会被擦除，而不仅是给定的部分。

当 CWnd 的更新区域不为空，并且应用程序的窗口消息队列中没有其它消息时，Windows 就发送一条 WM_PAINT 消息。

请参阅 CWnd::BeginPaint, CWnd::ValidateRect, CWnd::ValidateRgn, ::InvalidateRect

CWnd::InvalidateRgn

void InvalidateRgn(CRgn* *pRgn*, BOOL *bErase* = TRUE);

参数

pRgn

指向 CRgn 对象的指针，标识了要被加入更新区域的区域。这个区域被假定使用客户区坐标。如果这个参数为 NULL，则整个客户区都被加入到更新区域。

bErase

指定客户区内的背景是否要被擦除。

说明

这个函数将给定的区域加入 CWnd 更新区域，使该区域无效。无效的区域与更新区域内的其它区域一起被标记，以便在发送下一条 WM_PAINT 消息时处理重画。无效的区域在更新区域内累积，直到发生下一次 WM_PAINT 调用，这个区域被处理为止，或者直到这个区域被 ValidateRect 或 ValidateRgn 成员函数标为有效为止。

bErase 参数指定了在处理更新区域的时候是否要擦除更新区域内的背景。如果 *bErase* 为 TRUE，则当调用 BeginPaint 的时候，将擦除背景。如果 *bErase* 为 FALSE，则背景保持不变。如果对于更新区域的任何部分 *bErase* 为 TRUE，则整个区域的背景都会被擦除，而不仅是给定的部分。

当 CWnd 的更新区域不为空，并且应用程序的窗口消息队列中没有其它消息时，Windows 就发送一条 WM_PAINT 消息。

给定的区域必须是先前用一个区域函数创建的。

请参阅 CWnd::BeginPaint, CWnd::ValidateRect, CWnd::ValidateRgn, ::InvalidateRgn

CWnd::InvokeHelper

void InvokeHelper(DISPID *dwDispID*, WORD *wFlags*, VARTYPE *vtRet*, void*

pvRet, const BYTE* *pbParamInfo*, ...);
throw(COleException);
throw(COleDispatchException);

参数

dwDispID

标识了要激活的方法或属性。这个值通常是由组件廊提供的。

wFlags

描述了调用 IDispatch::Invoke **的上下文的标志。**wFlags **的可能取值参见《**Win32 SDK OLE **程序员参考》中的** IDispatch::Invoke**。**

vtRet

指定了返回值的类型。可能的取值参见 COleDispatchDriver::InvokeHelper **的说明部分。**

pvRet

将接收属性值或返回值的变量地址。它必须与 *vtRet* **指定的类型匹配。**

pbParamInfo

指向一个以 null **结尾的字节字符串，指定了** *pbParamInfo* **后面的参数的类型。可能的取值参见** COleDispatchDriver::InvokeHelper **的说明部分。**

...

参数的变量列表，属于 *pbParamInfo* 所指定的类型。

说明

调用这个函数以激活 dwDispID 所指定的 OLE 控件的方法或属性，使用 wFlags 指定的上下文。pbParamInfo 参数指定了传递给方法或属性的参数的类型。在句法定义中参数的变量列表用...来代表。

这个函数将参数转换为 VARIANTARG 值，然后对 OLE 控件调用 IDispatch::Invoke 方法。如果对 IDispatch::Invoke 的调用失败，这个函数将抛出一个异常。如果 IDispatch::Invoke 返回的 SCODE（状态码）是 DISP_E_EXCEPTION，则这个函数抛出一个 COleException 对象；否则它抛出一个 COleDispatchException 对象。

注意　这个函数只能在代表 OLE 控件的 CWnd 对象内调用。

有关在 OLE 控件容器中使用这个函数的更多信息参见联机的《Visual C++程序员指南》的文章“ActiveX 控件容器：在 ActiveX 控件容器中编写 ActiveX 控件”。

请参阅 CWnd::GetProperty, CWnd::SetProperty, COleDispatchDriver, CWnd::CreateControl

CWnd::IsChild

BOOL IsChild(const CWnd* *pWnd*) const;

返回值

描述函数的结果。若 pWnd **标识的窗口是** CWnd **的子窗口，则返回非零值；否则返回** 0。

参数

pWnd

标识了要测试的窗口。

说明

这个函数指明 pWnd **标识的窗口是否是** CWnd **的子窗口或直接后代窗口。如果** CWnd **原始弹出窗口到该子窗口的父窗口链中，则该子窗口是** CWnd **的直接后代窗口。**

请参阅 ::IsChild

CWnd::IsDialogMessage

BOOL IsDialogMessage(LPMSG *lpMsg*);

返回值

指明这个函数是否已处理了给定的消息。如果消息已被处理，则返回非零值；否则返回 0。如果返回值为 0，则调用基类的 CWnd::PreTranslateMessage **成员函数以处理这个消息。在** CWnd::PreTranslateMessage **成员函数的重载版本中的代码如下：**

```
BOOL CMyDlg::PreTranslateMessage( msg )
{
	if( IsDialogMessage( msg ) )
		return TRUE;
	else
		return CWnd::PreTranslateMessage( msg );
}
```

参数

lpMsg

指向一个 MSG **结构，其中包含了要被检查的消息。**

说明

调用这个函数以确定给定的消息是否是一个无模式对话框的。如果是，则函数处理这个消息。当 IsDialogMessage 函数处理消息的时候，它检查键盘消息并将它转换为对应对话框的选择命令。例如，TAB 键选择下一个控件或控件组，下箭头键选择组中的下一个控件。
你不能将一个已被 IsDialogMessage 处理的消息发送给 Windows 函数::TranslateMessage 或::DispatchMessage，因为它已经被处理了。
请参阅 ::DispatchMessage, ::TranslateMessage, ::GetMessage, CWnd::PreTranslateMessage,::IsDialogMessage

CWnd::IsDlgButtonChecked

UINT IsDlgButtonChecked(int *nIDButton*) const;

返回值

如果给定的控件被选中，则返回非零值；如果没有选中，则返回 0。只有单选按钮和复选按钮才能被选中。对于三态按钮，如果按钮状态不确定，返回值可以是 2。对于按钮，这个函数返回 0。

参数

nIDButton
指定了按钮控件的整数标识符。

说明

这个函数确定一个按钮控件的附加是否具有选中标记。如果按钮是三态控件，则这个函数确定它是变灰、被选中，或都不是。

请参阅 ::IsDlgButtonChecked, CButton::GetCheck

CWnd::IsIconic

BOOL IsIconic( ) const;

返回值

如果CWnd被最小化，则返回非零值；否则返回0。

说明

这个函数指明CWnd是否被最小化（图标化）。

请参阅 ::IsIconic

CWnd::IsWindowEnabled

BOOL IsWindowEnabled( ) const;

返回值

如果允许 CWed，则返回非零值；否则返回 0。

说明

指定了是否允许 CWnd 接收鼠标和键盘输入。

请参阅 ::IsWindowEnabled

CWnd::IsWindowVisible

BOOL IsWindowVisible( ) const;

返回值

如果 CWnd 可见（被设值了 WS_VISIBLE 风格位，其父窗口可见），则返回

非零值。因为返回值反映了WS_VISIBLE风格位的状态，因此即使CWnd被其它窗口完全遮住，返回值也可以是非零值。

说明

这个函数确定给定窗口的视觉状态。

窗口具有在WS_VISIBLE风格位中指定的可视状态。当用ShowWindow成员函数设置了这个风格位时，将显示窗口，后来在窗口中的绘图也一直显示。

当窗口被其它窗口覆盖或被父窗口裁剪时，在具有WS_VISIBLE风格位的窗口内的绘图将不会显示。

请参阅　CWnd::ShowWindow, ::IsWindowVisible

CWnd::IsZoomed

BOOL IsZoomed( ) const;

返回值

如果CWnd被最大化，则返回非零值；否则返回0。

说明

这个函数确定CWnd是否已被最大化。

请参阅 ::IsZoomed

CWnd::KillTimer

```
BOOL KillTimer( int nIDEvent );
```

返回值

指定了函数的结果。如果事件已经被销毁，则返回值为非零值。如果KillTimer成员函数不能找到指定的定时器事件，则返回0。

参数

nIDEvent
传递给SetTimer的定时器事件值。

说明

销毁以前调用SetTimer创建的用*nIDEvent*标识的定时器事件。任何与此定时器有关的未处理的WM_TIMER消息都从消息队列中清除。

请参阅 CWnd::SetTimer, ::KillTimer

CWnd::LockWindowUpdate

```
BOOL LockWindowUpdate( );
```

返回值

如果函数执行成功，则返回非零值。如果失败或者已经使用 LockWindowUpdate 函数锁定了其它窗口，则返回 0。

说明

禁止在指定的窗口内绘图。被锁定的窗口不能被移动。在同一时刻只能有一个窗口被锁定。要解锁一个用 LockWindowUpdate 锁定的窗口，调用 UnlockWindowUpdate。

如果拥有被锁定窗口（或者任何被锁定的子窗口）的应用程序调用了 Windows 的 GetDC，GetDCEx 或 BeginPaint 函数，则被调用的函数返回一个设备环境，其可视区域为空。直到应用程序调用 UnlockWindowUpdate 成员函数解锁了窗口，一直都会这样。

当窗口更新被锁定时，系统跟踪对与锁定窗口相关的设备环境所作的绘图操作

的边界矩形。当重又允许绘图时，被锁定的窗口和它的子窗口中的这个区域将被设为无效，强制发送一条 WM_PAINT 消息以更新屏幕。如果当窗口更新被锁定时没有发生绘图，则没有任何区域被设为无效。

LockWindowUpdate 成员函数不使给定的窗口不可见，也不清除 WS_VISIBLE 风格位。

请参阅 CWnd::GetDCEx, ::LockWindowUpdate

CWnd::MapWindowPoints

void MapWindowPoints(CWnd* *pwndTo*, LPRECT *lpRect*) const;
void MapWindowPoints(CWnd* *pwndTo*, LPPOINT *lpPoint*, UINT *nCount*) const;

参数

pwndTo
标识了指定点要被转换到的窗口。如果这个参数为 NULL，则该点被转换为屏幕坐标。

lpRect
指定了要转换的点所在的矩形。这个函数的第一个版本仅在 Windows 3.1 和以后的版本中有效。

lpPoint

执行 POINT 结构数组的指针，其中包含要转换的点。

nCount

指定了 *lpPoint* 指向的数组中 POINT 结构的数目。

说明

这个函数将一系列点从 CWnd 的坐标空间转换（映射）到其它窗口的坐标空间。

请参阅 CWnd::ClientToScreen, CWnd::ScreenToClient, ::MapWindowPoints

CWnd::MessageBox

int MessageBox(LPCTSTR *lpszText*, LPCTSTR *lpszCaption* = NULL, UINT *nType* = MB_OK);

返回值

指定了函数的结果。如果没有足够的内存以创建消息框，则返回 0。

参数

lpszText

指向一个 CString 对象或以 null 结尾的字符串，其中包含了要显示的信息。

lpszCaption

指向一个 CString 对象或以 null 结尾的字符串，被用作消息框标题。如果 *lpszCaption* 为 NULL，则将使用缺省的标题“Error”。

nType

指定了消息框的内容和行为。

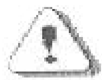
说明

这个函数创建并显示一个窗口，其中包含了应用程序提供的消息和标题，加上预定义的图标和按钮的组合，这些定义在消息框风格列表中。使用全局函数 AfxMessageBox 代替这个成员函数来为你的应用程序实现消息框。

下面显示了可以被用在消息框中的不同的系统图标：

MB_ICONHAND，MB_ICONSTOP 和 MB_ICONERROR

MB_ICONQUESTION



MB_ICONEXCLAMATION 和 MB_ICONWARNING

MB_ICONASTERISK 和 MB_ICONINFORMATION

请参阅 ::MessageBox, AfxMessageBox

CWnd::ModifyStyle

BOOL ModifyStyle(DWORD *dwRemove*, DWORD *dwAdd*, UINT *nFlags* = 0);

返回值

如果成功地修改了风格，则返回非零值；否则返回 0。

参数

dwRemove

指定了在修改风格时要清除的窗口风格。

dwAdd

指定了在修改风格时要加入的窗口风格。

nFlags

要传递给 SetWindowPos 的标志，如果不应调用 SetWinowPos，则为 0。缺省值为 0。预设的标志列表参见说明部分。

说明

调用这个成员函数以修改窗口的风格。要加入或清除的风格可以用位或操作符（|）来组合。有关可用窗口风格的信息参见《Win32 SDK 程序员参考》中的“通用窗口风格”主题和::CreateWindow。

如果 nFlags 为非零值，则 ModifyStyle 调用 Windows 的 API 函数::SetWindowPos，并将 nFLags 与下面的四个预定义值组合，以重画窗口：

- SWP_NOSIZE 保持当前大小。
- SWP_NOMOVE 保持当前位置。
- SWP_NOZORDER 保持当前的 Z 轴顺序。
- SWP_NOACTIVATE 不激活窗口。

要修改窗口的扩展风格，参见 ModifyStyleEx。

请参阅 SetWindowPos, CWnd::ModifyStyleEx, General Window Styles, ::SetWindowPos

CWnd::ModifyStyleEx

BOOL ModifyStyleEx(DWORD *dwRemove*, DWORD *dwAdd*, UINT *nFlags* = 0);

返回值

如果成功地修改了风格，则返回非零值；否则返回 0。

参数

dwRemove
　　指定了在修改风格时要清除的窗口风格。

dwAdd
　　指定了在修改风格时要加入的窗口风格。

nFlags
　　要传递给 SetWindowPos 的标志，如果不应调用 SetWinowPos，则为 0。缺省值为 0。预设的标志列表参见说明部分。

说明

调用这个成员函数以修改窗口的扩展风格。要加入或清除的风格可以用位或操作符（|）来组合。有关可用的扩展窗口风格的信息参见《Win32 SDK 程序员

参考》中的“扩展窗口风格”主题和::CreateWindow。

如果 nFlags 为非零值，则 ModifyStyleEx 调用 Windows 的 API 函数::SetW indowPos，并将 nFLags 与下面的四个预定义值组合，以重画窗口：

- SWP_NOSIZE 保持当前大小。
- SWP_NOMOVE 保持当前位置。
- SWP_NOZORDER 保持当前的 Z 轴顺序。
- SWP_NOACTIVATE 不激活窗口。

要修改窗口的常规风格，参见 ModifyStyle。

请参阅 CWnd::ModifyStyle, CreateWindowEx

CWnd::MoveWindow

void MoveWindow(int x, int y, int *nWidth*, int *nHeight*, BOOL *bRepaint* = TRUE);
void MoveWindow(LPCRECT *lpRect*, BOOL *bRepaint* = TRUE);

参数

x

指定了 CWnd 的左边的新位置。

y

指定了 CWnd 的顶部的新位置。

nWidth

指定了 CWnd 的新宽度。

nHeight

指定了 CWnd 的新高度。

bRepaint

指定了是否要重画 CWnd。如果为 TRUE，则 CWnd 象通常那样在 OnPaint 消息处理函数中接收到一条 WM_PAINT 消息。如果这个参数为 FALSE，则不会发生任何类型的重画操作。这应用于客户区、非客户区（包括标题条和滚动条）和由于 CWnd 移动而露出的父窗口的任何部分。当这个参数为 FALSE 的时候，应用程序必须明确地使 CWnd 和父窗口中必须重画的部分无效或重画。

lpRect

CRect 对象或 RECT 结构，指定了新的大小和位置。

说明

这个函数改变窗口的位置和大小。

对于顶层的 CWnd 对象，*x* 和 *y* 参数是相对于屏幕的左上角的。对于子对象，它们是相对于父窗口客户区的左上角的。

MoveWindow 函数发送一条 WM_GETMINMAXINFO 消息。处理这个消息时，CWnd 得到一个改变最大和最小的窗口缺省值的机会。如果传递给 MoveWindow 成员函数的参数超过了这些值，则在 WM_GETMINMAXINFO 处理函数中可以用最小或最大值来代替这些值。

请参阅 CWnd::SetWindowPos, WM_GETMINMAXINFO, ::MoveWindow

CWnd::OnActivate

afx_msg void OnActivate(UINT *nState*, CWnd* *pWndOther*, BOOL *bMinimized*);

参数

nState

指定 CWnd 是要被激活还是取消活动状态。它可以是下列值之一：

- WA_INACTIVE　窗口将被取消活动状态。
- WA_ACTIVE　窗口将通过不同于鼠标点击的某些方法激活（例如，用键盘接口选择窗口）。
- WA_CLICKACTIVE　窗口经鼠标点击而激活。

pWndOther

指向要激活或取消活动状态的 CWnd 对象的指针。这个指针可以为 NULL，也有可能是临时的。

bMinimized

指定了要激活或取消活动状态的 CWnd 的最小化状态。如果值为 TRUE，表明窗口是最小化的。

如果该值为 TRUE，则 CWnd 将被激活，否则将取消活动状态。

说明

当 CWnd 对象被激活或取消活动状态时，框架调用这个成员函数。首先调用要取消活动状态的主窗口的 OnActivate 函数，然后调用要被激活的主窗口的 OnActivate 函数。

如果 CWnd 对象是被鼠标点击激活的，则它还将接收到对 OnMouseActivate 的调用。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　WM_MOUSEACTIVATE, WM_NCACTIVATE, WM_ACTIVATE

CWnd::OnActivateApp

afx_msg void OnActivateApp(BOOL *bActive*, HTASK *hTask*);

参数

bActive

指定了CWnd是要被激活还是被取消活动状态。TRUE意味着CWnd要被激活。FALSE意味着CWnd将失去活动状态。

hTask

指定了任务句柄。如果bActive为TRUE，则该句柄标识了拥有被取消活动状态的CWnd对象的任务。如果bActive为FALSE，则该句柄标识了拥有被激活的CWnd对象的任务。

说明

框架为被激活的任务的所有顶层窗口或被取消活动状态的任务的所有顶层窗口调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_ACTIVATEAPP

CWnd::OnAmbientProperty

BOOL OnAmbientProperty(COleControlSite* *pSite*, DISPID *dispid*, VARIANT* *pvar*)

返回值

如果支持 ambient 属性，则返回 TRUE；否则返回 FALSE。

参数

pSite
: 指向请求 ambient 属性的空间的位置的指针。

dispid
: 被请求的 ambient 属性的调度 ID。

pvar
: 指向调用者分配的 VARIANT 结构的指针，通过这个结构来返回 ambient 属性的值。

属性

框架调用这个成员函数以从包含 OLE 控件的窗口获得 ambient 属性值。重载这

个函数以改变 OLE 控件容器向它的控件返回的缺省 ambient 属性值。任何没有被重载函数处理的 ambient 属性请求将被传递到基类的实现中。

CWnd::OnAskCbFormatName

afx_msg void OnAskCbFormatName(UINT *nMaxCount*, LPTSTR *lpszString*);

参数

nMaxCount
指定了要拷贝的最大字节数目。

lpszString
指向缓冲区的指针，格式名的拷贝将保存在此缓冲区中。

说明

当剪贴板中包含了 CF_OWNERDISPLAY 格式（这意味着剪贴板的拥有者将显示剪贴板的内容）的数据句柄时，框架调用这个成员函数。剪贴板的拥有者将为它的格式提供名字。

重载这个函数并将 CF_OWNERDISPLAY 格式的名字拷贝到指定的缓冲区中，不超过指定的最大字节数目。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数(而不是你提供给这个函数的参数)。

请参阅 WM_ASKCBFORMATNAME

CWnd::OnCancelMode

afx_msg void OnCancelMode( );

说明

框架调用这个成员函数以通知 CWnd 取消内部模式。如果 CWnd 拥有焦点，则当对话框或消息框被显示时,它的 OnCancelMode 成员函数将被调用。这使 CWnd 拥有一个取消模式的机会，如鼠标捕获等。

缺省的实现通过调用 Windows 的 ReleaseCapture 来作响应。在你的派生类中重载这个函数以处理其它模式。

请参阅 CWnd::Default, ::ReleaseCapture, WM_CANCELMODE

CWnd::OnCaptureChanged

afx_msg void OnCaptureChanged(CWnd* *pWnd*);

参数

pWnd

　　指向获得鼠标捕获的窗口的指针。

说明

框架调用这个函数以通知窗口，它已失去鼠标捕获状态。

即使窗口自己调用了::ReleaseCapture**，它也会接收到这个消息。应用程序不应试图在响应这个消息的时候设置鼠标捕获。当它接收到这个消息的时候，如果有必要，窗口必须重画自身以响应新的鼠标捕获状态。**

有关 Windows **的** ReleaseCapture **函数的信息参见《**Win32 SDK **程序员参考》。**

　　注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　WM_CAPTURECHANGED

CWnd::OnChangeCbChain

afx_msg void OnChangeCbChain(HWND *hWndRemove*, HWND *hWndAfter*);

参数

hWndRemove

指定了要从剪贴板观察器链中清除的窗口句柄。

hWndAfter

指定了剪贴板观察器链中要被清除的窗口后面的窗口的句柄。

说明

框架为剪贴板观察器链中的每个窗口调用这个成员函数，以通知它有一个窗口将从链中清除。
每个接收到 OnChangeCbChain **的** CWnd **对象应该使用** Windows **的** SendMessage **函数以向剪贴板观察器链（由** SetClipboardViewer **返回的句柄）中的下一个窗口发送** WM_CHANGECBCHAIN **消息。如果链中的下一个窗口是** hWndRemove**，则** hWndAfter **指定的窗口变为下一个窗口，剪贴板消息将发送给它。**

> **注意　框架调用这个成员函数以允许你的应用程序处理一个** Windows **消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。**

如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::ChangeClipboardChain, ::SendMessage

CWnd::OnChar

afx_msg void OnChar(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar

包含了键的字符代码值。

nRepCnt

包含了重复计数，当用户按下键时重复的击键数目。

nFlags

包含了扫描码，键暂态码，以前的键状态以及上下文代码，如下面的列表所示：

值	含义
0—15	指定了重复计数。其值是用户按下键时重复的击键数目
16—23	指定了扫描码。其值依赖于原始设备制造商（OEM）

续表

24	指明该键是否是扩展键，如增强的101或102键盘上右边的ALT或CTRL键 如果它是个扩展键，则该值为1；否则，值为0
25—28	Windows内部使用
29	指定了上下文代码。如果按键时ALT键是按下的，则该值为1；否则，值为0
30	指定了以前的键状态。如果在发送消息前键是按下的，则值为1；如果键是弹起的，则值为0
31	指定了键的暂态。如果该键正被放开，则值为1，如果键正被按下，则该值为0

说明

当击键被转换为非系统字符时，框架调用这个成员函数。这个函数是在OnKeyUp成员函数之前，OnKeyDown成员之后调用的。OnChar包含了被按下或放开的键值。

由于按键和产生的OnChar调用不必是一一对应的，因此nFlags中的信息对应用程序通常是没有用的。NFlags中的信息仅对最近在OnChar之前调用的OnKeyUp成员函数或OnKeyDown成员函数有用。

对于IBM增强型101和102键键盘，键盘的主体部分的增强键是右边的ALT和CTRL键；还有数字键盘左侧的INS，DEL，HOME，END，PAGE UP，PAGE

DOWN 以及箭头键等；以及数字键盘上的斜杠（/）和 ENTER 键。其它键盘可能会支持 nFlags 中的扩展键位。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_CHAR, WM_KEYDOWN, WM_KEYUP

CWnd::OnCharToItem

afx_msg int OnCharToItem(UINT *nChar*, ClistBox* *pListBox*, UINT *nIndex*);

返回值

框架调用这个函数以指定应用程序响应调用时的动作。返回值为-2 则表明应用程序处理选择项的所有方面，并且不需要列表框采取进一步动作。返回值为-1 则表明列表框应完成缺省动作以响应击键。返回值为 0 或更大的值则指出列表框中一个项的从 0 开始的索引值，并指出列表框应为在给定项上的击键完成缺省动作。

参数

nChar

指定了用户按下的键的值。

PListBox

指定了列表框指针，可能是临时的。

NIndex

指定了当前的插字符位置。

说明

当具有LBS_ WANTKEYBOARDINPUT风格的列表框响应WM_CHAR消息时向它的拥有者发送一个WM_CHARTOITEM消息，则调用这个函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_CHAR, WM_CHARTOITEM

CWnd::OnChildActivate

afx_msg void OnChildActivate( );

说明

如果 CWnd **对象是一个多文档界面（**MDI**）的子窗口，则当用户点击窗口的标题条或窗口被激活、移动或改变大小时，框架调用这个函数。**

请参阅 CWnd::SetWindowPos, WM_CHILDACTIVATE

CWnd::OnChildNotify

virtual BOOL OnChildNotify(UINT *message*, WPARAM *wParam*, LPARAM *lParam*, LRESULT* *pLResult*);

返回值

如果窗口负责处理发送给它的父窗口的消息，则返回非零值；否则返回 0**。**

参数

message

发送给父窗口的 Windows 消息。

WParam

与消息相关的 wparam。

LParam

与消息相关的 lparam。

PLResult

指向父窗口过程返回值的指针。如果没有返回值，则这个指针可以是 NULL。

说明

当窗口的父窗口接收到这个窗口有关的通知消息时，就调用这个成员函数。

永远不要直接调用这个成员函数。

这个成员函数的缺省实现返回 0，这意味着父窗口将处理消息。

重载这个成员函数以扩展响应通知消息的方式。

CWnd::OnClose

```
afx_msg void OnClose( );
```

说明

框架调用这个函数，作为 CWnd **或应用程序要被关闭的信号。缺省的实现调用** DestroyWindow。

请参阅 CWnd::DestroyWindow, WM_CLOSE

CWnd::OnCommand

virtual BOOL OnCommand(WPARAM *wParam*, LPARAM *lParam*);

返回值

如果应用程序要处理这个消息，则返回非零值；否则返回 0。

参数

wParam

wParam **的低位字标识了菜单项或控件的命令** ID。**如果消息是控件发出的，则** *wParam* **的高位字标识了通知消息。如果消息是加速键发出的，则高位字为** 1。**如果消息是菜单发出的，则高位字为** 0。

LParam

如果消息是控件发出的，则标识了发出消息的控件；否则 lParam **为** 0。

说明

当用户从菜单中选择了一项，或者子控件发出一个通知消息，或者转换了一个加速键的击键事件，框架就调用这个成员函数。

OnCommand 处理控件通知和 ON_COMMAND 入口的消息映射，并调用相应的成员函数。

在你的派生类中重载这个函数以处理 WM_COMMAND 消息。除非调用了基类的 OnCommand，否则不应处理消息映射。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_COMMAND, CcmdTarget::OnCmdMsg

CWnd::OnCompacting

afx_msg void OnCompacting(UINT *nCpuTime*);

参数

nCpuTime

指定了当前 Windows 压缩内存所耗费的 CPU 时间与执行其它操作所耗费的时间的比例。例如，8000h 代表有 50%的 CPU 时间用于压缩内存。

说明

当 Windows 检测到在 30 秒至 60 秒时间内有多于 12.5%的系统时间被花费在压缩内存上，则框架为索引的顶层窗口调用这个成员函数。这表明系统内存不足。

当 CWnd 对象接收到这个调用时，它应当释放尽可能多的内存，考虑应用程序的活动的等级以及在 Windows 下运行的应用程序总数。应用程序可以调用 Windows 函数以确定有多少个程序正在运行。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_COMPACTING

CWnd::OnCompareItem

afx_msg int OnCompareItem(int *nIDCtl*, LPCOMPAREITEMSTRUCT *lpCompareItemStruct*);

返回值

指明两个项的相对位置。它可能是如下值之一：

值	含义
-1	第一项排在第二项前面
0	第一项与第二项顺序相同
1	第一项排在第二项后面

参数

nIDCtl

发出WM_COMPAREITEM消息的控件的标识符。

LpCompareItemStruct

包含了一个指向COMPAREITEMSTRUCT数据结构的长指针，其中包含了标识符和应用为组合框或列表框中的两项提供的数据。

说明

框架调用这个成员函数以指定排序的自画组合框或列表框中新项的相对位置。

如果组合框或列表框是用CBS_SORT或LBS_SORT风格创建的，则当应用程序加入新项时，Windows向组合框或列表框的拥有者发送一条WM_COMPAREITEM消息。

组合框或列表框中的两项在 *lpCompareItemStruct* 指向的 COMPAREITEMSTRUCT 结构中改变形式。OnCompareItem 必须返回一个能指明哪一项出现在另一项之前的值。通常，Windows 调用这个函数若干次，直到它能够确定新项的确切位置。

如果 COMPAREITEMSTRUCT 结构的 hwndItem 成员属于 ClistBox 或 CcomboBox 对象，则将调用适当的类的虚拟函数 CompareItem。在你继承的 ClistBox 或 CcomboBox 类中重载 CcomboBox::CompareItem 或 ClistBox::CompareItem 以实现项比较。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 COMPAREITEMSTRUCT, WM_COMPAREITEM, ClistBox::CompareItem, CcomboBox::CompareItem

CWnd::OnContextMenu

afx_msg void OnContextMenu(CWnd* *pWnd*, Cpoint *pos*);

参数

pWnd

用户右击鼠标的窗口的句柄。这可以是接收到消息的窗口的一个子窗口。有关处理这个消息的更多信息参见说明部分。

Pos

点击鼠标时光标的位置，用屏幕坐标表示。

说明

当用户在窗口中点击鼠标右键（右击）时，框架调用这个函数。你可以处理这个消息，使用 TrackPopupMenu 显示上下文菜单。

如果你没有显示上下文菜单，你必须将这个消息传递给 DefWindowProc 函数。如果你的窗口是一个子窗口，DefWindowProc 将这个消息发送给父窗口；否则，如果指定的位置是在窗口的标题上，则 DefWindowProc 显示一个缺省的上下文菜单。

CWnd::OnCopyData

afx_msg BOOL OnCopyData(CWnd* *pWnd*, COPYDATASTRUCT* *pCopyDataStruct*);

返回值

如果接收应用程序成功地接收到了数据，则返回TRUE；否则返回FALSE。

参数

pWnd
指向发送数据的CWnd对象的指针。

PCopyDataStruct
指向一个COPYDATASTRUCT结构的指针，其中包含了要被发送的数据。

说明

框架调用这个成员函数以把数据从一个应用程序发送到另一个应用程序。

数据中不能包含接收数据的应用程序所不能访问的对象的指针或其它引用。

在拷贝数据的时候，它不能被发送过程的其它线程所改变。

接收应用程序应认为数据是只读的。PCopyDataStruct参数指向的结构仅在传输数据的时候有效；但是，接收应用程序不应释放与结构相关的内存。

如果接收应用程序需要在函数返回之后访问数据，它必须将接收到的数据拷贝到本地缓冲区中。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_COPYDATA

CWnd::OnCreate

afx_msg int OnCreate(LPCREATESTRUCT *lpCreateStruct*);

返回值

OnCreate **必须返回** 0 **以继续** CWnd **对象的创建过程。如果应用程序返回**-1**，窗口将被销毁。**

参数

lpCreateStruct

指向一个 CREATESTRUCT **结构，其中包含了与要创建的** CWnd **对象有关的信息。**

说明

当应用程序通过调用成员函数 Create 或 CreateEx 请求创建 Windows 的窗口时，框架调用这个成员函数。CWnd 对象在窗口被创建以后，但是在它变为可见之前接收到对这个函数的调用。OnCreate 是在 Create 或 CreateEx 成员函数返回之前被调用的。

重载这个成员函数以执行派生类所需的初始化工作。

CREATESTRUCT 结构中包含了用于创建窗口的参数的拷贝。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::CreateEx, CWnd::OnNcCreate, WM_CREATE, CWnd::Default, CWnd::FromHandle

CWnd::OnCtlColor

afx_msg HBRUSH OnCtlColor(CDC* *pDC*, CWnd* *pWnd*, UINT *nCtlColor*);

返回值

OnCtlColor必须返回一个刷子句柄，该刷子将被用于画出控件的背景。

参数

pDC

包含了子窗口的显示设备环境的指针。可能是临时的。

PWnd

包含了要求颜色的控件的指针。可能是临时的。

NCtlColor

包含了下列值，指定了控件的类型：

- CTLCOLOR_BTN 按钮控件
- CTLCOLOR_DLG 对话框
- CTLCOLOR_EDIT 编辑控件
- CTLCOLOR_LISTBOX 列表框控件
- CTLCOLOR_MSGBOX 消息框
- CTLCOLOR_SCROLLBAR 滚动条控件
- CTLCOLOR_STATIC 静态控件

说明

当要画出一个子控件时，框架就调用这个成员函数。多数控件将这个消息发送到它们的父窗口（通常是一个对话框），为使用正确的颜色画出控件而准备pDC。

要改变文本的颜色，使用要求的红、绿、蓝色值（RGB）调用SetTextColor成员函数。

要改变单行编辑控件的背景颜色，在CTLCOLOR_EDIT和CTLCOLOR_MSGBOX消息代码中设置刷子句柄，并在响应CTLCOLOR_EDIT的代码中调用CDC::SetBkColor函数。

不会为下拉组合框中的列表框调用OnCtlColor函数，因为下拉列表框实际上是组合框的子窗口，而不是窗口的子窗口。要改变下拉列表框的颜色，创建一个CcomboBox，在重载的OnCtlColor中的nCtlColor参数中检查CTLCOLOR_LISTBOX。在这个处理函数中，为设置文本的背景必须使用SetBkColor成员函数。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CDC::SetBkColor

CWnd::OnDeadChar

afx_msg void OnDeadChar(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar

指定死键的字符值。

NRepCnt

指定重复计数。

NFlags

指定了扫描码、键暂态码、以前的按键状态和上下文代码，如下面的列表所示：

值	含义
0—7	扫描码。高位字的低字节
8	扩展键，例如功能键或数字键盘上的键（如果为扩展键则值为 1，否则为 0）
9-10	没有使用
11—12	由 Windows 内部使用
13	上下文代码（如果按下键的时候 ALT 键是被放开的，则返回 1；否则返回 0）

续表

14	以前的键状态（如果此函数被调用前该键是被按下的，则为 1；如果该键是放开的，则该值为 0）
15	暂态（如果键正被放开，则为 1，如果正被按下，则为 0）

说明

当 OnKeyUp 成员函数和 OnKeyDown 成员函数被调用的时候，框架调用这个函数。这个成员函数可以被用来指定死键的字符值。死键是这样一种键，例如元音字符（双点），它与其它字符组合成符合字符。例如，元音 O 是由死键、元音和 O 键组成的。

通常应用程序使用 OnDeadChar 来对用户每次按键作反馈。例如，应用程序可以在当前字符位置显示一个重音字符，而不需要移动插字符。

由于按键和 OnDeadChar 调用不必是一一对应的，因此对应用程序来说，nFlags 中的信息通常是没有用的。nFlags 中的信息只对在 OnDeadChar 调用之前最近的 OnKeyUp 和 OnKeyDown 成员函数的调用有效。

对于 IBM 增强型 101 和 102 键键盘，键盘的主体部分的增强键是右边的 ALT 和 CTRL 键；还有数字键盘左侧的 INS，DEL，HOME，END，PAGE UP，PAGE DOWN，箭头键；以及数字键盘上的斜杠（/）和 ENTER 键等。其它键盘可能会支持 nFlags 中的扩展键位。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　WM_DEADCHAR

CWnd::OnDeleteItem

afx_msg void OnDeleteItem(int *nIDCtl*, LPDELETEITEMSTRUCT *lpDeleteItemStruct*);

参数

nIDCtl

发出 WM_DELETEITEM **消息的控件的标识符。**

lpDeleteItemStruct

指定了指向 DELETEITEMSTRUCT **结构的长指针，其中包含有关要删除的列表框项的信息。**

说明

框架调用这个函数以通知自画列表框或组合框的拥有者，列表框或组合框将被销毁，或者用 CComboBox::DeleteString，CListBox::DeleteString，CComboBox::ResetContent 或 CListBox::ResetContent 清除其中的项。

如果 DELETEITEMSTRUCT 结构中的 hwndItem 成员属于组合框或列表框，则将调用适当的类的虚拟函数 DeleteItem。重载适当的控件类中的 DeleteItem 成员函数以删除与项有关的数据。

> 注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CComboBox::DeleteString, CListBox::DeleteString, CComboBox::ResetContent, CListBox::ResetContent, WM_DELETEITEM, CListBox::DeleteItem, CComboBox::DeleteItem

CWnd::OnDestroy

```
afx_msg void OnDestroy( );
```

返回值

框架调用这个成员函数以通知 CWnd 对象它将被销毁。OnDestroy 是在 CWnd 对象已经从屏幕上清除以后被调用的。

首先为被销毁的 CWnd 调用 OnDestroy，然后当 CWnd 的子窗口被销毁时为它们调用 OnDestroy。可以假定当 OnDestroy 运行的时候，所有的子窗口依然存在。

如果被销毁的 CWnd 对象是剪贴板观察器链（通过调用 SetClipboardViewer 成员函数设置）的一部分，CWnd 必须在从 OnDestroy 函数返回之前调用 ChangeClipboardChain 成员函数，将自己从剪贴板观察器链中清除。

请参阅 CWnd::ChangeClipboardChain, CWnd::DestroyWindow, CWnd::SetClipboardViewer

CWnd::OnDestroyClipboard

```
afx_msg void OnDestroyClipboard( );
```

说明

当通过调用 Windows 的 EmptyClipboard 函数清空剪贴板时，框架为剪贴板的拥有者调用这个成员函数。

请参阅 ::EmptyClipboard, WM_DESTROYCLIPBOARD

CWnd::OnDeviceChange

afx_msg BOOL OnDeviceChange(UINT *nEventType*, DWORD *dwData*);

参数

nEventType

　　事件类型。有关可能取值的描述参见说明部分。

dwData

　　包含了与事件有关的数据的结构的地址。它的含义依赖于给定的事件。

说明

框架调用这个函数以通知应用程序或设备驱动程序，设备或计算机的硬件配置发生了改变。

对于提供了软件控制功能，如弹出和锁定的设备，操作系统通常发送一条DBT_DEVICEREMOVEPENDING消息，以便使应用程序和设备驱动程序停止对设备的使用。

如果操作系统强行清除了一个设备，它可能不会发送

DBT_DEVICEQUERYREMOVE 消息。

nEvent 参数可以是下列值之一：

- DBT_DEVICEARRIVAL　已经加入了一个设备，现在可以使用。
- DBT_DEVICEQUERYREMOVE　允许清除被请求的设备。任何应用程序都可以拒绝这个请求并取消清除操作。
- DBT_DEVICEQUERYREMOVEFAILED　清除设备的请求被取消了。
- DBT_DEVICEREMOVEPENDING　设备将要被清除。不能拒绝。
- DBT_DEVICEREMOVECOMPLETE　设备已经被清除。
- DBT_DEVICETYPESPECIFIC　与设备有关的事件。
- DBT_CONFIGCHANGED　当前配置发生了变化。
- DBT_DEVNODES_CHANGED　设备节点发生了变化。

> **注意**　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　WM_DEVICECHANGE

CWnd::OnDevModeChange

afx_msg void OnDevModeChange(LPTSTR *lpDeviceName*);

参数

lpDeviceName

指向 Windows 初始化文件 WIN.INI 中指定的设备名。

说明

当用户改变设备模式设置时，框架为所有的顶层 CWnd 对象调用这个成员函数。

处理 WM_DEVMODECHANGE 消息的应用程序可以重新初始化它们的设备模式设置。通常，使用 Windows 的 ExtDeviceMode 函数来保存和恢复设备设置的应用程序不处理这个函数。

当用户在控制面板中改变缺省打印机时，不会调用这个函数。在这种情况下，将调用 OnWinIniChange 函数。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_DEVMODECHANGE

CWnd::OnDrawClipboard

afx_msg void OnDrawClipboard( );

说明

当剪贴板的内容发生变化时，框架为剪贴板观察器链中的每个窗口调用这个成员函数。只有那些通过调用 SetClipboardViewer 成员函数加入了剪贴板观察器链的应用程序才需要响应这个调用。

接收到 OnDrawClipboard 的每个窗口必须调用 Windows 的 SendMessage 函数以向剪贴板观察器链中的下一个窗口传递一个 WM_DRAWCLIPBOARD 消息。下一个窗口的句柄是通过 SetClipboardViewer 成员函数返回的；它可能在响应 OnChangeCbChain 成员函数调用的时候被修改。

请参阅 ::SendMessage, CWnd::SetClipboardViewer, WM_CHANGECBCHAIN, WM_DRAWCLIPBOARD

CWnd::OnDrawItem

afx_msg void OnDrawItem(int *nIDCtl*, LPDRAWITEMSTRUCT *lpDrawItemStruct*);

参数

nIDCtl

包含了发送 WM_DRAWITEM 消息的控件的标识符。如果菜单发送了此消息，则 nIDCtl 中包含 0。

lpDrawItemStruct

指定了指向 DRAWITEMSTRUCT 数据结构的长指针，其中包含有关要画出的项和要求的绘图类型的信息。

说明

当控件或菜单的可视状态发生变化时，框架为自画按钮控件、组合框控件、列表框控件或者菜单的拥有者调用这个成员函数。

DRAWITEMSTRUCT 结构的 itemAction 成员定义了要执行的绘图操作。这个成员中的数据允许控件的拥有者确定需要什么绘图动作。

在从处理这个消息的过程中返回之前，应用程序必须确保 DRAWITEMSTRUCT 的 hDC 成员所标识的设备环境已经恢复到缺省状态。

如果 hwndItem 成员属于 CButton、CMenu、CListBox 或 CComboBox 对象，则将调用适当的类的虚函数 DrawItem。重载适当的控件类的 DrawItem 成员函数以画出项。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 DRAWITEMSTRUCT, WM_DRAWITEM, CButton::DrawItem, CMenu::DrawItem, CListBox::DrawItem, CComboBox::DrawItem

CWnd::OnDropFiles

afx_msg void OnDropFiles(HDROP *hDropInfo*);

参数

hDropInfo

指向描述下放文件的内部数据结构的指针。这个句柄被 Windows**的**DragFinish**，**DragQueryFile**和**DragQueryPoint**函数所使用，用来获得与下放文件有关的信息。**

说明

当用户在注册为下放文件接收者的窗口上方放开鼠标左键时，框架就调用这个成员函数。

通常，派生类被设计为支持下放文件，并且将在窗口构造过程中注册。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::DragAcceptFiles, WM_DROPFILES, ::DragAcceptFiles, ::DragFinish, ::DragQueryFile, ::DragQueryPoint

CWnd::OnDSCNotify

afx_msg BOOL OnDSCNotify(DSCSTATE *nState*, DSCREASON *nReason*, BOOL *pBool*);

返回值

如果 *nReason* 和 *nState* 所标识的操作被处理了，则返回 TRUE；否则返回 FALSE。

参数

nState

DSCSTATE 枚举量中的一个命名常量，列在 Remarks 部分。

nReason

DSCSTATE 枚举量中的一个命名常量，列在 Remarks 部分。

pBool

一个布尔型结果，指明 *nState* 和 *nReason* 代表的操作是否应当继续。

说明

当与数据源控件相绑定的控件修改或试图修改游标时，就在响应数据源控件引发的事件时调用这个接收方通知。使用它来跟踪数据源控件产生的原因（nReason）和状态（nState）。缺省情况下允许状态和原因的任意组合。编写你自己的代码以测试对你的应用程序而言是重要的状态和原因，然后返回适当的 TRUE 或 FALSE。

要使用 OnDSCNotify，在需要接收接收方通知的类的头文件中定义一个接收映射以及接收方通知的处理函数如下：

```
class CMyDlg : public CDialog
{
    ...
    DECLARE_EVENTSINK_MAP()
    BOOL OnDSCNotify(DSCSTATE nState,
                     DSCREASON nReason, BOOL* pBool);
    ...
```

```
};
```

然后，在你的类的实现中，定义接收映射并指定接收事件的函数如下：

```
BEGIN_EVENTSINK_MAP(CMyDlg, CDialog)
	ON_DSCNOTIFY(CMyDlg, IDC_RDCCTRL1, OnDSCNotify)
END_EVENTSINK_MAP()
```

当事件在数据源控件内发生时，通知回调函数，即你实现的 OnDSCNotify 将被调用。

```
enum DSCREASON
{
	dscNoReason = 0,
	dscClose, dscCommit, dscDelete,
	dscEdit, dscInsert, dscModify, dscMove
};
```

对于下面列出的每一种状态，它将被调用多次：

```
enum DSCSTATE
{
	dscNoState = 0,
	dscOKToDo,
	dscCancelled,
	dscSyncBefore,
	dscAboutToDo,
```

```
    dscFailedToDo,
    dscSyncAfter,
    dscDidEvent
};
```

多次调用使你能够以不同的次数跟踪事件。例如，由于事件通常是在响应控件对游标的修改时产生的，数据源控件需要做的第一件事情就是引发一个事件，询问是否可以真正执行这个动作；因此是 dsdOKToDo 状态的原因。如果监控事件（数据控件、应用程序等等）的所有客户都接收了事件，数据源控件将进入 dscSyncBefore 状态，此时如有必要，所有外面的数据都将被刷新。例如，如果编辑域的内容发生了变化，这个变化将被发送到游标。在这个事件之后，数据源控件将进入 dscAboutToDo 和 dscSyncAfter 状态，最后进入 dscDidEvent 状态。这些为你提供了从数据源空捕捉通知的更多机会。

请参阅 CWnd::GetDSCCursor, CWnd::BindDefaultProperty, CWnd::BindProperty

CWnd::OnEnable

afx_msg void OnEnable(BOOL *bEnable*);

参数

bEnable

指定了 CWnd 对象是被允许的还是被禁止的。如果 CWnd 是允许的，则这个参数为 TRUE；如果 CWnd 是被禁止的，则为 FALSE。

说明

当应用程序改变 CWnd 对象的允许状态时，框架调用这个成员函数。OnEnable 在 EnableWindow 成员函数返回之前被调用，但是是在窗口的允许状态（WS_DISABLE 风格位）改变之后。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　CWnd::EnableWindow, WM_ENABLE

CWnd::OnEndSession

afx_msg void OnEndSession(BOOL *bEnding*);

参数

bEnding

指定了会话过程是否要结束。如果要结束会话，则为 TRUE；否则为

FALSE。

说明

在 CWnd 对象从 OnQueryEndSession 成员函数中返回一个非零值之后，框架调用这个成员函数。OnEndSession 通知 CWnd 对象会话确实将要结束。

如果 *bEnding* 为 TRUE，则在所有的应用程序从这个函数中返回之后，Windows 可以在任何时候结束。因此，应当使应用程序在 OnEndSession 内部完成结束时所需的所有任务。

当会话结束的时候，你不必调用 DestroyWindow 成员函数或 Windows 的 PostQuitMessage 函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::DestroyWindow, CWnd::OnQueryEndSession, ::ExitWindows, ::PostQuitMessage, WM_QUERYENDSESSION, CWnd::Default, WM_ENDSESSION

CWnd::OnEnterIdle

afx_msg void OnEnterIdle(UINT *nWhy*, CWnd* *pWho*);

参数

nWhy

指明该消息是显示对话框还是显示菜单的结果。这个参数可以是下列值之一：

- MSGF_DIALOGBOX **系统空闲，因为正在显示对话框。**
- MSGF_MENU **系统空闲，因为正在显示菜单。**

pWho

指定了对话框指针（如果 *nWhy* **为** MSGF_DIALOGBOX**），或者是包含了显示的菜单（如果** *nWhy* **为** MSGF_MENU**）窗口的指针。这个指针可能是临时的，不能被保存以供将来使用。**

说明

框架调用这个函数以通知应用程序的主窗口过程，有一个模式对话框或菜单正在进入空闲状态。当模式对话框或菜单处理完原来的

一条或多条消息，并且没有其它消息在队列中等候时，它就进入空闲状态。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_ENTERIDLE

CWnd::OnEnterMenuLoop

afx_msg void OnEnterMenuLoop(BOOL *bIsTrackPopupMenu*);

参数

bIsTrackPopupMenu

指定了涉及的菜单是否是一个弹出菜单。如果函数成功，则具有非零值；否则为 0。

说明

当进入了菜单模式循环时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnExitMenuLoop, WM_ENTERMENULOOP

CWnd::OnEraseBkgnd

afx_msg BOOL OnEraseBkgnd(CDC* *pDC*);

返回值

如果它擦除了背景，则返回非零值；否则返回 0。

参数

pDC

指定了设备环境对象。

说明

当 CWnd 对象的背景需要被擦除时（例如，当窗口大小被改变时），框架就调

用这个函数。它被调用以便为绘图准备无效区域。

缺省的实现使用窗口类结构中 hbrBackground 成员指定的窗口类背景刷子擦除窗口背景。

如果 hbrBackground 成员为 NULL，你重载的 OnEraseBkgnd 必须擦除背景色。你的重载函数也可以为刷子调用 UnrealizeObject 函数，将目标刷子的原点与 CWnd 的坐标对齐，然后选择该刷子。

如果重载的 OnEraseBkgnd 在响应 WM_ERASEBKGND 时处理了这个消息并擦除了背景，则应当返回非零值，表明不需要进一步擦除。如果它返回 0，则窗口依然被标记为需要擦除（通常，这意味着 PAINTSTRUCT 结构的 fErase 成员将为 TRUE）。

Windows 假定背景是用 MM_TEXT 映射模式计算的。如果设备环境使用了其它映射模式，则擦除的区域可能不在客户区的可见部分之内。

> **注意**　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_ICONERASEBKGND, CGdiObject::UnrealizeObject, WM_ERASEBKGND

CWnd::OnExitMenuLoop

afx_msg void OnExitMenuLoop(BOOL *bIsTrackPopupMenu*);

参数

bIsTrackPopupMenu

指明涉及的菜单是否是一个弹出菜单。如果函数成功，则具有非零值；否则为 0。

说明

当退出菜单模式循环时，框架调用这个函数。

注意 **框架调用这个成员函数以允许你的应用程序处理一个** Windows **消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。**

请参阅 CWnd::OnEnterMenuLoop; WM_EXITMENULOOP

CWnd::OnFontChange

afx_msg void OnFontChange( );

说明

当应用程序改变了字体资源池之后，系统中的所有顶层窗口从框架接收到一个OnFontChange 调用。

在系统中加入或删除字体（例如，通过 Windows 的 AddFontResource 或 RemoveFontResource 函数）的应用程序必须向所有的顶层窗口发送 WM_FONTCHANGE 消息。

要发送这个消息，使用 Windows 的 SendMessage 函数，把 hWnd 参数设为 HWND_BROA-DCAST。

请参阅 ::AddFontResource, ::RemoveFontResource, ::SendMessage, WM_FONTCHANGE

CWnd::OnGetDlgCode

```
afx_msg UINT OnGetDlgCode( );
```

返回值

下列值中的一个或多个，指明了应用程序处理的输入类型：

- DLGC_BUTTON 按钮（通用）。

- DLGC_DEFPUSHBUTTON 缺省按钮。
- DLGC_HASSETSEL EM_SETSEL消息。
- DLGC_UNDEFPUSHBUTTON 没有缺省的按钮处理。（应用程序可以与DLGC_BUTTON一起使用这个标志，指明它处理按钮输入，但是依靠系统进行缺省按钮处理）
- DLGC_RADIOBUTTON 单选按钮。
- DLGC_STATIC 静态控件。
- DLGC_WANTALLKEYS 所有的键盘输入。
- DLGC_WANTARROWS 箭头键。
- DLGC_WANTCHARS WM_CHAR消息。
- DLGC_WANTMESSAGE 所有的键盘输入。应用程序将这个消息传递给控件。
- DLGC_WANTTAB TAB键。

说明

通常，Windows处理CWnd控件中所有的箭头键和TAB键输入。通过重载OnGetDlgCode，CWnd控件可以选择处理特定类型的输入。

预定义控件类的缺省OnGetDlgCode函数返回与每个类相对应的代码。

请参阅 WM_GETDLGCODE

CWnd::OnGetMinMaxInfo

afx_msg void OnGetMinMaxInfo(MINMAXINFO FAR* *lpMMI*);

参数

lpMMI

指向一个MINMAXINFO结构，其中包含了有关窗口的最大化大小和位置以及最小、最大跟踪大小的信息。有关这个结构的更多信息参见MINMAXINFO结构。

说明

每当Windows需要知道窗口的最大化位置或大小，或者最小、最大的跟踪大小时，框架就调用这个成员函数。最大化大小是指当窗口的边框被完全扩展时窗口的大小。窗口的最大跟踪大小是指用边框改变窗口的大小时可以达到的最大窗口大小。窗口的最小跟踪大小是指用边框改变窗口大小时可以达到的最小窗口大小。

Windows填充一个点组成的数组，为不同的位置和大小指定了缺省值。应用程序可以在OnGetMinMaxInfo中改变这些值。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。

如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_GETMINMAXINFO

CWnd::OnHelpInfo

```
afx_msg BOOL OnHelpInfo( HELPINFO* lpHelpInfo );
```

参数

lpHelpInfo

指向 HELPINFO 结构的指针，其中包含有关菜单项、控件、对话框或请求帮助的窗口的信息。

说明

当用户按下 F1 键时，框架调用这个函数。

如果当按下 F1 时菜单是激活的，则 WM_HELP 被发送到与菜单相关的窗口，否则 WM_HELP 被发送到拥有键盘焦点的窗口。如果没有窗口拥有键盘焦点，则 WM_HELP 被发送到当前的活动窗口。

请参阅 CWinApp::OnHelp, CWinApp::WinHelp

CWnd::OnHScroll

afx_msg void OnHScroll(UINT *nSBCode*, UINT *nPos*, CScrollBar* *pScrollBar*);

参数

nSBCode

指定了滚动条代码，指明了用户的滚动请求。这个参数可以是下列值之一：

- SB_LEFT **滚动到最左边。**
- SB_ENDSCROLL **结束滚动。**
- SB_LINELEFT **向左滚动。**
- SB_LINERIGHT **向右滚动。**
- SB_PAGELEFT **向左滚动一页。**
- SB_PAGERIGHT **向右滚动一页。**
- SB_RIGHT **滚动到最右边。**
- SB_THUMBPOSITION **滚动到绝对位置。当前的位置由** *nPos* **参数指定。**
- SB_THUMBTRACK **将滚动块拖动到指定的位置。当前的位置由** *nPos* **参数指定。**

nPos

如果滚动条代码为 SB_THUMBPOSITION 或者 SB_THUMBTRACK，则指定了滚动块的位置；否则没有使用。取决于初始的滚动范围，*nPos* 可能是负的，如有必要应该被强制转换为整数。

pScrollBar

如果滚动消息来自一个滚动条控件，则包含了控件的指针。如果用户点击了窗口的滚动条，则这个参数为 NULL。这个参数可能时临时的，不能被保存以供将来使用。

说明

当用户点击窗口的水平滚动条时，框架调用这个成员函数。

通常在滚动块要被拖动的时候，滚动条代码 SB_THUMBTRACK 被应用程序用来给出一些反馈。

如果应用程序滚动了滚动条所控制的内容，它必须用 SetScrollPos 成员函数复位滚动块的位置。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::SetScrollPos, WM_VSCROLL, WM_HSCROLL

CWnd::OnHScrollClipboard

afx_msg void OnHScrollClipboard(CWnd* *pClipAppWnd*, UINT *nSBCode*, UINT *nPos*);

参数

pClipAppWnd

　　指定指向剪贴板观察器窗口的指针。这个指针可能时暂时的，因而不应该被保存以供将来使用。

nSBCode

　　在低位字指定滚动条代码，此代码是下列值之一：

- SB_BOTTOM **滚动到右下角**
- SB_ENDSCROLL **结束滚动**
- SB_LINEDOWN **下滚一行**
- SB_LINEUP **上滚一行**
- SB_PAGEDOWN **下滚一页**
- SB_PAGEUP **上滚一页**
- SB_THUMBPOSITION **滚动到绝对位置。当前位置由** nPos **提供。**
- SB_TOP **滚动到左上角**

nPos

如果滚动条代码为 SB_THUMBPOSITION，*nPos* 包含滚动块位置；否则不使用。

说明

当剪贴板数据具有 CF_OWNERDISPLAY 格式并且剪贴板观察器的水平滚动条发生一个事件时，剪贴板所有者的 OnHScrollClipboard 成员函数被剪贴板观察器所调用。剪贴板所有者应滚动剪贴板图象，使适当的区域无效，并更新滚动块的值。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnVScrollClipboard, WM_HSCROLLCLIPBOARD

CWnd::OnIconEraseBkgnd

afx_msg void OnIconEraseBkgnd(CDC* *pDC*);

参数

pDC

指定图标的设备环境对象。该指针可能是临时的，并且不应为将来的使用保存该指针。

说明

当在画出图标之前必须填充图标的背景时，框架为一个最小化 CWnd 对象调用这个成员函数。仅当在窗口的缺省实现中定义了类图标时 CWnd 才会接收该调用；否则调用 OnEraseBkgnd。DefWindowProc 成员函数用父窗口的背景刷子填充图标的背景。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnEraseBkgnd, WM_ICONERASEBKGND

CWnd::OnInitMenu

afx_msg void OnInitMenu(CMenu* *pMenu*);

参数

pMenu

指定了要初始化的菜单。可能是临时的，不应被保存以供将来使用。

说明

当菜单要被激活时，框架调用这个成员函数。这个调用在用户点击菜单条上的菜单项或者按下了菜单键时产生。重载这个函数以在显示之前修改菜单。

OnInitMenu 仅在第一次访问菜单时调用；对于每次访问，OnInitMenu 仅调用一次。这意味着，例如，当保持按钮被按下时，在几个菜单项之间移动鼠标不会产生新的调用。这个调用不提供有关菜单项的信息。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　CWnd::OnInitMenuPopup, WM_INITMENU

CWnd::OnInitMenuPopup

afx_msg void OnInitMenuPopup(CMenu* *pPopupMenu*, UINT *nIndex*, BOOL *bSysMenu*);

参数

pPopupMenu

指定了弹出菜单的菜单对象。可能时临时的，不能被保存以供将来使用。

nIndex

指定了主菜单中弹出菜单的索引。

bSysMenu

如果弹出菜单为控制菜单，则为TRUE；否则为FALSE。

说明

当一个弹出菜单将被激活时，框架调用这个成员函数。这允许应用程序在弹出菜单被显示之前修改它，而不需要改变整个菜单。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnInitMenu, WM_INITMENUPOPUP

CWnd::OnKeyDown

afx_msg void OnKeyDown(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar

指定了给定键的虚拟键码。

nRepCnt

重复计数（用户按住键引起的重复击键数目）。

nFlags

指定了扫描码、暂态键码、原来的键状态和上下文代码，如下面的列表所示：

值	描述
0–7	扫描码（依赖于 OEM 的值）
8	扩展键，比如功能键或数字键盘上的键（如果它是扩展键，则为 1）
9–10	未使用
11–12	Windows 内部使用
13	上下文代码（如果按下键时 ALT 键时被按下的，则为 1；否则为 0）

续表

14	原来的键状态（如果在调用之前键时按下的，则为 1；如果键是弹起的，则为 0）
15	暂态（如果键正被释放，则为 1；如果键正被按下，则为 0）

对于 WM_KEYDOWN 消息，键暂态位（15 位）为 0，并且上下文代码位（13 位）为 0。

说明

当用户按下了一个非系统键时，框架调用这个成员函数。非系统键是指当 ALT 键为被按下时按下的键盘键或者当 CWnd 拥有输入焦点时按下的键盘键。

由于自动重复，在调用 OnKeyUp 成员函数之前可能会产生多个 OnKeyDown 调用。指明原来的键状态的位可以被用来确定 OnKeyDown 调用时是第一次被按下还是重复的按下状态。

对于 IBM 增强 101 和 102 键键盘，增强键包括键盘主体部分的右 ALT 键和右 CTRL 键；数字键盘左侧的 INS，DEL，HOME，END，PAGE UP，PAGE DOWN 和箭头键；以及数字键盘上的斜杠（/）和 ENTER 键。一些其它的键盘可能支持 nFlags 中的扩展键位。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。

如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_CHAR, WM_KEYUP, WM_KEYDOWN

CWnd::OnKeyUp

```
afx_msg void OnKeyUp( UINT nChar, UINT nRepCnt, UINT nFlags );
```

参数

nChar

指定了给定键的虚拟键码。

nRepCnt

重复计数（用户按住键引起的重复击键数目）。

nFlags

指定了扫描码、暂态键码、原来的键状态和上下文代码，如下面的列表所示：

值	描述
0–7	扫描码（依赖于OEM的值）。高位字的低字节
8	扩展键，比如功能键或数字键盘上的键（如果它时扩展键则为1）
9–10	未使用

续表

11–12	Windows内部使用
13	上下文代码（如果按下键时ALT键时被按下的，则为1；否则为0）
14	原来的键状态（如果在调用之前键时按下的，则为1；如果键是弹起的，则为0）
15	暂态（如果键正在被释放，则为1；如果键正被按下，则为0）

对于WM_KEYDOWN消息，键暂态位（15位）为1，并且上下文代码位（13位）为0。

说明

当一个非系统键被释放的时候，框架调用这个成员函数。非系统键是指当ALT键未按下时按下的键盘键，或者是当CWnd拥有输入焦点时按下的键盘键。

对于IBM增强101和102键键盘，增强键包括键盘主体部分的右ALT键和右CTRL键；数字键盘左侧的INS，DEL，HOME，END，PAGE UP，PAGE DOWN和箭头键；以及数字键盘上的斜杠（/）和ENTER键。一些其它的键盘可能支持nFlags中的扩展键位。

> **注意**　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的

参数（而不是你提供给这个函数的参数）。

请参阅 WM_CHAR, WM_KEYUP, CWnd::Default, WM_KEYDOWN

CWnd::OnKillFocus

afx_msg void OnKillFocus(CWnd* *pNewWnd*);

参数

pNewWnd

指定了获得输入焦点的窗口指针（可能为NULL，或可能是临时的）。

说明

框架在失去输入焦点之后立刻调用这个成员函数。

如果CWnd显示了一个插字符，则此时必须销毁插字符。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::SetFocus, WM_KILLFOCUS

CWnd::OnLButtonDblClk

afx_msg void OnLButtonDblClk(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL **如果** CTRL **键被按下，则设置此位。**
- MK_LBUTTON **如果鼠标左键被按下，则设置此位。**
- MK_MBUTTON **如果鼠标中键被按下，则设置此位。**
- MK_RBUTTON **如果鼠标右键被按下，则设置此位。**
- MK_SHIFT **如果** SHIFT **键被按下，则设置此位。**

point

指定了光标的 x **和** y **轴坐标。这些坐标通常是相对于窗口的左上角的。**

说明

当用户双击鼠标左键时框架调用这个成员函数。

只有具有 CS_DBLCLKS WNDCLASS **风格的窗口才接收** OnLButtonDlbClk **调**

用。这是微软基础类窗口的缺省状态。当用户按下、释放，然后在系统规定的双击时间限制之内再次按下鼠标左键时，Windows 就调用 OnLButtonDblClk。双击鼠标左键实际产生四个事件：WM_LBUTTONDOWN，WM_LBUTTONUP 消息，WM_LBUTTONDBLCLK 调用，以及释放按钮时的另一个 WM_LBUTTONUP 消息。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnLButtonDown, CWnd::OnLButtonUp, WM_LBUTTONDBLCLK

CWnd::OnLButtonDown

afx_msg void OnLButtonDown(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果 CTRL 键被按下，则设置此位。

- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果 SHIFT 键被按下，则设置此位。

point
指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户按下鼠标左键时，框架调用这个成员函数。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnLButtonDblClk, CWnd::OnLButtonUp, WM_LBUTTONDOWN

CWnd::OnLButtonUp

afx_msg void OnLButtonUp(UINT *nFlags*, CPoint *point*);

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果CTRL键被按下，则设置此位。
- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果SHIFT键被按下，则设置此位。

point

指定了光标的x和y轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户放开鼠标左键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnLButtonDblClk, CWnd::OnLButtonDown, WM_LBUTTONUP

CWnd::OnMButtonDblClk

afx_msg void OnMButtonDblClk(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL **如果** CTRL **键被按下，则设置此位。**
- MK_LBUTTON **如果鼠标左键被按下，则设置此位。**
- MK_MBUTTON **如果鼠标中键被按下，则设置此位。**
- MK_RBUTTON **如果鼠标右键被按下，则设置此位。**
- MK_SHIFT **如果** SHIFT **键被按下，则设置此位。**

point

指定了光标的 x **和** y **轴坐标。这些坐标通常是相对于窗口的左上角的。**

说明

当用户双击鼠标中键时框架调用这个成员函数。

只有具有 CS_DBLCLKS WNDCLASS **风格的窗口才接收** OnMButtonDlbClk **调用。这是微软基础类窗口的缺省状态。当用户按下、释放，然后在系统规定的**

双击时间限制之内再次按下鼠标中键时，Windows 就调用 OnMButtonDblClk。双击鼠标中键实际产生四个事件：WM_MBUTTONDOWN，WM_MBUTTONUP 消息，WM_MBUTTONDBLCLK 调用，以及释放按钮时的另一个 WM_MBUTTONUP 消息。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnMButtonDown, CWnd::OnMButtonUp, WM_MBUTTONDBLCLK

CWnd::OnMButtonDown

afx_msg void OnMButtonDown(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON 如果鼠标左键被按下，则设置此位。

- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户按下鼠标中键时，框架调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnMButtonDblClk, CWnd::OnMButtonUp, WM_MBUTTONDOWN

CWnd::OnMButtonUp

afx_msg void OnMButtonUp(UINT *nFlags*, CPoint *point*);

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户放开鼠标中键时，框架调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnMButtonDblClk, CWnd::OnMButtonDown, WM_MBUTTONUP

CWnd::OnMDIActivate

afx_msg void OnMDIActivate(BOOL *bActivate*, CWnd* *pActivateWnd*, CWnd* *pDeactivate- Wnd*);

参数

bActivate

如果子窗口要被激活，则为 TRUE；如果要被取消激活状态，则为 FALSE。

pActivateWnd

包含了要激活的 MDI 子窗口的指针。当被一个 MDI 子窗口接收的时候，*pActiveWnd* 中包含了要激活的子窗口指针。这个指针可能时临时的，不应被保存以供将来使用。

pDeactivateWnd

包含了将失去激活状态的 MDI 子窗口的指针。这个指针可能时临时的，不应被保存以供将来使用。

说明

框架为被激活的子窗口和取消激活状态的子窗口调用这个成员函数。

一个 MDI 子窗口可以与 MDI 框架窗口独立地被激活。当框架窗口被激活时，

最近在OnMDIActive调用中被激活的子窗口接收到一个WM_NCACTIVATE消息以画出活动窗口的边框和标题条，但是它不接收另一个OnMDIActivate调用。

> 注意　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　CMDIFrameWnd::MDIActivate, WM_MDIACTIVATE

CWnd::OnMeasureItem

afx_msg void OnMeasureItem(int *nIDCtl*, LPMEASUREITEMSTRUCT *lpMeasureItemSt- ruct*);

参数

nIDCtl

　　控件的ID。

lpMeasureItemStruct

　　指向一个MEASUREITEMSTRUCT数据结构，其中包含自画控件的大小。

说明

当控件被创建的时候，框架为自画按钮、组合框、列表框或菜单项调用这个成员函数。

重载这个函数并填充 lpMeasureItemStruct 指向的 MEASUREITEMSTRUCT 数据结构，然后返回；这将通知 Windows 控件的大小，并使 Windows 能够正确地处理控件的用户交互。

如果列表框或组合框是用 LBS_OWNERDRAWVARIABLE 或 CBS_OWNERDRAWVARIA- BLE 风格创建的，则框架为控件中的每一个项调用这个函数；否则这个函数只被调用一次。

在发送 WM_INITDIALOG 消息之前，Windows 为用 OWNERDRAWFIXED 风格创建的组合框和列表框的拥有者发出对 OnMeasureItem 的调用。其结果是，当拥有者接收到这个调用时，Windows 还没有确定在控件中使用的字体的高度和宽度；需要这些值的函数调用和计算应该发生在应用程序或库的主函数中。

如果要测量的的项是 CMenu，CListBox 或 CComboBox 对象，则将调用适当的类的虚函数 MeasureItem。重载适当的控件类的 MeasureItem 成员函数以计算并设置每个项的大小。

仅当控件类是在运行时创建，或者它是用 LBS_OWNERDRAWVARIABLE 或 CBS_OWNERDRAWVARIABLE 风格创建的时候，OnMeasureItem 才会被调用。这是因为 WM_MEASUREITEM 消息时在控件创建过程的早期被发送的。

如果你使用DDX_Control，SubclassDlgItem或SubclassWindow进行了子类化，则子类化过程通常发生在创建过程之后。因此，在控件的OnChildNotify函数中无法处理WM_MEASUREITEM消息，这是MFC用来实现ON_WM_MEASUREITEM_REFLECT的机制。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CMenu::MeasureItem, CListBox::MeasureItem, CComboBox::MeasureItem, WM_MEASUREITEM

CWnd::OnMenuChar

afx_msg LRESULT OnMenuChar(UINT *nChar*, UINT *nFlags*, CMenu* *pMenu*);

返回值

返回值的高位字中必须包含下列命令代码之一：

值	描述
0	告诉 Windows 废弃用户按下的字符，并在系统扬声器中产生一个短响
1	告诉 Windows 关闭当前的菜单
2	通知 Windows 在返回值的低位字中包含了指定项的号码。这个项是 Windows 选中的

如果高位字中包含 0 或 1，则低位字被忽略。当使用了加速键（快捷方式）选中菜单中的位图时，应用程序应当处理这个消息。

参数

nChar

依赖于编译设置，指定了用户按下的 ANSI 或 Unicode 字符。

nFlags

如果菜单是弹出菜单，则包含了 MF_POPUP 标志。如果菜单是控制菜单，则包含了 MF_SYSMENU 标志。

pMenu

包含了选中菜单的指针。这个指针可能时临时的，不能被保存以供将来使用。

说明

当用户按下一个不能与当前菜单中任何预定义的助记符相匹配的菜单助记符时，框架就调用这个成员函数。它被发送到拥有这个菜单的 CWnd 对象。当用户按下 ALT 以及其它任何键时，也会调用 OnMenuChar，即使这些键并不对应助记符字符。在这种情况下，pMenu 指向 CWnd 拥有的菜单，nFlags 为 0。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_MENUCHAR

CWnd::OnMenuSelect

afx_msg void OnMenuSelect(UINT *nItemID*, UINT *nFlags* HMENU *hSysMenu*);

参数

nItemID

标识了被选中的项。如果选中项是一个菜单项，则 *nItemID* 包含了菜单项 ID。如果选中项包含了弹出菜单，则 *nItemID* 包含了弹出菜单索引，

而 *hSysMenu* 中包含了主菜单（用户点击的）的句柄。

nFlags

包含下列菜单标志的组合：

- MF_BITMAP 该项是位图。
- MF_CHECKED 该项被选中。
- MF_DISABLED 该项被禁止。
- MF_GRAYED 该项被变灰。
- MF_MOUSESELECT 该项是用鼠标选中的。
- MF_OWNERDRAW 该项是一个自画项。
- MF_POPUP 该项包含了一个弹出菜单。
- MF_SEPARATOR 该项是一个菜单分隔符。
- MF_SYSMENU 该项包含在控制菜单中。

hSysMenu

如果 *nFlags* 中包含了 MF_SYSMENU，标识了与消息相关的菜单。如果 *nFlags* 中包含了 MF_POPUP，则标识了主菜单的句柄。如果 *nFlags* 中既没有 MF_SYSMENU 也没有 MF_POPUP，则没有使用。

说明

如果 CWnd 对象与一个菜单相关联，则当用户选择一个菜单项时，框架调用 OnMenuSelect。

如果nFlags中为0xFFFF并且hSysMenu为0，则Windows已经因为用户按下ESC键或在菜单外点击而关闭了菜单。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_MENUSELECT

CWnd::OnMouseActivate

afx_msg int OnMouseActivate(CWnd* *pDesktopWnd*, UINT *nHitTest*, UINT *message*);

返回值

指定了是否要激活CWnd以及是否放弃鼠标事件。它必须是下列值之一：

- MA_ACTIVATE 激活CWnd对象。
- MA_NOACTIVATE 不激活CWnd对象 。
- MA_ACTIVATEANDEAT 激活CWnd对象并放弃鼠标事件。
- MA_NOACTIVATEANDEAT 不激活CWnd对象并放弃鼠标事件。

参数

pDesktopWnd

指定了要激活的窗口的顶层父窗口的指针。这个指针可能是临时的，不能被保存。

nHitTest

指定了击中测试区域代码。击中测试是用来确定光标的位置的。

message

指定了鼠标消息。

说明

当光标位于非激活窗口内并且用户按下了鼠标按钮时，框架就调用这个成员函数。

缺省的实现在进行任何处理之前把这个消息传递给父窗口。如果父窗口返回TRUE，则处理过程中止。

有关不同的击中测试区域代码的描述参见OnNcHitTest成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的

参数（而不是你提供给这个函数的参数）。

• **请参阅** CWnd::OnNcHitTest, WM_MOUSEACTIVATE

CWnd::OnMouseMove

afx_msg void OnMouseMove(UINT *nFlags*, CPoint *point*);

参数

nFlags

指明是否按下了不同的虚拟键。这个参数可以是下列值的组合：

MK_CONTROL　如果 CTRL 键被按下，则设置此位。

MK_LBUTTON　如果鼠标左键被按下，则设置此位。

MK_MBUTTON　如果鼠标中键被按下，则设置此位。

MK_RBUTTON　如果鼠标右键被按下，则设置此位。

MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当鼠标光标移动时，框架调用这个成员函数。如果鼠标没有被捕获，则 WM_MOUSEMOVE 由鼠标下方的窗口所接收；否则消息被发往捕获了鼠标的窗口。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的

参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::SetCapture, CWnd::OnNCHitTest, WM_MOUSEMOVE

CWnd::OnMouseWheel

afx_msg BOOL OnMouseWheel(UINT *nFlags*, short *zDelta*, CPoint *pt*);

返回值

如果允许鼠标轮滚动，则返回非零值；否则返回 0。

参数

nFlags

指明是否按下了虚拟键。这个参数可以是下列值的组合：

MK_CONTROL 如果 CTRL 键被按下，则设置此位。

- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果 SHIFT 键被按下，则设置此位。

zDelta

指明了旋转的距离。zDelta 值以 WHEEL_DELTA，即 120 的倍数或部分的形式表达。小于零的数表明往回滚动（向着用户），而大于零的数表明滚向远处（离开用户）。用户可以在鼠标软件中改变滚轮设置以反转这种响应。有关这个参数的更多信息参见说明部分。

pt

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户旋转鼠标滚轮并达到滚轮的下一个刻度时，框架就调用这个成员函数。除非被重载，否则 OnMouseWheel 调用 WM_MOUSEWHEEL 的缺省处理。Windows 自动将该消息转发到具有输入焦点的控件或子窗口。Win32 函数 DefWindowProc 将该消息上传到拥有它的窗口。

zDelta 参数是 WHEEL_DELTA 的倍数，它被设为 120。这个值是要采取的动作的开端，这一类动作（比如向前滚动到下一个刻度）必须为每一个 delta 产生。

delta 被设为 120，以允许将来使用更高精度的滚轮，例如没有刻度的自由旋转滚轮。这种设备在每次旋转是可能会发送多个消息，但是每次消息中的值更小。要支持这个可能性，或者可以累计输入的 delta 值，直到达到一个 WHEEL_DELTA（因此你达到与给定 delta 的旋转相同的响应），或者滚动部分行以响应更频繁的消息。你可以选择你的滚动精度并累计 delta 值直到达到 WHEEL_DELTA。

重载这个成员函数以提供你自己的鼠标滚轮滚动特性。

注意 OnMouseWheel 为 Windows NT 4.0 处理消息。对于 Windows 95 或 Windows NT 3.51 的消息处理，应使用 OnRegisteredMouseWheel。

请参阅 mouse_event

CWnd::OnMove

afx_msg void OnMove(int *x*, int *y*);

参数

x

指定了客户区左上角的新 x 轴坐标。对于重叠式和弹出式窗口，坐标是用屏幕坐标给出的，对于子窗口，是用父窗口的客户区坐标给出的。

y

指定了客户区左上角的新 y 轴坐标。对于重叠式和弹出式窗口，坐标是用屏幕坐标给出的，对于子窗口，是用父窗口的客户区坐标给出的。

说明

框架在 CWnd **对象被移动之后调用这个成员函数。**

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的

参数（而不是你提供给这个函数的参数）。

请参阅 WM_MOVE

CWnd::OnMoving

afx_msg void OnMoving(UINT *nSide*, LPRECT *lpRect*);

参数

nSide

被移动的窗口的边界。

lpRect

CRect 或 RECT 的地址，其中包含了项的坐标。

说明

当用户移动 CWnd 对象时框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_MOVING

CWnd::OnNcActivate

afx_msg BOOL OnNcActivate(BOOL *bActive*);

返回值

如果 Windows **必须进行缺省处理，则返回非零值；如果要防止标题条或图标变为非激活状态，则为** 0**。**

参数

bActive

表明需要改变标题条或图标以指明活动或非活动状态。如果要画出活动的标题条或图标，则 *bActive* **参数为** TRUE**。如果要画出非活动的标题条或图标，则为** FALSE**。**

说明

当需要改变非客户区以指明活动和非活动状态时，框架调用这个成员函数。如果 bActive **为** TRUE**，则缺省的实现用活动颜色画出标题条和标题条文本；如果** bActive **为** FALSE**，则用非活动颜色画出。**

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::Default, WM_NCACTIVATE

CWnd::OnNcCalcSize

afx_msg void OnNcCalcSize(BOOL *bCalcValidRects*, NCCALCSIZE_PARAMS* *lpncsp*);

参数

bCalcValidRects

指定应用程序是否需要指定客户区的哪个部分包含了有效信息。Windows 将把有效信息拷贝到新客户区的指定位置。如果这个参数为 TRUE，则应用程序必须指定客户区的哪个部分有效。

lpncsp

指向一个 NCCALCSIZE_PARAMS 数据结构，其中包含了应用程序可用于计算 CWnd 矩形（包括客户区、边框、标题和滚动条等）的新大小和位置的信息。

说明

当客户区的大小和位置需要重新计算时，框架调用这个成员函数。通过处理这

个消息，应用程序可以在窗口的大小和位置发生变化时控制窗口客户区的内容。

不论 bCalcValidRects 的值是什么，NCCALCSIZE_PARAMS 结构的 rgrc 结构成员所指向的数组中的第一个矩形中包含了窗口的坐标。对于一个子窗口，这个坐标是相对于父窗口的客户区的。对于顶层窗口，该坐标是屏幕坐标。应用程序应当修改 rgrc[0]矩形以反映客户区的大小和位置。

rgrc[1]和 rgrc[2]矩形仅当 bCalcValidRects 为 TRUE 时有效。在这种情况下，rgrc[1]矩形中包含了被移动或改变大小的窗口的坐标。rgrc[2]矩形中包含了窗口被移动之前的客户区坐标。所有的坐标都是相对于父窗口或屏幕的。

缺省的实现根据窗口特征计算客户区的大小（是否有滚动条，菜单等），并且将结果放入 lpncsp。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_NCCALCSIZE, CWnd::MoveWindow, CWnd::SetWindowPos

CWnd::OnNcCreate

afx_msg BOOL OnNcCreate(LPCREATESTRUCT *lpCreateStruct*);

返回值

如果创建了非客户区，则返回非零值。如果发生错误，则返回 0；在这种情况下，Create 函数将返回错误。

参数

lpCreateStruct

指向CWnd的CREATESTRUCT数据结构。

说明

当CWnd对象第一次被创建时，框架在WM_CREATE消息之前调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　CWnd::Create, CWnd::CreateEx, WM_NCCREATE

CWnd::OnNcDestroy

afx_msg void OnNcDestroy( );

说明

当非客户区即将被销毁时，框架调用这个函数，这是Windows的窗口被销毁时调用的最后一个成员函数。缺省的实现执行一些清除工作，然后调用虚成员函数PostNcDestroy。

如果你希望执行自己的清除操作，例如删除，则应重载PostNcDestroy。如果你重载了OnNcDestroy，则必须调用基类的OnNcDestroy以确保内部为窗口分配的内存都被释放。

请参阅　CWnd::DestroyWindow, CWnd::OnNcCreate, WM_NCDESTROY,

CWnd::Default, CWnd::PostNcDestroy

CWnd::OnNcHitTest

afx_msg UINT OnNcHitTest(CPoint *point*);

返回值

下面列出的鼠标击中测试枚举值之一。

参数

point

包含了光标的 x **轴和** y **轴坐标。这些坐标总是用屏幕坐标给出的。**

说明

每当鼠标移动时，框架就为包含光标（或者是用 SetCapture **成员函数捕获了鼠标输入的** CWnd **对象）的** CWnd **对象调用这个成员函数。**

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows **消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。**

请参阅　CWnd::GetCapture, WM_NCHITTEST

鼠标枚举值

- HTBORDER　**在不具有可变大小边框的窗口的边框上。**
- HTBOTTOM　**在窗口的水平边框的底部。**
- HTBOTTOMLEFT　**在窗口边框的左下角。**

- HTBOTTOMRIGHT 在窗口边框的右下角。
- HTCAPTION 在标题条中。
- HTCLIENT 在客户区中。
- HTERROR 在屏幕背景或窗口之间的分隔线上（与 HTNOWHERE 相同，除了 Windows 的 DefWndProc 函数产生一个系统响声以指明错误）。
- HTGROWBOX 在尺寸框中。
- HTHSCROLL 在水平滚动条上。
- HTLEFT 在窗口的左边框上。
- HTMAXBUTTON 在最大化按钮上。
- HTMENU 在菜单区域。
- HTMINBUTTON 在最小化按钮上。
- HTNOWHERE 在屏幕背景或窗口之间的分隔线上。
- HTREDUCE 在最小化按钮上。
- HTRIGHT 在窗口的右边框上。
- HTSIZE 在尺寸框中。（与 HTGROWBOX 相同）
- HTSYSMENU 在控制菜单或子窗口的关闭按钮上。
- HTTOP 在窗口水平边框的上方。
- HTTOPLEFT 在窗口边框的左上角。
- HTTOPRIGHT 在窗口边框的右上角。
- HTTRANSPARENT 在一个被其它窗口覆盖的窗口中。
- HTVSCROLL 在垂直滚动条中。

- HTZOOM　在最大化按钮上。

CWnd::OnNcLButtonDblClk

afx_msg void OnNcLButtonDblClk(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在 CWnd 的非客户区内双击鼠标左键时，框架调用这个成员函数。如果合适的话，将发出 WM_SYSCOMMAND 消息。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的

参数（而不是你提供给这个函数的参数）。

请参阅 WM_NCLBUTTONDBLCLK, CWnd::OnNcHitTest

CWnd::OnNcLButtonDown

afx_msg void OnNcLButtonDown(UINT *nHitTest*, CPoint *point*);

参数

nHitTest
指定了击中测试代码。击中测试用于确定光标的位置。

point
指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在 CWnd 的非客户区内按下鼠标左键时，框架调用这个成员函数。如果合适的话，将发出 WM_SYSCOMMAND 消息。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消

息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcLButtonDblClk, CWnd::OnNcLButtonUp,CWnd:: OnSysCommand, WM_NCLBUTTONDOWN, CWnd::Default

CWnd::OnNcLButtonUp

afx_msg void OnNcLButtonUp(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在 CWnd 的非客户区内放开鼠标左键时，框架调用这个成员函数。如果合适的话，将发出 WM_SYSCOMMAND 消息。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcLButtonDown, CWnd::OnSysCommand, WM_NCLBUTTONUP

CWnd::OnNcMButtonDblClk

afx_msg void OnNcMButtonDblClk(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。

这些坐标总是相对于屏幕的左上角的。

说明

当用户在 CWnd 的非客户区内双击鼠标中键时，框架调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcMButtonDown, CWnd::OnNcMButtonUp,WM- _NCMBUTTONDBLCLK

CWnd::OnNcMButtonDown

afx_msg void OnNcMButtonDown(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个CPoint对象，其中包含了光标位置的x轴和y轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在CWnd的非客户区内按下鼠标中键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcMButtonDblClk,CWnd::OnNcMButtonUp,WM_NCMBUTTONDOWN

CWnd::OnNcMButtonUp

afx_msg void OnNcMButtonUp(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个CPoint对象，其中包含了光标位置的x轴和y轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在CWnd的非客户区内放开鼠标右键时，框架调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcMButtonDblClk, CWnd::OnNcMButtonDown, WM_NCMBUTTONUP

CWnd::OnNcMouseMove

afx_msg void OnNcMouseMove(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当光标在 CWnd 的非客户区内移动时，框架调用这个成员函数。如果合适的话，将发出 WM_SYSCOMMAND 消息。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnSysCommand, WM_NCMOUSEMOVE

CWnd::OnNcPaint

```
afx_msg void OnNcPaint( );
```

说明

当非客户区需要重画时，框架调用这个成员函数。缺省的实现画出窗口的框架。应用程序可以重载这个调用并画出自己定义的窗口边框。裁剪区域通常是矩形，即使框架的形状已经改变。

请参阅 WM_NCPAINT

CWnd::OnNcRButtonDblClk

afx_msg void OnNcRButtonDblClk(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在 CWnd 的非客户区内双击鼠标右键时，框架调用这个成员函数。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcRButtonDown, CWnd::OnNcRButtonUp,WM_NCRBUTTONDBLCLK

CWnd::OnNcRButtonDown

afx_msg void OnNcRButtonDown(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个 CPoint 对象，其中包含了光标位置的 x 轴和 y 轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在 CWnd 的非客户区内按下鼠标右键时，框架调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcRButtonDblClk, CWnd::OnNcRButtonUp

CWnd::OnNcRButtonUp

afx_msg void OnNcRButtonUp(UINT *nHitTest*, CPoint *point*);

参数

nHitTest

指定了击中测试代码。击中测试用于确定光标的位置。

point

指定了一个CPoint对象，其中包含了光标位置的x轴和y轴屏幕坐标。这些坐标总是相对于屏幕的左上角的。

说明

当用户在CWnd的非客户区内双击鼠标左键时，框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, CWnd::OnNcRButtonDblClk, CWnd::OnNcRButtonDown, WM_NCRBUTTONUP

CWnd::OnNotify

virtual BOOL CWnd::OnNotify(WPARAM *wParam*, LPARAM *lParam*, LRESULT* *pResult*);

返回值

如果应用程序处理这个消息，则返回非零值；否则返回 0。

参数

wParam

如果消息是从控件发出的，则标识了发出消息的控件；否则 wParam 为 0。

lParam

指向通知消息（NMHDR）结构的指针，其中包含了通知消息代码和附加信息。对于某些通知消息，这个指针指向一个更大的结构，NMHDR 是其第一个参数。

pResult

指向 LRESULT 变量的指针，如果消息被处理，则用它来保存返回代码。

说明

框架调用这个函数以通知控件的父窗口，在控件中发生了一个事件，或者该控件需要某些类型的信息。

OnNotify 处理控件通知的消息映射。

在你的派生类中重载这个成员函数以处理 WM_NOTIFY 消息。重载函数不会处理消息映射，除非调用了基类的 OnNotify。

有关 WM_NOTIFY 消息的更多信息参见“TN061：ON_NOTIFY 和 WM_NOTIFY 消息”。可能你会对“TN060：新的 Windows 公共控件”和“TN062：Windows 控件的消息反射”中描述的相关主题感兴趣。所有这些都在 Visual C++联机文档中。

CWnd::OnPaint

```
afx_msg void OnPaint( );
```

说明

当 Windows 或应用程序请求重画应用程序窗口的一部分时，框架调用这个成员函数。WM_PAINT 在调用 UpdateWindow 或 RedrawWindow 成员函数时发出。当设置了 RDW_INTERNALPAINT 标志并调用 RedrawWindow 成员函数时，窗口可能会接收到内部重画消息。在这种情况下，窗口可能没有更新区域。应用程序必须调用 GetUpdateRect 成员函数以确定窗口是否具有更新区域。如果 GetUpdateRect 返回 0，则应用程序不应调用 BeginPaint 和 EndPaint 成员函数。

应用程序负责检查是否需要内部重画或更新，这可通过查看每条 WM_PAINT 消息的内部数据结构来完成，因为一条 WM_PAINT 可能是由于一个无效区域

或由于使用 RDW_INTERNALPAINT 标志调用了 RedrawWindow 成员函数而引起的。

Windows 只发送一次内部 WM_PAINT 消息。在通过 UpdateWindow 成员函数向窗口发送了内部 WM_PAINT 消息以后，将不会再向窗口发送其它 WM_PAINT 消息，直到再次使用 RDW_INTERNALPAINT 标志调用了 RedrawWindow 成员函数。
有关在文档/视应用程序中描绘图象的信息参见 CView::OnDraw。

有关使用 WM_PAINT 的更多信息参见《Win32 SDK 程序员参考》中的下列主题：

- “WM_PAINT 消息”
- “使用 WM_PAINT 消息”

请参阅

CWnd::BeginPaint,CWnd::EndPaint,CWnd::RedrawWindow,CPaintDC,CView::OnDraw

CWnd::OnPaintClipboard

afx_msg void OnPaintClipboard(CWnd* *pClipAppWnd*, HGLOBAL *hPaintStruct*);

参数

pClipAppWnd

指定了剪贴板应用程序的窗口指针。这个指针可能是临时的，不能被保存以供将来使用。

hPaintStruct

标识了一个 PAINTSTRUCT 数据结构，它确定了要画出客户区的什么部分。

说明

当剪贴板的拥有者以 CF_OWNERDISPLAY 格式在剪贴板中放置了数据并且剪贴板观察器的客户区需要重画时，剪贴板观察器就调用剪贴板拥有者的 OnPaintClipboard 成员函数。

要确定是整个客户区还是客户区的一部分需要重画，剪贴板拥有者必须把 PAINTSTRUCT 结构的 rcpaint 成员所指定的绘图区域的大小与最近调用的 OnSizeClipboard 成员函数中指定的大小相比较。

OnPaintClipboard 应该使用 Windows 的 GlobalLock 函数来锁定包含了 PAINTSTRUCT 数据结构的内存，并且在退出之前用 Windows 的 GlobalUnlock 函数来释放内存。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消

息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 ::GlobalLock, ::GlobalUnlock, CWnd::OnSizeClipboard, WM_PAINTCLIPBOARD

CWnd::OnPaletteChanged

afx_msg void OnPaletteChanged(CWnd* *pFocusWnd*);

参数

pFocusWnd

指定了引起系统调色板改变的窗口的指针。这个指针可能是临时的，不能被保存。

说明

当拥有输入焦点的窗口实现了它的逻辑调色板，因而改变了系统调色板时，框架为所有的顶层窗口调用这个成员函数。这个调用允许不具有输入焦点且拥有调色板的窗口实现它的逻辑调色板并更新其客户区。

OnPalletteChanged 成员函数是为所有的顶层窗口和重叠窗口，包括改变了系统调色板，因而引起 WM_PALETTECHANGED 消息的窗口而调用的。如果子窗口也使用调色板，则这个消息必须发送给它。

为了避免无限循环，除非窗口确定 pFocusWnd 中不包含指向自己的指针，否则就不应实现其调色板。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 ::RealizePalette, WM_PALETTECHANGED, CWnd::OnPaletteIsChanging, CWnd::OnQueryNewPalette

CWnd::OnPaletteIsChanging

afx_msg void OnPaletteIsChanging(CWnd* *pRealizeWnd*);

参数

pRealizeWnd

指定了要实现逻辑调色板的窗口。

说明

框架调用这个成员函数以通知应用程序，有一个应用程序即将实现它的逻辑调色板。

> **注意**　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnPaletteChanged, CWnd::OnQueryNewPalette, WM_PALETTEISCHANGING

CWnd::OnParentNotify

afx_msg void OnParentNotify(UINT *message*, LPARAM *lParam*);

参数

message

指定了父窗口要被通知的事件和子窗口的标识符。事件是消息的低位字。如果事件为 WM_CREATE 或 WM_DESTROY，则消息的高位字是子窗口的标识符；否则高位字没有定义。事件（消息的低位字）可以是下列

值中的任意一个：

- WM_CREATE　子窗口正在创建。
- WM_DESTROY　子窗口正被销毁。
- WM_LBUTTONDOWN　用户将鼠标光标放在子窗口上方并按下了鼠标的左键。
- WM_MBUTTONDOWN　用户将鼠标光标放在子窗口上方并按下了鼠标的中键。
- WM_RBUTTONDOWN　用户将鼠标光标放在子窗口上方并按下了鼠标的右键。

lParam

如果消息的事件(低位字)是 WM_CREATE 或 WM_DESTROY,则 *lParam* 指定了子窗口的窗口句柄，否则 *lParam* 包含了光标的 x 和 y 坐标。x 坐标为低位字，y 坐标为高位字。

说明

当子窗口被创建或销毁，或者用户在子窗口上方点击鼠标按钮时，框架调用父窗口的 OnParentNotify 成员函数。在子窗口被创建的时候，系统在创建窗口的 Create 成员函数返回之前调用 OnParentNotify。当子窗口被销毁的时候，系统在任何销毁窗口的处理产生之前调用 OnParentNotify。

OnParentNotify 是为子窗口的所有父类窗口，包括顶层窗口调用的。

除了那些具有WS_EX_NOPARENTNOTIFY风格的子窗口外，所有的子窗口都向它们的父窗口发送这个消息。在缺省情况下，对话框中的子窗口都具有WS_EX_NOPARENTNO- TIFY风格，除非在用CreateEx成员函数创建子窗口的时候没有使用这个风格。

注意　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　CWnd::OnCreate, CWnd::OnDestroy, CWnd::OnLButtonDown, CWnd::OnMButtonDown, CWnd::OnRButtonDown, WM_PARENTNOTIFY

CWnd::OnQueryDragIcon

```
afx_msg HCURSOR OnQueryDragIcon( );
```

返回值

一个双字值，它在低位字中包含了光标或图标的句柄。光标和图标必须与显示器的分辨率匹配。如果应用程序返回NULL，则系统将显示缺省的光标。缺省的返回值是NULL。

说明

框架为不具有为类图标的最小化（图标化）窗口调用这个成员函数。系统调用这个函数以在用户拖拉最小化窗口的时候显示光标。

如果应用程序返回图标或光标的句柄，系统将它转换为黑与白。

如果应用程序返回一个句柄，则这个句柄必须标识与显示设备分辨率相兼容的单色光标或图标。应用程序可以调用 CWinApp::LoadCursor 或 CWinApp::LoadIcon 成员函数以从它的可执行文件的资源中载入光标或图标并获得其句柄。

请参阅 CWinApp::LoadCursor, CWinApp::LoadIcon, WM_QUERYDRAGICON

CWnd::OnQueryEndSession

```
afx_msg BOOL OnQueryEndSession( );
```

返回值

如果应用程序可以被方便地关闭，则返回非零值；否则返回 0。

说明

当用户选择关闭 Windows 或者应用程序调用 Windows 的 ExitWindows 函数时，框架调用这个成员函数。如果应用程序返回 0，则 Windows 会话不会结束。只要有一个应用程序返回了 0，Windows 就停止调用 OnQueryEndSession，并且向所有已经返回非零值的应用程序发送一个 WM_ENDSESSION 消息，参数值为 FALSE。

请参阅 ::ExitWindows, CWnd::OnEndSession, WM_QUERYENDSESSION

CWnd::OnQueryNewPalette

afx_msg BOOL OnQueryNewPalette( );

返回值

如果 CWnd 实现了它的逻辑调色板，则返回非零值；否则为 0。

说明

当 CWnd 对象即将接收输入焦点时，框架就调用这个成员函数，使 CWnd 有机会在接收到输入焦点时实现它的逻辑调色板。

请参阅 CWnd::Default, CWnd::OnPaletteChanged, WM_QUERYNEWPALETTE

CWnd::OnQueryOpen

afx_msg BOOL OnQueryOpen( );

返回值

如果图标可以被打开，则返回非零值；如果要防止图标被打开，则返回0。

说明

当CWnd是最小化的，并且用户请求CWnd恢复它在被最小化之前的大小和位置时，框架调用这个成员函数。

在OnQueryOpen中，CWnd不应该执行任何可能引起激活或改变焦点的动作(例如创建对话框）。

请参阅 WM_QUERYOPEN

CWnd::OnRButtonDblClk

afx_msg void OnRButtonDblClk(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL　如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON　如果鼠标左键被按下，则设置此位。
- MK_MBUTTON　如果鼠标中键被按下，则设置此位。
- MK_RBUTTON　如果鼠标右键被按下，则设置此位。
- MK_SHIFT　如果 SHIFT 键被按下，则设置此位。

point

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户双击鼠标右键时框架调用这个成员函数。

只有具有 CS_DBLCLKS WNDCLASS 风格的窗口才接收 OnRButtonDlbClk 调用。这是微软基础类窗口的缺省状态。当用户按下、释放，然后在系统规定的双击时间限制之内再次按下鼠标右键时，Windows 就调用 OnRButtonDblClk。双击鼠标右键实际产生四个事件：WM_RBUTTONDOWN，WM_RBUTTONUP 消息，WM_RBUTTONDBLCLK 调用，以及释放按钮时的另一个 WM_RBUTTONUP 消息。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnRButtonDown, CWnd::OnRButtonUp, WM_RBUTTONDBLCLK

CWnd::OnRButtonDown

afx_msg void OnRButtonDown(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果 CTRL 键被按下，则设置此位。
- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果 SHIFT 键被按下，则设置此位。

point

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户按下鼠标右键时框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnRButtonDblClk, CWnd::OnRButtonUp, WM_RBUTTONDOWN

CWnd::OnRButtonUp

afx_msg void OnRButtonUp(UINT *nFlags*, CPoint *point*);

参数

nFlags

指定了不同的虚拟键是否被按下。这个参数可以是下列值之一：

- MK_CONTROL 如果 CTRL 键被按下，则设置此位。

- MK_LBUTTON 如果鼠标左键被按下，则设置此位。
- MK_MBUTTON 如果鼠标中键被按下，则设置此位。
- MK_RBUTTON 如果鼠标右键被按下，则设置此位。
- MK_SHIFT 如果 SHIFT 键被按下，则设置此位。

point

指定了光标的 x 和 y 轴坐标。这些坐标通常是相对于窗口的左上角的。

说明

当用户放开鼠标右键时框架调用这个成员函数。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnRButtonDblClk, CWnd::OnRButtonDown, WM_RBUTTONUP

CWnd::OnRegisteredMouseWheel

afx_msg LRESULT OnRegisteredMouseWheel(WPARAM *wParam*, LPARAM *lParam*);

返回值

此时不重要。总为零。

参数

wParam
　　指针的水平位置。

lParam
　　指针的垂直位置。

说明

当用户滚动鼠标滚轮并遇到滚轮的下一个刻度时，框架就调用这个成员函数。除非被重载，否则 OnRegisteredMouseWheel 注册 Windows 的消息，将消息转发到适当的窗口，然后为该窗口调用 WM_MOUSEWHEEL 处理函数。

重载这个成员函数以提供你自己的消息转发机制或改变鼠标滚轮的滚动特性。

> **注意** OnRegisteredMouseWheel 处理 Windows 95 和 Windows NT 3.51 中的消息。对于 Windows NT 4.0 的消息处理，则使用 OnMouseWheel。

请参阅 RegisterWindowMessage

CWnd::OnRenderAllFormats

afx_msg void OnRenderAllFormats( );

说明

当剪贴板的拥有者应用程序即将被销毁时，框架就调用拥有者的 OnRenderAllFormats **成员函数。**

剪贴板的拥有者应当提交它能够产生的的所有格式的数据，并调用 Windows **的** SetClipboardData **函数将每种格式的数据句柄传递给剪贴板。这确保剪贴板中包含了有效的数据，即使提交数据的应用程序已经被销毁了。应用程序必须在调用** Windows **的** SetClipboardData **函数之前调用** OpenClipboard **成员函数，然后调用** Windows **函数** CloseClipboard**。**

请参阅 ::CloseClipboard, CWnd::OpenClipboard, ::SetClipboardData, CWnd::OnRenderFormat, WM_RENDERALLFORMATS

CWnd::OnRenderFormat

afx_msg void OnRenderFormat(UINT *nFormat*);

参数

nFormat

指定了剪贴板的格式。

说明

当一种延迟提交的特定格式需要提交时，框架就调用剪贴板拥有者的 OnRenderFormat **成员函数。接收者应当按照该格式提交数据，并调用** Windows **函数** SetClipboard **将它传递到剪贴板。**

不要在 OnRenderFormat **之内调用** OpenClipboard **成员函数或者** Windows **函数** CloseClipboard。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 ::CloseClipboard, CWnd::OpenClipboard, ::SetClipboardData, WM_RENDERFORMAT

CWnd::OnSetCursor

afx_msg BOOL OnSetCursor(CWnd* *pWnd*, UINT *nHitTest*, UINT *message*);

返回值

如果要停止进一步处理，则返回非零值；如果要继续，则返回 0**。**

参数

pWnd

指定了包含光标的窗口指针。这个指针可能是临时的，不能被保存以供将来使用。

nHitTest

指定了击中测试区域代码。击中测试确定了光标的位置。

message

指定了鼠标消息。

说明

如果鼠标输入没有被捕获并且鼠标使光标在 CWnd **对象内移动，则框架调用这个成员函数。**

缺省的实现在处理之前调用父窗口的OnSetCursor。如果父窗口返回TRUE，则将停止进一步处理。调用父窗口使父窗口能够控制子窗口中光标的设置。

如果光标不在客户区内，缺省的实现将光标设为箭头；如果是在客户区内，则将光标设为注册的类光标。

如果nHitTest为HTERROR并且该消息是一个鼠标键按下消息，则将调用MessageBeep成员函数。

当CWnd进入菜单模式时，消息参数为0。

注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnNcHitTest, WM_SETCURSOR

CWnd::OnSetFocus

afx_msg void OnSetFocus(CWnd* *pOldWnd*);

参数

pOldWnd

包含了失去输入焦点的 CWnd 对象（可能为 NULL）。这个指针可能是临时的，不能被保存以供将来使用。

说明

框架在获得输入焦点以后调用这个成员函数。如果要显示插字符，CWnd 必须在此时调用适当的插字符函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　WM_SETFOCUS

CWnd::OnShowWindow

afx_msg void OnShowWindow(BOOL *bShow*, UINT *nStatus*);

参数

bShow

指定窗口是否要被显示。如果窗口要被显示，则为 TRUE；如果窗口要

被隐藏，则为 FALSE。

nStatus

指定了要显示的窗口的状态。如果是因为调用 ShowWindow 成员函数而发出的消息，则为 0；否则 *nStatus* 为下列值之一：

- SW_PARENTCLOSING 父窗口正被关闭（变为图标）或弹出式窗口正被隐藏。
- SW_PARENTOPENING 父窗口正被打开（被显示）或弹出式窗口正被显示。

说明

当 CWnd 对象要被显示或隐藏时，框架调用这个成员函数。当调用 ShowWindow 成员函数时，或者重叠窗口被最大化或复原，或者重叠式或弹出式窗口被关闭（变为图标）或打开（被显示）时，窗口被显示或隐藏。当重叠窗口被关闭时，所有的与此窗口相关的所有弹出窗口都被隐藏。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_SHOWWINDOW

CWnd::OnSize

afx_msg void OnSize(UINT *nType*, int *cx*, int *cy*);

参数

nType

指定了要求的调整大小的类型。这个参数可以是下列值之一：

- SIZE_MAXIMIZED **窗口已经被最大化。**
- SIZE_MINIMIZED **窗口已经被最小化。**
- SIZE_RESTORED **窗口被改变了大小，但** SIZE_MINIMIZED **和** SIZE_ MAXIMIZED **都不适用。**
- SIZE_MAXHIDE **当其它窗口被最大化时，消息被发送到所有的弹出窗口。**
- SIZE_MAXSHOW **当其它窗口被恢复到原来的大小时，消息被发送到所有的弹出窗口。**

cx

指定了客户区域的新宽度。

cy

指定了客户区域的新高度。

说明

框架在窗口的大小被改变以后调用这个成员函数。

如果在 OnSize 中为子窗口调用了 SetScrollPos 或 MoveWindow 成员函数，则 SetScrollPos 或 MoveWindow 的 bRedraw 参数必须为非零值，以使 CWnd 能被重画。

> 注意　框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　CWnd::MoveWindow, CWnd::SetScrollPos, WM_SIZE

CWnd::OnSizeClipboard

afx_msg void OnSizeClipboard(CWnd* *pClipAppWnd*, HGLOBAL *hRect*);

参数

pClipAppWnd
　　标识一个剪贴板应用程序的窗口。这个指针可能是临时的，不能被保存。

hRect

标识一个全局内存对象。这个内存对象中包含了一个RECT数据结构，指定了剪贴板拥有者要画出的区域。

说明

当剪贴板中包含了具有CF_OWNERDISPLAY属性的数据并且剪贴板观察器窗口的客户区发生了改变时，剪贴板观察器就调用剪贴板拥有者的OnSizeClipboard成员函数。

当剪贴板应用程序要被销毁或最小化时，OnSizeClipboard成员函数将会被调用，并且以一个空矩形（0，0，0，0）作为新的大小。这使得剪贴板拥有者可释放它的显示资源。

在OnSizeClipboard中，应用程序必须用Windows函数GlobalLock来锁定包含了RECT数据结构的内存。并且在返回控制权之前用Windows函数GlobalUnlock释放内存。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅

::GlobalLock, ::GlobalUnlock, ::SetClipboardData,CWnd::SetClipboardViewer,WM_ SIZECLIPBOARD

CWnd::OnSizing

afx_msg void OnSizing(UINT *nSide*, LPRECT *lpRect*);

参数

nSide

要移动的窗口的边界。

lpRect

CRect **或** RECT **结构的地址，将包含项的坐标。**

说明

框架调用这个成员函数以指明用户正在改变矩形的大小。通过处理这个消息，应用程序可以监控拖动矩形的大小和位置，并且如果需要，可以改变它的大小和位置。

> **注意** **框架调用这个成员函数以允许你的应用程序处理一个** Windows **消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。**

如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

CWnd::OnSpoolerStatus

afx_msg void OnSpoolerStatus(UINT *nStatus*, UINT *nJobs*);

参数

nStatus

指定了 SP_JOBSTATUS 标志。

nJobs

指定了打印管理器队列中的作业的数目。

说明

每当在打印管理器中加入或删除一个打印作业时，框架就调用这个成员函数。这个调用仅用于通知目的。

注意 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的

参数（而不是你提供给这个函数的参数）。

请参阅 WM_SPOOLERSTATUS

CWnd::OnStyleChanged

afx_msg void OnStyleChanged(int *nStyleType*, LPSTYLESTRUCT *lpStyleStruct*);

参数

nStyleType

指定改变的是窗口的扩展风格还是非扩展风格。这个参数可以是下列值之一：

- GWL_EXSTYLE 窗口的扩展风格发生改变。
- GWL_STYLE 窗口的非扩展风格发生改变。

lpStyleStruct

指向 STYLESTRUCT 结构，其中包含了窗口的新风格。应用程序可以检查这个风格，但是不能改变它们。

说明

框架在::SetWindowLong 函数改变了窗口的一个或多个风格之后调用这个成员

函数。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_STYLECHANGED

CWnd::OnStyleChanging

afx_msg void OnStyleChanging(int *nStyleType*, LPSTYLESTRUCT *lpStyleStruct*);

参数

nStyleType

指定改变的是窗口的扩展风格还是非扩展风格。这个参数可以是下列值之一：

- GWL_EXSTYLE 窗口的扩展风格发生改变。
- GWL_STYLE 窗口的非扩展风格发生改变。

lpStyleStruct

指向一个STYLESTRUCT结构，其中包含了窗口的新风格。应用程序可

以检查这些风格并改变它们。

说明

框架在::SetWindowLong将要改变窗口的一个或多个风格时调用这个成员函数。

注意　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

CWnd::OnSysChar

afx_msg void OnSysChar(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar
　　指定了控制菜单键的ASCII字符码。

nRepCnt
　　指定了重复计数（当用户按住键时产生的重复击键次数）。

nFlags

nFlags参数可以具有这些值：

值	含义
0—15	指定了重复计数。该值为用户按住键时产生的重复击键次数
16—23	指定了扫描码。其值依赖于原始设备制造商（OEM）
24	指定了该键是否是扩展键，如增强 101 或 102 键的键盘上右手的 ALT 和 CTRL 键。如果它是扩展键，则其值为 1；否则为 0
25—28	Windows内部使用
29	指定了上下文代码。如果当按下键时 ALT 键是按下的，则其值为 1；否则，其值为 0
30	指定了原来的键状态。如果在消息发出之前键是按下的，则其值为 1；如果是弹起的，则为 0
31	指定了键的暂态。如果键正被释放，则其值为 1，如果键正被按下，则为 0

说明

如果 CWnd 拥有输入焦点，并且转换出 WM_SYSKEYUP 和 WM_SYSKEYDOWN 消息，则框架调用这个成员函数。它指定了控制菜单键的虚拟键码。

如果上下文代码为 0，则可以将 WM_SYSCHAR 消息传递给 Windows 的

TranslateAccelerator函数，它将处理这个消息，就好像这个消息是一个普通按键消息而不是系统字符键。这就使得加速键可以在活动窗口中使用，即使活动窗口不具有输入焦点。

对于IBM增强101和102键键盘，增强键包括键盘主体部分的右ALT键和右CTRL键；数字键盘左侧的INS，DEL，HOME，END，PAGE UP，PAGE DOWN和箭头键；以及数字键盘上的斜杠（/）和ENTER键。一些其它的键盘可能支持nFlags中的扩展键位。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 ::TranslateAccelerator, WM_SYSKEYDOWN, WM_SYSKEYUP, WM_SYSCHAR

CWnd::OnSysColorChange

afx_msg void OnSysColorChange( );

说明

当系统颜色设置发生变化时，框架为所有的顶层窗口调用这个成员函数。

Windows 为系统颜色变化所影响的任何窗口调用 OnSysColorChange。

具有使用现有的系统颜色的刷子的应用程序必须删除这些刷子，并用新的系统颜色重新创建这些刷子。

请参阅 ::SetSysColors, WM_SYSCOLORCHANGE

CWnd::OnSysCommand

afx_msg void OnSysCommand(UINT *nID*, LPARAM *lParam*);

参数

nID

指定了请求的系统命令的类型。这个参数可以是下列值之一：

- SC_CLOSE 关闭 CWnd 对象。
- SC_HOTKEY 激活与应用程序指定的热键相关的 CWnd 对象。lParam 的低位字标识了要激活的窗口的 HWND 句柄。
- SC_HSCROLL 水平滚动。
- SC_KEYMENU 通过击键获得菜单。
- SC_MAXIMIZE (或 SC_ZOOM) 最大化 CWnd 对象。
- SC_MINIMIZE (或 SC_ICON) 最小化 CWnd 对象。
- SC_MOUSEMENU 通过鼠标点击获得菜单。

- SC_MOVE 移动 CWnd 对象。
- SC_NEXTWINDOW 移动到下一个窗口。
- SC_PREVWINDOW 移动到前一个窗口。
- SC_RESTORE 将窗口恢复为普通的位置和大小。
- SC_SCREENSAVE 执行 SYSTEM.INI 文件中[boot]部分指定的屏幕保护应用程序。
- SC_SIZE 调整 CWnd 对象的大小。
- SC_TASKLIST 执行或激活 Windows 的任务管理器应用程序。
- SC_VSCROLL 垂直滚动。

lParam

如果控制菜单是通过鼠标选择的，则 *lParam* 中包含了光标的位置。低位字包含了 x 轴坐标，高位字包含了 y 轴坐标；否则这个参数没有使用。

- SC_HOTKEY 激活与应用程序指定的热键相关的窗口。lParam 的低位字标识了要激活的窗口。
- SC_SCREENSAVE 在控制面板的桌面部分执行屏幕保护应用程序。

说明

当用户从控制菜单选择了一个命令，或者用户选择了最大化户最小化按钮时，框架调用这个函数。

在缺省情况下，OnSysCommand 执行控制菜单对前面表格中描述的预定义动作

的请求。

在WM_SYSCOMMAND消息中，nID参数的低四位被Windows内部使用。当应用程序测试nID的值时，它必须用位与操作符AND将值0xFFF0与nID的值组合在一起以获得正确的结果。

控制菜单中的菜单项可以用GetSystemMenu，AppendMenu，InsertMenu和ModifyMenu成员函数来修改。修改了控制菜单的应用程序必须处理WM_SYSCOMMAND消息，并且应用程序没有处理的任何WM_SYSCOMMAND消息都必须被发送给OnSysCommand。应用程序加入的任何命令值必须由应用程序处理，并且不能被传递给OnSysCommand。

应用程序可以在任何时候通过向OnSysCommand发送WM_SYSCOMMAND消息来执行任何系统命令。
为选择控制菜单中的项的加速键（快捷方式）被转换为OnSysCommand调用；所有其它的加速键被转换为WM_COMMAND消息。

> 注意 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 WM_SYSCOMMAND

CWnd::OnSysDeadChar

afx_msg void OnSysDeadChar(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar

指定了死键的字符值。

nRepCnt

指定了重复计数。

nFlags

指定了扫描码、键暂态、一起的键状态和上下文代码，如下面的列表所示：

值	含义
0—7	扫描码（依赖于OEM）。高位字的低位字节
8	扩展键，如功能键或数字键盘上的键（如果该键为扩展键，则返回1；否则返回0）
9—10	没有使用
11—12	Windows内部使用
13	上下文代码（如果键被按下时ALT键是被按住的，则为1；否则为0）

续表

14	以前的键状态（如果键在调用前被按下，则为1；否则为0）
15	暂态（如果键是要被放开，则为1；如果键是要被按下，则为0）

说明

如果当OnSysKeyUp或OnSysKeyDown成员函数被调用时CWnd对象拥有输入焦点，则框架调用这个成员函数。它指定了死键的字符值。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 CWnd::OnSysKeyDown, CWnd::OnSysKeyUp, WM_SYSDEADCHAR, CWnd::OnDeadChar

CWnd::OnSysKeyDown

afx_msg void OnSysKeyDown(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar

指定了被按下的键的虚拟键码。

nRepCnt

指定了重复计数。

nFlags

指定了扫描码、键暂态、以前的键状态和上下文代码，如下面的列表所示：

值	含义
0–7	扫描码（依赖于OEM）。高位字的低位字节
8	扩展键，如功能键或数字键盘上的键（如果该键为扩展键，则返回1；否则返回0）
9–10	没有使用
11–12	Windows内部使用
13	上下文代码（如果键被按下时ALT键是被按住的，则为1；否则为0）
14	以前的键状态（如果键在调用前被按下，则为1；否则为0）
15	暂态（如果键是要被放开，则为1；如果键是要被按下，则为0）

对于OnSysKeyDown调用，键暂态位（15位）为0。如果按下键时ALT键被按下，则上下文代码位（13位）为1；如果因为没有窗口拥有输入焦点消息被

发送到活动窗口时，它为 0。

说明

如果 CWnd 对象具有输入焦点，则当用户按住 ALT 键然后按下其它键时，框架就调用 OnSysKeyDown 成员函数。如果没有窗口具有当前的输入焦点，活动窗口的 OnSysKeyDown 成员函数将会被调用。接收调用的 CWnd 对象可以通过检查 nFlags 中的上下文代码位来区别这两种环境。

当上下文代码为 0 时，OnSysKeyDown 接收到的 WM_SYSKEYDOWN 消息可以被传递给 Windows 的函数 TranslateAccelerator，它将处理这个消息，就像这是一种普通的键盘消息而不是系统键消息。

由于自动重复，在接收到 WM_SYSKEYUP 消息之前，可能会产生多个 OnSysKeyDown 调用。以前的键状态（14 位）可以用来确定 OnSysKeyDown 调用是代表第一个按下暂态还是重复的按下暂态。

对于 IBM 增强 101 和 102 键键盘，增强键包括键盘主体部分的右 ALT 键和右 CTRL 键；数字键盘左侧的 INS，DEL，HOME，END，PAGE UP，PAGE DOWN 和箭头键；以及数字键盘上的斜杠（/）和 ENTER 键。一些其它的键盘可能支持 nFlags 中的扩展键位。

> **注意** 框架调用这个成员函数以允许你的应用程序处理一个 Windows 消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。

如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅 ::TranslateAccelerator, WM_SYSKEYUP, WM_SYSKEYDOWN

CWnd::OnSysKeyUp

afx_msg void OnSysKeyUp(UINT *nChar*, UINT *nRepCnt*, UINT *nFlags*);

参数

nChar

指定了被释放的键的虚拟键码。

nRepCnt

指定了重复计数。

nFlags

指定了扫描码、键暂态码、以前的键状态和上下文代码，如下面的列表所示：

值	含义
0–7	扫描码（依赖于OEM）。高位字的低位字节
8	扩展键，如功能键或数字键盘上的键（如果该键为扩展键，则返回1；否则返回0）
9–10	没有使用
11–12	Windows内部使用
13	上下文代码（如果键被按下时ALT键是被按住的，则为1；否则为0）
14	以前的键状态（如果键在调用前被按下，则为1；否则为0）
15	暂态（如果键正被放开，则为1；如果键正被按下，则为0）

对于OnSysKeyUp调用，键暂态位（15位）为0。如果按下键时ALT键被按下，则上下文代码位（13位）为1；如果因为没有窗口拥有输入焦点消息被发送到活动窗口时，它为0。

说明

如果CWnd对象具有输入焦点，则当用户放开一个在按住ALT键时按下的键时，框架就调用OnSysKeyUp成员函数。如果没有窗口具有当前的输入焦点，活动窗口的OnSysKeyUp成员函数将会被调用。接收调用的CWnd对象可以通过检查nFlags中的上下文代码位来区别这两种环境。

当上下文代码为0时，OnSysKeyUp接收到的WM_SYSKEYUP消息可以被传

递给Windows的函数TranslateAccelerator，它将处理这个消息，就像这是一种普通的键盘消息，而不是系统键消息。

对于IBM增强101和102键键盘，增强键包括键盘主体部分的右ALT键和右CTRL键；数字键盘左侧的INS，DEL，HOME，END，PAGE UP，PAGE DOWN和箭头键；以及数字键盘上的斜杠（/）和ENTER键。一些其它的键盘可能支持nFlags中的扩展键位。

对于非美国增强型102键键盘，右边的ALT键被当作CTRL+ALT键组合处理。下面显示用户按下并释放这个键时生成的消息和调用的顺序：

顺序	访问的函数	传递的消息
1.	WM_KEYDOWN	VK_CONTROL
2.	WM_KEYDOWN	VK_MENU
3.	WM_KEYUP	VK_CONTROL
4.	WM_SYSKEYUP	VK_MENU

注意　框架调用这个成员函数以允许你的应用程序处理一个Windows消息。传递给你的成员函数的参数反映了接收到消息时框架接收到的参数。如果你调用了这个函数的基类实现，则该实现将使用最初传递给消息的参数（而不是你提供给这个函数的参数）。

请参阅　::TranslateAccelerator, WM_SYSKEYDOWN, WM_SYSKEYUP

CWnd::OnTCard

afx_msg void OnTCard(UINT *idAction*, DWORD *dwActionData*);

参数

idAction

表明用户所进行的动作。此参数可以是下列值之一：

- IDABORT 用户点击了可选的 Abort 按钮。
- IDCANCEL 用户点击了可选的 Cancel 按钮。
- IDCLOSE 用户关闭了训练卡片。
- IDHELP 用户点击了可选的 Windows Help 按钮。
- IDIGNORE 用户点击了可选的 Ignore 按钮。
- IDOK 用户点击了可选的 OK 按钮。
- IDNO 用户点击了可选的 No 按钮。
- IDRETRY 用户点击了可选的 Retry 按钮。
- HELP_TCARD_DATA 用户点击了一个可选的按钮。*dwActionData* 参数包含了一个由帮助编写者指定的长整数。
- HELP_TCARD_NEXT 用户点击了可选的 Next 按钮。
- HELP_TCARD_OTHER_CALLER 另一个应用程序请求训练卡片。
- IDYES 用户点击了可选的 Yes 按钮。

dwActionData

如果 *idAction* 指定的是 HELP_TCARD_DATA，则此参数是由帮助编写者指定的长整数；否则这个参数是零。

说明

当用户点击一个可选的按钮时，框架调用这个成员函数。只有当用户单击了Windows Help的某个训练卡片时，这个函数才会被调用。应用程序通过在一个对WinHelp函数的调用中指定HELP_TCARD命令来初始化一个训练卡片。

请参阅 ::WinHelp, CWinApp::WinHelp

CWnd::OnTimeChange

afx_msg void OnTimeChange( );

说明

在系统时间被改变之后，框架调用此成员函数。

任何应用程序在改变系统时间之后都向所有的顶层窗口发送这个消息。要向所有的顶层窗口发送 WM_TIMECHANGE **消息，应用程序可以使用** Windows **函数** SendMessage**，将它的** hwnd **参数设置为** HWND_BROADCAST**。**

请参阅 ::SendMessage, WM_TIMECHANGE

CWnd::OnTimer

afx_msg void OnTimer(UINT *nIDEvent*);

参数

nIDEvent

指定定时器的标识符。

说明

当在 SetTimer 成员函数中指定的每一个时间间隔都被用来安装一个定时器之后，框架调用这个成员函数。

当在应用程序的消息队列中没有其它的消息时，Windows 函数 DispatchMessage 发送一个 WM_TIMER 消息。

> 注意　框架调用这个成员函数来使你的应用程序可以处理一个 Windows 消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅　CWnd::SetTimer, WM_TIMER

CWnd::OnToolHitTest

virtual int CWnd::OnToolHitTest(CPoint *point*, TOOLINFO* *pTI*) const;

返回值

如果发现了工具提示控制，则返回 1；如果没有发现工具提示控制，则返回 -1。

参数

point

指定光标的 x-和 y-坐标。这些坐标总是相对于窗口的左上角的。

pTI

是一个指向 TOOLINFO 结构的指针。下面的结构值是缺省设置的：

- hwnd = m_hWnd 一个窗口的句柄。
- uId = (UINT)hWndChild 一个子窗口的句柄。
- uFlags |= TTF_IDISHWND 工具的句柄。
- lpszText = LPSTR_TEXTCALLBACK 指向要被显示在指定窗口中的字符串的指针。

说明

框架调用这个成员函数来确定一个点是否在指定工具的边界矩形之内。如果该点是在此矩形之内，则此函数返回有关这个工具的信息。

如果与工具提示关联的该区域不是一个按钮，则 OnToolHitTest 将结构标志设置为 TTF_NOTBUTTON 和 TTF_CENTERTIP。

重载 OnToolHitTest 可以提供与缺省不同的信息。

参见《Win32 SDK 程序员参考》中的“TOOLINFO”可以获得有关此结构的更多信息。

请参阅 TOOLINFO, CWnd::FilterTooltipMessage

CWnd::OnVKeyToItem

protafx_msg int OnVKeyToItem(UINT *nKey*, CListBox* *pListBox*, UINT *nIndex*);

返回值

返回值指定了在对消息作出响应时应用程序所执行的动作。返回值为–2 表示应用程序处理了所选项的各个方面，并且不需要列表框执行更进一步的动作。返回值为–1 表示必须执行缺省的动作来响应按键。返回值为 0 或更大表示列表框中的一个项从零基索引，并且表明了该列表框必须对给定项执行缺省的动作来响应按键。

参数

nKey

指定用户按下的键的虚键代码。

pListBox

指定一个指向此列表框的指针。该指针可能是临时的，且不得为将来使用而保存该指针。

nIndex

指出插字符的当前位置。

说明

如果CWnd对象拥有一个具有LBS_WANTKEYBOARDINPUT风格的列表框，则该列表框将发送WM_VKEYTOITEM消息来响应一个WM_KEYDOWN消息。

只有对具有LBS_HASSTRINGS风格的列表框框架才调用这个成员函数。

注意　框架调用这个成员函数来使你的应用程序可以处理一个Windows消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅　WM_KEYDOWN, WM_VKEYTOITEM

CWnd::OnVScroll

afx_msg void OnVScroll(UINT *nSBCode*, UINT *nPos*, CScrollBar* *pScrollBar*);

参数

nSBCode

指定一个指示用户的滚动请求的滚动条代码。这个参数可以是下列值之一：

- SB_BOTTOM 滚动到底部。
- SB_ENDSCROLL 结束滚动。
- SB_LINEDOWN 向下滚动一行。
- SB_LINEUP 向上滚动一行。
- SB_PAGEDOWN 向下滚动一页。
- SB_PAGEUP 向上滚动一页。
- SB_THUMBPOSITION 滚动到一个绝对位置。当前位置在 *nPos* 中指定。
- SB_THUMBTRACK 拖动滚动框到指定位置。当前位置在 *nPos* 中指定。
- SB_TOP 滚动到顶部。

nPos

如果滚动条代码是 SB_THUMBPOSITION 或 SB_THUMBTRACK，则此参数指定滚动框的位置；否则不使用此参数。根据初始的滚动范围，*nPos* 可能会是负值，如果需要的话可将其强制转换为 int 值。

pScrollBar

如果滚动信息来自于一个滚动条控制，则此参数是指向该控制的指针。如果用户单击了一个窗口的滚动条，则此参数是 NULL。该指针可能是

临时的，不能被保存为给将来使用。

说明

当用户单击窗口的垂直滚动条时，框架调用此成员函数。

希望滚动框被拖动时给出一定反馈的应用程序通常会使用 OnVScroll 函数。

如果 OnVScroll 滚动此 CWnd 对象的内容，则必须调用 SetScrollPos 成员函数来恢复滚动条的位置。

注意 框架调用这个成员函数来使你的应用程序可以处理一个 Windows 消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅 CWnd::SetScrollPos, CWnd::OnHScroll, WM_VSCROLL

CWnd::OnVScrollClipboard

afx_msg void OnVScrollClipboard(CWnd* *pClipAppWnd*, UINT *nSBCode*, UINT *nPos*);

参数

pClipAppWnd

指定了一个指向 Clipboard-viewer 窗口的指针。该指针可能是临时的，不能被保存来给将来使用。

nSBCode

指定下列滚动条值之一：

- SB_BOTTOM 滚动到底部。
- SB_ENDSCROLL 结束滚动。
- SB_LINEDOWN 向下滚动一行。
- SB_LINEUP 向上滚动一行。
- SB_PAGEDOWN 向下滚动一页。
- SB_PAGEUP 向上滚动一页。
- SB_THUMBPOSITION 滚动到一个绝对位置。当前位置在 *nPos* 中指定。
- SB_THUMBTRACK 拖动滚动框到指定位置。当前位置在 *nPos* 中指定。
- SB_TOP 滚动到顶部。

nPos

如果滚动条代码是 SB_THUMBPOSITION，则此参数包含了滚动框的位

置；否则不使用 *nPos*。

说明

当剪贴板数据具有 CF_OWNERDISPLAY 格式，并且在剪贴板查看者的垂直滚动条中发生了一个事件时，剪贴板查看者调用该剪贴板拥有者的 OnVScrollClipboard 成员函数。该拥有者应该滚动剪贴板图象使适当的区域无效，并更新滚动条的值。

> **注意** 框架调用这个成员函数来使你的应用程序可以处理一个 Windows 消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅 CWnd::Invalidate, CWnd::OnHScrollClipboard, CWnd::InvalidateRect, WM_VSCROLLCLIPBOARD, CWnd::Default

CWnd::OnWindowPosChanged

afx_msg void OnWindowPosChanged(WINDOWPOS* *lpwndpos*);

参数

lpwndpos

指向一个 WINDOWPOS 数据结构，该结构包含了有关此窗口的新尺寸和位置的信息。

说明

当调用 SetWindowPos 成员函数或其它窗口管理函数改变了窗口的尺寸、位置或 Z 次序时，框架调用此成员函数。

此函数的缺省实现给窗口发送 WM_SIZE 和 WM_MOVE。如果应用程序在处理 OnWindowPosChanged 调用时没有调用它的基类，则不发送这些消息。在 OnWindowPosChanged 调用中不调用它的基类版来完成一定或修改尺寸的处理将是更加有效的。

> 注意　框架这个成员函数来使你的应用程序可以处理一个 Windows 消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅　WM_WINDOWPOSCHANGED

CWnd::OnWindowPosChanging

afx_msg void OnWindowPosChanging(WINDOWPOS* *lpwndpos*);

参数

lpwndpos

指向一个 WINDOWPOS 数据结构，其中包含了有关窗口新的大小和位置的信息。

说明

当由于调用了 SetWindowPos 成员函数或其它的窗口管理函数，窗口的大小、位置或 Z 轴次序将要发生变化时，框架就调用这个成员函数。

应用程序可以设置或清除 WINDOWPOS 结构中 flags 成员的适当的位以便防止窗口发生变化。

对于具有 WS_OVERLAPPED 或 WS_THICKFRAME 风格的窗口，缺省的实现向窗口发送一个 WM_GETMINMAXINFO 消息。这被用来使窗口的新大小和位置有效，并强制使用 CS_BYTEALIGNCLIENT 和 CS_BYTEALIGN 客户风格。应用程序可以不调用基类而替换这个功能。

注意 框架这个成员函数来使你的应用程序可以处理一个 Windows 消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅 CWnd::OnWindowPosChanged, WM_WINDOWPOSCHANGING

CWnd::OnWinIniChange

afx_msg void OnWinIniChange(LPCTSTR *lpszSection*);

参数

lpszSection

　　指向一个字符串，该字符串指定了已改变部分的名称（该字符串不包括围绕该部分名称的方括号）。

说明

在 Windows **初始化文件** WIN.INI **发生了一个改变后，框架调用此成员函数。** Windows **函数** SystemParametersInfo **在应用程序使用此函数修改了** WIN.INI **文件中的一个设置之后调用** OnWinIniChange。

要向所有的顶层窗口发送 WM_WININICHANGE **消息，应用程序可以通过将其** hwnd **参数设置为** HWND_BROADCAST **来使用** Windows **函数** SendMessage。

如果一个应用程序同时改变了 WIN.INI **中的不同部分，应用程序应该将** *lpszSection* **设置为** NULL，**发送一个** WM_WININICHANGE **消息；否则应用程序在每次对** WIN.INE **作一个改变时就应该发送一个** WM_WININICHANGE **消息。**

如果应用程序接收到对OnWinIniChange的调用，并且*lpszSection*被设为NULL，则应用程序将检查WIN.INI中影响这个应用程序的所有部分。

> **注意** 框架这个成员函数来使你的应用程序可以处理一个Windows消息。传递给你的函数的参数反映了在消息收到时框架收到的参数。如果你调用了这个函数的基类实现，这个实现将使用最初随着消息传递过来的参数，而不是使用你提供给函数的参数。

请参阅 ::SendMessage, ::SystemParametersInfo, WM_WININICHANGE

CWnd::OnWndMsg

virtual BOOL OnWndMsg(UINT *message*, WPARAM *wParam*, LPARAM *lParam*, LRESULT* *pResult*);

返回值

如果这个消息被处理，则返回非零值；否则返回0。

参数

message

指定了被发送的消息。

wParam

指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

pResult

WindowProc 的返回值。与消息有关，可能是 NULL。

说明

这个成员函数是由 WindowProc 调用的，或者是在消息反射时调用。

有关消息反射的更多信息参见联机的《Visual C++ 程序员指南》中的“处理反射的消息”。

请参阅 CWnd::OnChildNotify, CWnd::SendChildNotifyLastMsg, CWnd::ReflectChildNotify, CCmdTarget::OnCmdMsg, CWnd::ReflectLastMsg

CWnd::OpenClipboard

BOOL OpenClipboard( );

返回值

如果通过 CWnd 打开了剪贴板，则返回非零值；如果其它应用程序或窗口已经打开了剪贴板，则返回 0。

说明

打开剪贴板。在调用 Windows 的 CloseClipboard 函数之前，其它应用程序将不能修改剪贴板的内容。

在调用 Windows 的 EmptyClipboard 函数之前，当前的 CWnd 对象将不会称为剪贴板的拥有者。

请参阅 ::CloseClipboard, ::EmptyClipboard, ::OpenClipboard

CWnd::PostMessage

BOOL PostMessage(UINT *message*, WPARAM *wParam* = 0, LPARAM *lParam* = 0);

返回值

如果公布了消息，则返回非零值；否则返回 0。

参数

message

指定了要公布的消息。

wParam

指定了附加的消息信息。这个参数的内容依赖于要公布的消息。

lParam

指定了附加的消息信息。这个参数的内容依赖于要公布的消息。

说明

这个函数将一个消息放入窗口的消息队列，然后直接返回，并不等待对应的窗口处理消息。消息队列中的消息是通过调用 Windows 的 GetMessage 或 PeekMessage 函数来获得的。

可以通过 Windows 的 PostMessage 函数来访问其它应用程序。

请参阅

::GetMessage, ::PeekMessage, ::PostMessage, ::PostAppMessage,

CWnd::SendMessage

CWnd::PostNcDestroy

```
virtual void PostNcDestroy( );
```

参数

在窗口被销毁以后，缺省的 OnNcDestroy **成员函数调用这个函数。派生类可以利用这个函数来执行自定义的清除工作，比如删除指针。**

请参阅 CWnd::OnNcDestroy

CWnd::PreCreateWindow

virtual BOOL PreCreateWindow(CREATESTRUCT& *cs*);

返回值

如果要继续窗口的创建过程，则返回非零值；返回 0 **则表明创建过程失败。**

参数

cs

一个 CREATESTRUCT **结构。**

说明

框架在与 CWnd 对象相连接的 Windows 窗口被创建之前调用这个成员函数。

永远不要直接调用这个函数。

这个函数的缺省实现检验窗口类名是否为 NULL，如是，则用适当的缺省值来代替。重载这个函数以在窗口被创建之前修改 CREATESTRUCT 结构。

每个从 CWnd 派生的类都在它重载的 PreCreateWindow 中加入了自己的功能。在设计时，没有描述这些派生的 PreCreatWindow。要确定每个类的适当的风格以及风格之间的相互依赖关系，你可以检查与你的应用程序的基类有关的 MFC 源代码。如果你选择了重载 PreCreateWindow，则你可以使用从 MFC 源代码中收集的信息来确定你的应用程序的基类中使用的风格是否能够提供你需要的功能。

有关改变窗口风格的更多信息参见联机的《Visual C++程序员指南》中的“改变 MFC 创建的窗口的风格”。

请参阅 CWnd::Create, CWnd::CreateEx, CREATESTRUCT

CWnd::PreSubclassWindow

virtual void PreSubclassWindow( );

说明

框架调用这个成员函数以允许在窗口被子类化之前进行其它必要的子类化。重载这个函数以允许控件的动态子类化。这是个高级可重载函数。

请参阅 CWnd::SubclassWindow, CWnd::UnsubclassWindow, CWnd::GetSuperWndProcAddr, CWnd::DefWindowProc, CWnd::SubclassDlgItem, CWnd::Attach, CWnd::PreCreateWindow

CWnd::PreTranslateMessage

virtual BOOL PreTranslateMessage(MSG* *pMsg*);

返回值

如果消息被转换但是不会被分派，则返回非零值；如果消息没有被转换并且要被分派，则返回 0。

参数

pMsg

指向一个 MSG **结构，其中包含了要处理的消息。**

说明

CWinApp 类使用这个函数以在消息被分派到 Windows 的 TranslateMessage 和 DispatchMessage 函数之前进行转换。

请参阅 ::TranslateMessage, ::IsDialogMessage, CWinApp::PreTranslateMessage

CWnd::Print

void Print(CDC* *pDC*, DWORD *dwFlags*) const;

参数

pDC

指向设备环境的指针。

dwFlags

指定了绘图选项。这个参数可以是下列标志中的一个或多个：

- PRF_CHECKVISIBLE 仅当窗口可见时才画出窗口。
- PRF_CHILDREN 画出所有可见的子窗口。
- PRF_CLIENT 画出窗口的客户区。
- PRF_ERASEBKGND 在画出窗口之前擦去背景。
- PRF_NONCLIENT 画出窗口的非客户区。

- PRF_OWNED　画出拥有的所有窗口。

说明

调用这个成员函数以在指定的设备环境中画出当前窗口，通常是在打印机设备环境中。

CWnd::DefWindowProc 函数根据指定的绘图选项处理这个消息：

- 如果指定了 PRF_CHECKVISIBLE 并且窗口不可见，则不做任何操作。
- 如果指定了 PRF_NONCLIENT，则在指定的设备环境中画出非客户区。
- 如果指定了 PRF_ERASEBKGND，则向窗口发送一条 WM_ERASEBKGND 消息。
- 如果指定了 PRF_PRINTCLIENT，则向窗口发送一条 WM_PRINTCLIENT 消息。
- 如果指定了 PRF_PRINTCHILDREN，则向每个可见的子窗口发送一条 WM_PRINT 消息。
- 如果指定了 PRF_OWNED，则向拥有的每个可见窗口发送一条 WM_PRINT 消息。

请参阅　WM_PRINT, WM_PRINTCLIENT

CWnd::PrintClient

```
void PrintClient( CDC* pDC, DWORD dwFlags ) const;
```

参数

pDC

指向设备环境的指针。

dwFlags

指定了绘图选项。这个参数可以是这些标志中的一个或多个：

- PRF_CHECKVISIBLE　仅当窗口可见时才画出窗口。
- PRF_CHILDREN　画出所有可见的子窗口。
- PRF_CLIENT　画出窗口的客户区。
- PRF_ERASEBKGND　在画出窗口之前擦除背景。
- PRF_NONCLIENT　画出窗口的非客户区。
- PRF_OWNED　画出拥有的所有窗口。

说明

调用这个成员函数以在指定的设备环境（通常是一个打印机设备环境）中画出任何窗口。

请参阅 WM_PRINTCLIENT

CWnd::RedrawWindow

BOOL RedrawWindow(LPCRECT *lpRectUpdate* = NULL, CRgn* *prgnUpdate* = NULL, UINT *flags* = RDW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE);

返回值

如果窗口被成功地重画，则返回非零值；否则返回 0。

参数

lpRectUpdate

  指向一个 RECT **结构，其中包含了更新区域的坐标。如果** prgnUpdate **中包含了有效的区域句柄，则这个参数将被忽略。**

prgnUpdate

  表示了更新区域。如果 *prgnUpdate* **和** *lpRectUpdate* **都为** NULL，**则整个客户区将被加入更新区域。**

flags

下面的标志被用于使窗口无效：

- RDW_ERASE 使窗口在重画时接收到一个 WM_ERASEBKGND 消息。必须同时指定 RDW_INVALIDATE 标志；否则 RDW_ERASE 标志将没有效果。
- RDW_FRAME 使窗口非客户区中与更新区域重叠的任何部分接收到一条 WM_NCPAINT 消息。必须同时指定 RDW_INVALIDATE 标志，否则 RDW_FRAME 标志将没有效果。
- RDW_INTERNALPAINT 使一条 WM_PAINT 消息被传递到窗口，而不管窗口是否包含一个无效区域。
- RDW_INVALIDATE 使 *lpRectUpdate* 或 *prgnUpdate*（仅有一个可能为 NULL）无效。如果这个两个参数都为 NULL，则整个窗口都无效。

下面的标志被用于使窗口有效：

- RDW_NOERASE 禁止任何未处理的 WM_ERASEBKGND 消息。
- RDW_NOFRAME 禁止任何未处理的 WM_NCPAINT 消息。这个标志必须与 RDW_VALIDATE 一起使用，通常也与 RDW_NOCHILDREN 一起使用。这个选项必须小心使用，因为它可能会使窗口的某些部分不能正确地画出。
- RDW_NOINTERNALPAINT 禁止任何未处理的内部 WM_PAINT 消息。这个标志不影响从无效区域产生的 WM_PAINT 消息。
- RDW_VALIDATE 使 *lpRectUpdate* 或 *prgnUpdate*（仅有一个可能为

NULL）有效。如果这个两个参数都为 NULL，则整个窗口都有效。这个标志不影响内部 WM_PAINT 消息。

下面的标志控制着何时产生重画动作。除非指定了这些位，否则 RedrawWindow 函数不会执行绘图动作。

- RDW_ERASENOW　如果有必要，则在函数返回前使涉及的窗口（如 RDW_ALLCHILDREN 和 RDW_NOCHILDREN 标志所指定的）接收到 WM_NCPAINT 和 WM_ERASEBKGND 消息。WM_PAINT 消息将被延缓。
- RDW_UPDATENOW　如果有必要，则在函数返回前使涉及的窗口（如 RDW_ALLCHILDREN 和 RDW_NOCHILDREN 标志所指定的）接收到 WM_NCPAINT，WM_ERASEBKGND 和 WM_PAINT 消息。

在缺省情况下，RedrawWindow 函数影响的窗口依赖于指定的窗口是否具有 WS_CLIPCHILDREN 风格。WS_CLIPCHILDREN 窗口的子窗口不会被影响。但是，那些不具有 WS_CLIPCHILDREN 风格的窗口将被递归地有效或无效，直到遇见具有 WS_CLIPCHILDREN 风格的窗口。下面的标志控制着 RedrawWindow 函数将影响哪些窗口：

- RDW_ALLCHILDREN　在重画操作中包含子窗口，如果有的话。
- RDW_NOCHILDREN　在重画操作中不包括子窗口，如果有的化。

说明

这个函数更新给定窗口的客户区中指定的矩形或区域。

当 RedrawWindow **成员函数被用于使捉摸窗口的部分无效的时候，该窗口不接收** WM_PAINT **消息。如果要重画桌面，应用程序必须使用** CWnd::ValidateRgn**，**CWnd::InvalidateRgn**，** CWnd::UpdateWindow **或** ::RedrawWindow**。**

CWnd::ReflectChildNotify

BOOL ReflectChildNotify(UINT *message*, WPARAM *wParam*, LPARAM *lParam*, LRESULT* *pResult*);

返回值

如果消息被反射，则返回 TRUE**；否则返回** FALSE**。**

参数

message

指定了要反射的消息。

wParam

指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

pResult

子窗口生成的结果，将要由父窗口返回。可以是 NULL。

说明

这个消息函数是由框架在 OnChildNotify 内调用的。它是在将消息反射到它的来源时使用的帮助函数。

反射消息是用 CWnd::OnWndMsg 或 CCmdTarget::OnCmdMsg 直接发送的。

有关消息反射的更多信息参见联机的《Visual C++ 程序员指南》中的“处理反射消息”。

请参阅 CWnd::OnChildNotify, CWnd::SendChildNotifyLastMsg, CWnd::OnWndMsg,C CmdTarget::OnCmdMsg, CWnd::ReflectLastMsg

CWnd::ReflectLastMsg

static BOOL PASCAL ReflectLastMsg(HWND *hWndChild*, LRESULT* *pResult* = NULL);

返回值

如果消息被处理，则返回非零值；否则返回 0。

参数

hWndChild

子窗口句柄。

pResult

子窗口生成的结果，将要由父窗口返回。可以是 NULL。

说明

这个成员函数是由框架调用的，用于将最近的消息反射到子窗口。

如果 hWndChild 标识的窗口是一个 OLE 控件或永久映射中的窗口，则这个成员函数调用 SendChildNotifyLastMsg。

有关消息反射的更多信息参见 Visual C++ 程序员联机指南中的“处理反射消息”。

请参阅 CWnd::OnChildNotify, CWnd::SendChildNotifyLastMsg, CWnd::ReflectChildNotify, CCmdTarget::OnCmdMsg

CWnd::ReleaseDC

```
int ReleaseDC( CDC* pDC );
```

返回值

如果成功，则返回非零值；否则返回 0。

参数

pDC
标识了要释放的设备环境。

说明

释放设备环境，以供其它应用程序使用。ReleaseDC **成员函数的效果依赖于设备环境的类型。**

对于 GetWindowDC **和** GetDC **成员函数的每一次调用，应用程序都应当调用** ReleaseDC **成员函数。**

请参阅　CWnd::GetDC, CWnd::GetWindowDC, ::ReleaseDC

CWnd::RepositionBars

void RepositionBars(UINT *nIDFirst*, UINT *nIDLast*, UINT *nIDLeftOver*, UINT *nFlag* = CWnd::reposDefault, LPRECT *lpRectParam* = NULL, LPCRECT *lpRectClient* = NULL, BOOL *bStretch* = TRUE);

参数

nIDFirst

要重新定位并改变大小的控制条范围中的第一个控制条的 ID。

nIDLast

要重新定位并改变大小的控制条范围中的最后一个控制条的 ID。

nIDLeftOver

指定了填充客户区其余部分的方格的 ID。

nFlag

可以具有下列值：

- CWnd::reposDefault **实现控制条的布局。***lpRectParam* **没有被使用，可以是** NULL。
- CWnd::reposQuery **没有执行控制条的布局，相反用客户区的大小初始化了** lpRectParam**，就像已经完成了布局一样。**

- CWnd::reposExtra 将 lpRectParam 的值加到 nIDLast 的客户区上，并执行布局。

lpRectParam

指向一个 RECT 结构；其用法依赖于 *nFlag* 的值。

lpRectClient

指向一个 RECT 结构，其中包含了可用的客户区。如果为 NULL，则窗口的客户区将被使用。

bStretch

指明控制条是否要被缩放到框架的大小。

说明

调用这个函数以在窗口的客户区中重定位控制条并改变其大小。nIDFirst 和 nIDLast 参数定义了要在客户区内重定位的控制条 ID 的范围。nIDLeftOver 参数指定了被用来重定位并改变大小，以填充客户区中没有被控制条覆盖的区域的子窗口（通常是视）的 ID。

请参阅 CFrameWnd::RecalcLayout

CWnd::RunModalLoop

int RunModalLoop(DWORD *dwFlags*);

返回值

指定了传递给 EndModalLoop 成员函数的 *nResult* 参数的值，随后该函数被用于结束模式循环。

参数

dwFlags

指定了要被发送的 Windows 消息。可以是下列值之一：

- MLF_NOIDLEMSG　不向父窗口发送 WM_ENTERIDLE 消息。
- MLF_NOKICKIDLE　不向窗口发送 WM_KICKIDLE 消息。
- MLF_SHOWONIDLE　当消息队列变为空闲时显示窗口。

说明

调用这个成员函数以获得、转换或分派消息，直到 ContinueModal 返回 FALSE。在缺省情况下，ContinueModal 在调用 EndModalLoop 之后返回 FALSE。返回作为 EndModalLoop 的 *nResult* 提供的值。

请参阅　EndModalLoop, WM_ENTERIDLE

CWnd::ScreenToClient

void ScreenToClient(LPPOINT *lpPoint*) const;
void ScreenToClient(LPRECT *lpRect*) const;

参数

lpPoint
　　指向一个CPoint对象或POINT结构，其中包含了要转换的屏幕坐标。

lpRect
　　指向一个CRect对象或RECT结构，其中包含了要转换的屏幕坐标。

说明

将显示器上给定点或矩形的屏幕坐标转换为客户坐标。

ScreenToClient成员函数将lpPoint或lpRect给定的屏幕坐标替换为客户坐标。新的坐标是相对于CWnd客户区的左上角的。

请参阅　CWnd::ClientToScreen, ::ScreenToClient

CWnd::ScrollWindow

void ScrollWindow(int *xAmount*, int *yAmount*, LPCRECT *lpRect* = NULL, LPCRECT *lpClipRect* = NULL);

参数

xAmount

指定了水平滚动的量，使用设备单位。在左滚时，该参数必须为负。

yAmount

指定了垂直滚动的量，使用设备单位。在上滚时，该参数必须为负。

lpRect

指向一个 CRect **对象或** RECT **结构，指定了要滚动的客户区的部分。如果** lpRect **为** NULL**，则将滚动整个客户区。如果光标区域与滚动矩形重叠，则插字符将被重定位。**

lpClipRect

指向一个 CRect **对象或** RECT **结构，指定了要滚动的裁剪区域。只有这个矩形中的位才会被滚动。在矩形之外的位不会被影响，即使它们是在** *lpRect* **矩形之内。如果** *lpClipRect* **为** NULL**，则不会在滚动矩形上进行裁剪。**

说明

这个函数滚动当前 CWnd 对象的客户区内容。

如果插字符在要滚动的 CWnd 之内，则 ScrollWindow 自动将插字符隐藏，以避免它被擦除，然后当滚动完成以后，再恢复插字符。插字符的位置将相应地调整。

ScrollWindow 成员函数所涉及的区域将不会被重画，但是将被加入当前 CWnd 对象的更新区域。应用程序最终将接收到一条 WM_PAINT 消息，通知它这个区域需要重画。要在滚动完成的同时重画涉及的区域，则应在调用 ScrollWindow 之后立即调用 UpdateWindow 成员函数。

如果 *lpRect* 为 NULL，则窗口的任何子窗口的位置将被设为 *xAmout* 和 *yAmout* 指定的偏移，并且 CWnd 中任何无效（未画出）区域也被加上偏移。当 *lpRect* 为 NULL 的时候，ScrollWindow 更快一些。

如果 *lpRect* 不为 NULL，则子窗口的位置不发生变化，并且 CWnd 的无效区域也没有偏移。当 *lpRect* 为 NULL 的时候，如果要防止更新问题，则应在调用 ScrollWindow 之前调用 UpdateWindow 成员函数以重画 CWnd。

请参阅　CWnd::UpdateWindow, ::ScrollWindow

CWnd::ScrollWindowEx

int ScrollWindowEx(int *dx*, int *dy*, LPCRECT *lpRectScroll*, LPCRECT *lpRectClip*, CRgn* *prgnUpdate*, LPRECT *lpRectUpdate*, UINT *flags*);

返回值

如果函数成功，则返回值为 SIMPLEREGION**（矩形的无效矩形），**COMPLEXREGION**（非矩形无效区域；重叠矩形）或** NULLREGION**（没有无效区域）；否则返回值为** ERROR**。**

参数

dx

指定了水平滚动的量，使用设备单位。在左滚时，该参数必须为负。

dy

指定了垂直滚动的量，使用设备单位。在上滚时，该参数必须为负。

lpRectScroll

指向 RECT **结构，指定了要滚动的客户区的部分。如果该参数为** NULL**，则将滚动整个客户区。**

lpRectClip

指向 RECT 结构，指定了要滚动的裁剪区域。这个结构优先于 lpRectScroll 指定的矩形。只有这个矩形中的位才会被滚动。在矩形之外的位不会被影响，即使它们是在 *lpRectScroll* 矩形之内。如果这个参数为 NULL，则不会在滚动矩形上进行裁剪。

prgnUpdate

标识了被修改的区域，用于保存因滚动而无效的区域。这个参数可能为 NULL。

lpRectUpdate

指向一个 RECT 结构，该结构将接收因滚动而无效的矩形的边界。这个参数可能为 NULL。

flags

可以具有下列值之一：

- SW_ERASE　当与 SW_INVALIDATE 一起设置时，向窗口发送一条 WM_ERASEBKGND 消息以擦除新的无效区域。
- SW_INVALIDATE　在滚动后使 *prgnUpdate* 标识的区域无效。
- SW_SCROLLCHILDREN　将与 *lpRectScroll* 所指定的矩形相交的所有子窗口滚动 *dx* 和 *dy* 所指定的数目的像素。Windows 向与 *lpRectScroll* 相交的所有子窗口发送 WM_MOVE 消息，即使它们没有移动。当子窗口被滚动并且光标矩形与滚动矩形相交时，插字符被重定位。

说明

这个函数滚动窗口的客户区内容。该函数与 ScrollWindow 函数类似，并具有一些附加特性。

如果没有指定 SW_INVALIDATE 和 SW_ERASE，则 ScrollWindowEx 成员函数并不使滚动的区域无效。如果设置了这两者之一，则 ScrollWindowEx 将使区域无效。在应用程序调用 UpdateWindow 成员函数，或调用 RedrawWindow 成员函数（指定了 RDW_UPDATENOW 或 RDW_ERASENOW），或从应用程序队列中获得 WM_PAINT 消息之前，这个区域将不会被更新。

如果窗口具有 WS_CLIPCHILDREN 风格，则 prgnUpdate 和 lpRectUpdate 指定的区域代表了必须更新的滚动窗口的全部区域，包括子窗口中所有需要更新的区域。

如果指定了 SW_SCROLLCHILDREN 标志，并且滚动了子窗口的一个部分，则 Windows 将不会正确地更新屏幕。在源矩形之外的滚动的子窗口的部分将不会被擦除，并且在新的位置也不会被正确地重画。使用 Windows 的 DeferWindowPos 函数以移动不完全位于 lpRectScroll 矩形之内的子窗口。如果设置了 SW_SCROLLCHILDREN 标志并且插字符矩形与滚动矩形相交，则光标将被重新定位。

所有输入和输出的坐标（属于 lpRectScroll，lpRectClip，lpRectUpdate 和 prgnUpdate）都被假定使用了客户坐标，而不管窗口具有的是 CS_OWNDC 风

格还是CS_CLASSDC风格。如果有必要，可以使用Windows的LPtoDP和DPtoLP函数将该坐标与逻辑坐标相互转换。

请参阅 CWnd::RedrawWindow, CDC::ScrollDC, CWnd::ScrollWindow, CWnd::UpdateWindow, ::DeferWindowPos, ::ScrollWindowEx

CWnd::SendChildNotifyLastMsg

BOOL SendChildNotifyLastMsg(LRESULT* *pResult* = NULL);

返回值

如果子窗口已经处理了发送给它的父窗口的消息，则返回非零值；否则返回0。

参数

pResult
子窗口产生的结果，将通过父窗口返回。

说明

这个成员函数被框架调用，用于从父窗口向子窗口提供通知消息，子窗口可以借此处理任务。

如果该消息为反射消息，SendChildNotifyLastMsg将当前消息发送给它的来源。有关消息反射的更多信息参见联机的《Visual C++ 程序员指南》中的“处理反射消息”。

请参阅 CWnd::OnChildNotify

CWnd::SendDlgItemMessage

LRESULT SendDlgItemMessage(int *nID*, UINT *message*, WPARAM *wParam* = 0, LPARAM *lParam* = 0);

返回值

指定了控件的窗口过程返回的值。如果没有找到控件，则返回0。

参数

nID

指定了对话框控件的标识符，该控件将接收消息。

message

指定了要发送的消息。

wParam

指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

说明

这个函数向控件发送一条消息。

SendDlgItemMessage 直到消息被处理之后才会返回。

使用 SendDlgItemMessage 等同于获得给定控件的 CWnd* 指针并调用其 SendMessage 成员函数。

请参阅　CWnd::SendMessage, ::SendDlgItemMessage

CWnd::SendMessage

LRESULT SendMessage(UINT *message*, WPARAM *wParam* = 0, LPARAM *lParam* = 0);

返回值

消息处理的结果；它的值依赖于发送的消息。

参数

message

指定了要发送的消息。

wParam

指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

说明

这个函数向窗口发送指定的消息。SendMessage 成员函数直接调用窗口过程并在窗口过程处理了消息以后才返回。这与 PostMessage 成员函数形成对比，该函数将消息放入窗口的消息队列并立即返回。

请参阅 ::InSendMessage, CWnd::PostMessage, CWnd::SendDlgItemMessage, ::SendMessage

CWnd::SendMessageToDescendants

void SendMessageToDescendants(UINT *message*, WPARAM *wParam* = 0, LPARAM *lParam* = 0, BOOL *bDeep* = TRUE, BOOL *bOnlyPerm* = FALSE);

参数

message

指定了要发送的消息。

wParam

指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

bDeep

指定了搜索的深度。如果为 TRUE，则递归搜索所有的子窗口；如果为 FALSE，则仅搜索直接子窗口。

bOnlyPerm

指定该消息是否将被临时窗口接收。如果为 TRUE，则临时窗口可以接收这个消息；如果为 FALSE，则只有永久窗口才能接收这个消息。有关临时窗口的更多信息参见技术注释 3。

说明

调用这个成员函数以向所有的后代窗口发送指定的 Windows 消息。

如果 bDeep 为 FALSE，则消息仅被发送到窗口的直接子窗口，否则消息被发

送到所有的后代窗口。

如果 bDeep 和 bOnlyPerm 为 TRUE，则在临时窗口之下进行搜索。在这种情况下，只有在搜索过程中遇到的永久窗口才能接收消息。如果 bDeep 为 FALSE，则消息仅被发送到窗口的直接子窗口。

请参阅　CWnd::SendMessage, CWnd::FromHandlePermanent, CWnd::FromHandle

CWnd::SendNotifyMessage

BOOL SendNotifyMessage(UINT *message*, WPARAM *wParam*, LPARAM *lParam*);

返回值

如果函数成功，则返回非零值；否则返回 0。

参数

message
　　指定了要发送的消息。

wParam
　　指定了与消息有关的附加信息。

lParam

指定了与消息有关的附加信息。

说明

这个函数向窗口发送指定的消息。如果窗口是由调用线程创建的，则SendNotifyMessage调用窗口的窗口过程，并在窗口处理了消息之后返回。如果窗口是由其它线程创建的，则SendNotifyMessage将消息传递给窗口过程并立即返回；它并不等待窗口过程结束处理消息。

请参阅 CWnd::SendMessage, ::SendNotifyMessage

CWnd::SetActiveWindow

CWnd* SetActiveWindow( );

返回值

原来活动的窗口。

返回的指针可能是临时的，不能被保存以供将来使用。

说明

这个函数使 CWnd 成为活动窗口。

SetActiveWindow 成员函数必须小心使用，因为它允许应用程序任意地接管活动窗口和输入焦点。通常，Windows 管理着所有的活动。

请参阅 ::SetActiveWindow, CWnd::GetActiveWindow

CWnd::SetCapture

```
CWnd* SetCapture( );
```

返回值

原来接收所有鼠标输入的窗口的指针。如果没有这样的窗口，则返回值为 NULL。返回的指针可能是临时的，不能被保存以供将来使用。

说明

这个函数使随后的所有鼠标输入都被发送到当前的 CWnd 对象，并不考虑光标的位置。

当 CWnd 不再需要所有的鼠标输入时，应用程序应当调用 ReleaseCapture 函数

以使其它窗口能够接收鼠标输入。

请参阅　::ReleaseCapture, ::SetCapture, CWnd::GetCapture

CWnd::SetCaretPos

static void PASCAL SetCaretPos(POINT *point*);

参数

point

　　指定了插字符的新的 x 和 y 坐标（客户坐标）。

说明

这个函数设置插字符的位置。

SetCaretPos 成员函数仅当插字符属于当前任务的一个窗口时才移动它。不论插字符是否隐藏，SetCaretPos 都移动插字符的位置。

插字符是一种共享资源。如果窗口不拥有插字符，它就不应移动插字符。

请参阅　CWnd::GetCaretPos, ::SetCaretPos

CWnd::SetClipboardViewer

HWND SetClipboardViewer( );

返回值

如果成功，则返回剪贴板观察器链中下一个窗口的句柄。应用程序必须保存这个句柄（可以当作成员变量保存），并且在响应剪贴板观察器链消息的时候使用它。

说明

每当剪贴板的内容发生变化时，这个函数将窗口加入被通知的窗口链（通过WM_DRAWCLIPBOARD消息）。

属于剪贴板观察器链的窗口必须响应WM_DRAWCLIPBOARD、WM_CHANGECBCHAIN和WM_DESTROY消息，并将消息传递到链中的下一个窗口。

这个成员函数向窗口发送一条WM_DRAWCLIPBOARD消息。由于剪贴板观察器链中下一个窗口的句柄还没有返回，应用程序不应传递它在SetClipboardViewer调用过程中接收到的WM_DRAWCLIPBOARD消息。

如果要将自己从剪贴板观察器链中去掉，则应用程序必须调用ChangeClipboard

成员函数。

请参阅 CWnd::ChangeClipboardChain, ::SetClipboardViewer

CWnd::SetDlgCtrlID

int SetDlgCtrlID(int *nID*);

返回值

如果成功，则返回窗口以前的标识符；否则返回0。

参数

nID

被设为控件标识符的新值。

说明

这个函数将窗口的窗口ID或控制ID设为新值。这里的窗口可以是任何子窗口，不仅是对话框中的控件。这种窗口不能是顶层窗口。

请参阅 CWnd::GetDlgCtrlID, CWnd::Create, CWnd::CreateEx, CWnd::GetDlgItem

CWnd::SetDlgItemInt

void SetDlgItemInt(int *nID*, UINT *nValue*, BOOL *bSigned* = TRUE);

参数

nID

指定了要改变的控件的整数 ID。

nValue

指定了用于生成项目文本的整数值。

bSigned

指定这个整数是带符号的还是无符号的。如果这个参数为 TRUE，则 *nValue* 为带符号的。如果这个参数为 TRUE 并且 *nValue* 小于 0，则在字符串中，将在第一个数字之前加上负号。如果这个参数为 FALSE，则 *nValue* 为无符号的。

说明

这个函数将对话框中给定控件的文本设为代表指定整数值的字符串。

SetDlgItemInt 向给定的控件发送一条 WM_SETTEXT 消息。

请参阅 CWnd::GetDlgItemInt, ::SetDlgItemInt, WM_SETTEXT

CWnd::SetDlgItemText

void SetDlgItemText(int *nID*, LPCTSTR *lpszString*);

参数

nID
　　要设置文本的控件的标识符。

lpszString
　　指向一个 CString **对象或以** null **结尾的字符串，其中包含了要拷贝到控件的文本。**

说明

设置窗口或对话框拥有的标题或是控件文本。

SetDlgItemText **向给定的控件发送一条** WM_SETTEXT **消息。**

请参阅 ::SetDlgItemText, WM_SETTEXT, CWnd::GetDlgItemText

CWnd::SetForegroundWindow

BOOL SetForegroundWindow( );

返回值

如果函数成功，则返回非零值；否则返回0。

说明

这个函数将创建窗口的线程推向前台并激活该窗口。键盘输入被定向到这个窗口，并且将向用户显示不同的视觉提示。前台窗口是指用户当前工作的窗口。前台窗口仅针对顶层窗口而言（框架窗口或对话框）。

请参阅 CWnd::GetForegroundWindow

CWnd::SetFocus

```
CWnd* SetFocus( );
```

返回值

原来拥有输入焦点的窗口对象的指针。如果没有这样的窗口，则返回值为NULL。返回的指针可能是临时的，不应被保存。

说明

这个函数要求得到输入焦点。输入焦点将随后的所有键盘输入定向到这个窗

口。原来拥有输入焦点的任何窗口都将失去它。

SetFocus成员函数项失去输入焦点的窗口发送一条WM_KILLFOCUS消息，并向接收输入焦点的窗口发送一条WM_SETFOCUS消息。它还激活该窗口或它的父窗口。

如果当前窗口是激活的，但是不具有输入焦点（这意味着，没有窗口具有输入焦点），则任何按下的键都将产生WM_SYSCHAR，WM_SYSKEYDOWN或WM_SYSKEYUP消息。

请参阅 ::SetFocus, CWnd::GetFocus

CWnd::SetFont

void SetFont(CFont* *pFont*, BOOL *bRedraw* = TRUE);

参数

pFont

指定了新的字体。

bRedraw

如果为TRUE，则重画CWnd对象。

说明

这个函数将窗口的当前字体设为指定的字体。如果 bRedraw **为** TRUE**，则窗口还会被重画。**

请参阅 CWnd::GetFont, WM_SETFONT

CWnd::SetIcon

HICON SetIcon(HICON *hIcon*, BOOL *bBigIcon*);

返回值

指向一个图标的句柄。

参数

hIcon

以前图标的句柄。

bBigIcon

如果为 TRUE**，则指定了** 32×32 **像素的图标；如果为** FALSE**，则指定了** 16×16 **像素的图标。**

说明

调用这个函数以将图标设为 hIcon 所标识的指定图标。如果注册了窗口类，则它选择一个图标。

请参阅 GetIcon

CWnd::SetMenu

```
BOOL SetMenu( CMenu* pMenu );
```

返回值

如果菜单发生了变化，则返回非零值；否则返回 0。

参数

pMenu

标识了新的菜单。如果这个参数为 NULL，则当前菜单被清除。

说明

这个函数将当前菜单设为指定的菜单。它使窗口被重画以反映菜单的变化。

SetMenu 不会销毁以前的菜单。应用程序必须调用 CMenu::DestroyMenu 成员函数以完成这个任务。

请参阅 CMenu::DestroyMenu, CMenu::LoadMenu, ::SetMenu, CWnd::GetMenu

CWnd::SetOwner

void SetOwner(CWnd* *pOwnerWnd*);

参数

pOwnerWnd

标识了窗口对象的新拥有者。如果这个参数为 NULL，则窗口对象没有拥有者。

说明

这个函数将当前窗口的拥有者设为指定的窗口对象。然后这个拥有者就可以从当前窗口对象接收命令消息。在缺省情况下，当前窗口的父窗口是其拥有者。

通常在与窗口层次关系不相关的窗口之间建立联系是很有用的。例如，CToolBar 向它的拥有者而不是父窗口发送通知消息。这允许工具条在向其它窗口（例如现场框架窗口）发送通知的时候变成某个窗口（例如 OLE 容器应用程序的窗

口）的子窗口。此外，当服务器窗口在现场编辑时失去活动状态或被激活时，由框架窗口拥有的任何窗口都被隐藏或显示。这种拥有关系是通过调用 SetOwner 来明确设置的。

这个函数中的拥有概念与 GetWindow 中的拥有概念不同。

请参阅 CWnd::GetOwner, CToolBar

CWnd::SetParent

CWnd* SetParent(CWnd* *pWndNewParent*);

返回值

如果成功，则返回以前的父窗口的指针。返回的指针可能是临时的，不应被保存以供将来使用。

参数

pWndNewParent
标识了新的父窗口。

说明

这个函数改变子窗口的父窗口。

如果子窗口可见，则 Windows 执行适当的重画和重绘工作。

请参阅 ::SetParent, CWnd::GetParent

CWnd::SetProperty

void SetProperty(DISPID *dwDispID*, VARTYPE *vtProp*, ...);

参数

dwDispID

标识了要设置的属性。这个值通常是由组件廊提供的。

vtProp

指定了要设置的属性的类型。可能的取值参见 COleDispatchDriver::InvokeHelper 中的说明部分。

...

vtProp 指定的类型的单个参数。

说明

调用这个成员函数以设置 dwDispID 所标识的 OLE 控件属性。

> **注意** 这个函数仅应在代表 OLE 控件的 CWnd 对象中调用。

有关在 OLE 控件容器中使用这个成员函数的更多信息参见联机的《Visual C++ 程序员指南》中的文章“ActiveX 控件容器：在 ActiveX 控件容器中编写 ActiveX 控件”。

请参阅 CWnd::InvokeHelper, COleDispatchDriver, CWnd::CreateControl

CWnd::SetRedraw

void SetRedraw(BOOL *bRedraw* = TRUE);

参数

bRedraw

指定了重画标志的状态。如果这个参数为 TRUE，则重画标志被设置；如果为 FALSE，则该标志被清除。

说明

应用程序调用 SetRedraw 以允许重画变化或防止变化被重画。

这个成员函数设置或清除重画标志。当重画标志被清除时，在每次变化以后，内容不会更新，直到重画标志被设置才会重新绘出。例如，如果一个应用程序需要在列表框中加入几个项，则可以清除重画标志，加入项，然后设置重画标志。最后，应用程序可以调用 Invalidate 或 InvalidateRect 成员函数以使列表框被重画。

请参阅 WM_SETREDRAW

CWnd::SetScrollInfo

BOOL SetScrollInfo(int *nBar*, LPSCROLLINFO *lpScrollInfo*, BOOL *bRedraw* = TRUE);

返回值

如果成功，则返回 TRUE；否则返回 FALSE。

参数

nBar

指定滚动条是一个控件还是窗口非客户区的一部分。如果它是非客户区的一部分，则 *nBar* 也指定了滚动条是水平的、垂直的，还是都有。它可以是下列值之一：

- SB_CTL　包含滚动条的控制参数。M_hWnd 的数值必须是滚动条控件的句柄。
- SB_HORZ　指定了窗口的水平滚动条。
- SB_VERT　指定了窗口的垂直滚动条。

lpScrollInfo

指向 SCROLLINFO 结构的指针。有关这个结构的更多信息参见《Win32 SDK 程序员参考》。

bRedraw

指定滚动条是否应被重画以反映新的位置。如果 bRedraw 为 TRUE，则滚动条将被重画。如果它为 FALSE，则它不会被重画。缺省情况下滚动条将被重画。

说明

调用这个成员函数以设置 SCROLLINFO 结构为滚动条维护的信息。

SCROLLINFO 结构中包含了有关滚动条的信息，包括最小和最大滚动位置、页面大小和滚动块的位置。有关改变这个结构缺省值的更多信息参见《Win32

SDK 程序员参考》中的 SCROLLINFO 主题。

指明滚动条位置的 MFC Windows 消息处理函数，CWnd::OnHScroll 和 CWnd::OnVScroll，仅提供 16 位的位置数据。GetScrollInfo 和 SetScrollInfo 提供了 32 位的滚动条位置数据。因而，在处理 CWnd::OnHScroll 或 CWnd::OnVScroll 的时候，应用程序可以调用 GetScrollInfo 以获得 32 位的滚动条位置数据。

注意　CWnd::GetScrollInfo 允许应用程序使用 32 位的滚动条位置。

请参阅　CWnd::GetScrollInfo, CWnd::SetScrollPos, CWnd::OnVScroll, CWnd::OnHScroll, SCROLLINFO

CWnd::SetScrollPos

int SetScrollPos(int *nBar*, int *nPos*, BOOL *bRedraw* = TRUE);

返回值

滚动块的以前位置。

参数

nBar

指定了要设置的滚动条。这个参数可以是下列值：

- SB_HORZ　设置窗口的水平滚动条的滚动块位置。
- SB_VERT　设置窗口的垂直滚动条的滚动块位置。

nPos

指定了滚动块的新位置。它必须在滚动范围之内。

bRedraw

指定滚动条是否应被重画以反映新的滚动块位置。如果这个参数为TRUE，则滚动条将被重画；如果为FALSE，则滚动条不会被重画。

说明

这个函数设置滚动块的当前位置，并且如果需要，则重画滚动条以反映滚动块的新位置。

当滚动条需要在随后对别的函数的调用中重画时，将 bRedraw 设为 FALSE 将会是有用的。

请参阅　::SetScrollPos, CWnd::GetScrollPos, CScrollBar::SetScrollPos

CWnd::SetScrollRange

void SetScrollRange(int *nBar*, int *nMinPos*, int *nMaxPos*, BOOL *bRedraw* =

TRUE);

参数

nBar

指定了要设置的滚动条。这个参数可以取下列值：

- SB_HORZ　**设置窗口的水平滚动条的范围。**
- SB_VERT　**设置窗口的垂直滚动条的范围。**

nMinPos

指定滚动位置的最小值。

nMaxPos

指定滚动位置的最大值。

bRedraw

指定滚动条是否需要被重画以反映变化。如果 bRedraw 为 TRUE，则滚动条被重画；如果为 FALSE，则滚动条不会被重画。

说明

这个函数设置给定滚动条的最小和最大位置。它也可以被用于隐藏或显示标志

的滚动条。

应用程序不应在处理滚动条通知消息的时候调用这个函数以隐藏滚动条。

如果在调用了 SetScrollPos 成员函数之后立即调用 SetScrollRange，则 SetScrollPos 成员函数中的 bRedraw 参数必须为 0 以防止滚动条被重画两次。

标准滚动条的缺省范围是 0 到 100。滚动条控件的缺省范围为空（*nMinPos* 和 *nMaxPos* 的值都为 0）。*nMinPos* 和 *nMaxPos* 所指定的值之间的差不能大于 INT_MAX。

请参阅 CWnd::SetScrollPos, ::SetScrollRange, CWnd::GetScrollRange

CWnd::SetTimer

UINT SetTimer(UINT *nIDEvent*, UINT *nElapse*, void (CALLBACK EXPORT* *lpfnTimer*) (HWND, UINT, UINT, DWORD));

返回值

如果函数成功，则返回新定时器的标识符。应用程序可以将这个值传递给 KillTimer 成员函数以销毁定时器。如果成功，则返回非零值；否则返回 0。

参数

nIDEvent

指定了不为零的定时器标识符。

nElapse

指定了定时值；以毫秒为单位。

lpfnTimer

指定了应用程序提供的 TimerProc 回调函数的地址，该函数被用于处理 WM_TIMER 消息。如果这个参数为 NULL，则 WM_TIMER 消息被放入应用程序的消息队列并由 CWnd 对象来处理。

说明

这个函数设置一个系统定时器。指定了一个定时值，每当发生超时，则系统就向设置定时器的应用程序的消息队列发送一个 WM_TIMER 消息，或者将消息传递给应用程序定义的 TimerProc 回调函数。

lpfnTimer 回调函数不需要被命名为 TimerProc，但是它必须按照如下方式定义：

```
void CALLBACK EXPORT TimerProc(
    HWND hWnd,          // 调用 SetTimer 的 CWnd 的句柄
    UINT nMsg,          // WM_TIMER
    UINT nIDEvent       // 定时器标识
```

```
    DWORD dwTime      // 系统时间
);
```

定时器是有限的全局资源；因此对于应用程序来说，检查 SetTimer 返回的值以确定定时器是否可用是很重要的。

请参阅 WM_TIMER, CWnd::KillTimer, ::SetTimer

CWnd::SetWindowContextHelpId

BOOL SetWindowContextHelpId(DWORD *dwContextHelpId*);

返回值

如果函数成功，则返回非零值；否则返回 0。

参数

dwContextHelpId
帮助上下文标识符。

说明

调用这个成员函数以将帮助上下文标识符与指定的窗口相关联。

如果子窗口不具有帮助上下文标识符，则它继承它的父窗口的标识符。同理，如果被拥有的窗口不具有帮助上下文标识符，则它继承它的拥有者窗口的标识符。这种对帮助上下文标识符的继承允许应用程序只为对话框和它的所有控件设置一个标识符。

请参阅 CWnd::GetWindowContextHelpId

CWnd::SetWindowPlacement

BOOL SetWindowPlacement(const WINDOWPLACEMENT**lpwndpl*);

返回值

如果函数成功则返回非零值；否则返回0。

参数

lpwndpl

指向一个WINDOWPLACEMENT结构，指定了新的显示状态和位置。

说明

这个函数设置显示状态和窗口的正常（复原）、最小化和最大化位置。

请参阅 CWnd::GetWindowPlacement, ::SetWindowPlacement

CWnd::SetWindowPos

BOOL SetWindowPos(const CWnd* *pWndInsertAfter*, int *x*, int *y*, int *cx*, int *cy*, UINT *nFlags*);

返回值

如果函数成功，则返回非零值；否则返回0。

参数

pWndInsertAfter
标识了在Z轴次序上位于这个CWnd对象之前的CWnd对象。这个参数可以是指向CWnd对象的指针，也可以是指向下列值的指针：

- wndBottom 将窗口放在Z轴次序的底部。如果这个CWnd是一个顶层窗口，则窗口将失去它的顶层状态；系统将这个窗口放在其它所有窗口的底部。
- wndTop 将窗口放在Z轴次序的顶部。
- wndTopMost 将窗口放在所有非顶层窗口的上面。这个窗口将保持它的顶层位置，即使它失去了活动状态。

- wndNoTopMost 将窗口重新定位到所有非顶层窗口的顶部（这意味着在所有的顶层窗口之下）。这个标志对那些已经是非顶层窗口的窗口没有作用。

有关这个函数以及这些参数的使用规则参见说明部分。

x

指定了窗口左边的新位置。

y

指定了窗口顶部的新位置。

cx

指定了窗口的新宽度。

cy

指定了窗口的新高度。

nFlags

指定了大小和位置选项。这个参数可以是下列值的组合：

- SWP_DRAWFRAME 围绕窗口画出边框（在创建窗口的时候定义）。
- SWP_FRAMECHANGED 向窗口发送一条 WM_NCCALCSIZE 消息，即使窗口的大小不会改变。如果没有指定这个标志，则仅当窗口的大小发生变化时才发送 WM_NCCALCSIZE 消息。
- SWP_HIDEWINDOW 隐藏窗口。

- SWP_NOACTIVATE　不激活窗口。如果没有设置这个标志，则窗口将被激活并移动到顶层或非顶层窗口组（依赖于 *pWndInsertAfter* 参数的设置）的顶部。
- SWP_NOCOPYBITS　废弃这个客户区的内容。如果没有指定这个参数，则客户区的有效内容将被保存，并在窗口的大小或位置改变以后被拷贝回客户区。
- SWP_NOMOVE　保持当前的位置（忽略 *x* 和 *y* 参数）。
- SWP_NOOWNERZORDER　不改变拥有者窗口在 Z 轴次序上的位置。
- SWP_NOREDRAW　不重画变化。如果设置了这个标志，则不发生任何种类的变化。这适用于客户区、非客户区（包括标题和滚动条）以及被移动窗口覆盖的父窗口的任何部分。当这个标志被设置的时候，应用程序必须明确地无效或重画要重画的窗口和父窗口的任何部分。
- SWP_NOREPOSITION　与 SWP_NOOWNERZORDER 相同。
- SWP_NOSENDCHANGING　防止窗口接收 WM_WINDOWPOSCHANGING 消息。
- SWP_NOSIZE　保持当前的大小（忽略 *cx* 和 *cy* 参数）。
- SWP_NOZORDER　保持当前的次序（忽略 *pWndInsertAfter*）。
- SWP_SHOWWINDOW　显示窗口。

说明

调用这个成员函数以改变子窗口、弹出窗口和顶层窗口的大小、位置和 Z 轴次序。

窗口在屏幕上按照它们的 Z 轴次序排序。在 Z 轴次序上处于顶端的窗口将程序在所有其它窗口的顶部。

子窗口的所有坐标都是客户坐标（相对于父窗口客户区的左上角）。

窗口可以被移动到 Z 轴次序的顶部，既可以通过将 pWndInsertAfter 参数设为 &wndTopMost，并确保没有设置 SWP_NOZORDER 标志，也可以通过设置窗口的 Z 轴次序使它位于所有现存的顶层窗口上方。当一个非顶层窗口被设为顶层窗口时，它拥有的窗口也被设为顶层的。它的拥有者不发生变化。

如果顶层窗口被重新定位到 Z 轴次序的底部（&wndBottom）或任何非顶层窗口之后，则它将不再是顶层窗口。当顶层窗口被变为非顶层窗口时，它所有的拥有者和它拥有的所有窗口都被变为非顶层窗口。

如果既没有指定 SWP_NOACTIVE 标志也没有指定 SWP_NOZORDER 标志（这意味着应用程序要求窗口被同时激活并放入指定的 Z 轴次序），则 pWndInsertAfter 参数中指定的值将只在下列环境下适用：

- 在 pWndInsertAfter 参数中既没有指定 &wndTopMost 也没有指定 &wndNoTopMost。

- 这个窗口不是活动窗口。

应用程序不能激活一个非活动窗口但同时又不把它带到 Z 轴次序的顶部。应用程序可以没有任何限制地改变活动窗口的 Z 轴次序。

非顶层窗口可能拥有一个顶层窗口，但是反之则不成立。任何被顶层窗口拥有的窗口（例如对话框）都将自己变为顶层窗口，以确保所有被拥有的窗口位于它们的拥有者上方。

在 Windows 3.1 或更新的版本中，可以将窗口移动到 Z 轴次序的顶部，并通过设置它们的 WS_EX_TOPMOST 风格而将之锁定在那里。这种顶层窗口即使在失去活动状态以后也会保持顶层位置。例如，选择 WinHelp 的 Always On Top 命令会使帮助窗口变为顶层，并且在你返回应用程序之后它还保持可见。

要创建一个顶层窗口，应在调用 SetWindowPos 的时候将 pWndInsertAfter 参数设为 *&wndTopMost*，或者在创建窗口的时候设置 WS_EX_TOPMOST 风格。

如果 Z 轴次序中包含了任何具有 WS_EX_TOPMOST 风格的窗口，则用 &wndTopMost 移动的窗口将被放到所有非顶层窗口的顶部，但是位于任何顶层窗口的下面。当应用程序激活一个不具有 WS_EX_TOPMOST 风格的非活动窗口时，该窗口将被移动到所有非顶层窗口的上方，但是位于所有顶层窗口的下方。

如果在调用 SetWindowPos 的时候 pWndInsertAfter 参数被设为&wndBottom，并且 CWnd 是一个顶层窗口，则该窗口失去顶层状态（WS_EX_BOTTOM 风

格被清除），并且系统将窗口放在 Z 轴次序的底部。

请参阅 ::DeferWindowPos, ::SetWindowPos

CWnd::SetWindowRgn

int SetWindowRgn(HRGN *hRgn*, BOOL *bRedraw*);

返回值

如果函数成功，则返回值为非零值。如果函数失败，则返回值为零。

参数

hRgn

一个区域的句柄。

bRedraw

如果为 TRUE，则操作系统在设置区域之后重画窗口；否则，它不进行重画。通常，如果窗口可见，则将 bRedraws 设为 TRUE。如果被设为 TRUE，则系统向窗口发送 WM_WINDOWPOSCHANGING 和 WM_WINDOWPOSCHANGED 消息。

说明

调用这个成员函数以设置窗口的区域。

窗口区域的坐标是相对于窗口（而不是窗口客户区）的左上角。

在成功地调用 SetWindowRgn 之后，操作系统拥有区域句柄 hRgn 所指定的区域。操作系统不生成该区域的拷贝，因此不要使用此区域句柄进行其它函数调用，并且不要关闭这个区域句柄。

请参阅 ::SetWindowRgn, CWnd::GetWindowRgn

CWnd::SetWindowText

```
void SetWindowText( LPCTSTR lpszString );
```

参数

lpszString

指向一个 CString 对象或以 null 结尾的字符串，将被用作新的标题或控件文本。

说明

这个函数将窗口的标题设为指定的文本。如果窗口为一个控件，则将设置控件内的文本。
这个函数使一条 WM_SETTEXT 消息被发送到这个窗口。

请参阅 CWnd::GetWindowText, ::SetWindowText

CWnd::ShowCaret

void ShowCaret( );

说明

这个函数在屏幕上当前插字符位置显示插字符。一旦被显示，则插字符就开始自动闪烁。

ShowCaret 成员函数仅当插字符具有形状并且没有被连续隐藏两次或更多次时才显示它。

隐藏插字符是积累性的。如果 HideCaret 成员函数被调用了五次，则 ShowCaret 也必须被调用五次以显示插字符。

插字符是一种共享资源。窗口仅应在它拥有输入焦点或处于活动状态时才显示

插字符。

请参阅 CWnd::HideCaret, ::ShowCaret

CWnd::ShowOwnedPopups

```
void ShowOwnedPopups( BOOL bShow = TRUE );
```

参数

bShow

指定弹出窗口是要被显示还是隐藏。如果这个参数为 TRUE，则所有隐藏的弹出窗口将被显示。如果这个参数为 FALSE，则所有可见的弹出窗口将被隐藏。

说明

显示或隐藏这个窗口拥有的所有弹出窗口。

请参阅 ::ShowOwnedPopups

CWnd::ShowScrollBar

void ShowScrollBar(UINT *nBar*, BOOL *bShow* = TRUE);

参数

nBar

指定滚动条是一个控件还是窗口非客户区的一部分。如果它是非客户区的一部分，则 *nBar* 还指明了滚动条是水平的、垂直的，还是都有。它必须是下列值之一：

- SB_BOTH 指定了窗口的水平和垂直滚动条。
- SB_HORZ 指定了窗口的水平滚动条。
- SB_VERT 指定了窗口的垂直滚动条。

bShow

指定了 Windows 要显示还是隐藏滚动条。如果这个参数为 TRUE，则滚动条被显示；否则隐藏滚动条。

说明

这个函数显示或隐藏滚动条。

应用程序不应在处理滚动条通知消息的时候调用 ShowScrollBar 以隐藏滚动条。

请参阅 ::ShowScrollBar, CScrollBar::ShowScrollBar

CWnd::ShowWindow

BOOL ShowWindow(int *nCmdShow*);

返回值

如果窗口原来可见，则返回非零值；如果CWnd原来是隐藏的，则返回0。

参数

nCmdShow

指定了CWnd应如何被显示。它必须是下列值之一：

- SW_HIDE 隐藏窗口并将活动状态传递给其它窗口。
- SW_MINIMIZE 最小化窗口并激活系统列表中的顶层窗口。
- SW_RESTORE 激活并显示窗口。如果窗口是最小化或最大化的，Windows恢复其原来的大小和位置。
- SW_SHOW 激活窗口并以其当前的大小和位置显示。
- SW_SHOWMAXIMIZED 激活窗口并显示为最大化窗口。
- SW_SHOWMINIMIZED 激活窗口并显示为图标。
- SW_SHOWMINNOACTIVE 将窗口显示为图标。当前活动的窗口将

保持活动状态。

- SW_SHOWNA　按照当前状态显示窗口。当前活动的窗口将保持活动状态。
- SW_SHOWNOACTIVATE　按窗口最近的大小和位置显示。当前活动的窗口将保持活动状态。
- SW_SHOWNORMAL　激活并显示窗口。如果窗口是最小化或最大化的，则 Windows 恢复它原来的大小和位置。

说明

这个函数设置窗口的可视状态。

每个应用程序只应用 CWinApp::m_nCmdShow 为主窗口调用一次 ShowWindow。以后调用 ShowWindow 应该用上面列出的值来代替 CWinApp::m_nCmdShow 指定的值。

请参阅　::ShowWindow, CWnd::OnShowWindow, CWnd::ShowOwnedPopups

CWnd::SubclassDlgItem

BOOL SubclassDlgItem(UINT *nID*, CWnd* *pParent*);

返回值

如果函数成功，则返回非零值；否则返回0。

参数

nID

控件的ID。

pParent

控件的父窗口（通常是对话框）。

说明

调用这个成员函数以动态地子类化一个从对话框模板创建的控件，并将之与CWnd对象相连接。当控件被动态子类化时，窗口消息将通过CWnd的消息映射转发，并首先调用CWnd类的消息处理函数。被发往基类的消息将被传递到控件的缺省消息处理函数。

这个成员函数将一个Windows控件与CWnd对象连接，并替换控件的WndProc和AfxWndProc函数。这个函数在GetSuperWndProcAddr成员函数返回的位置中保存旧的WndProc地址。

请参阅 CWnd::GetSuperWndProcAddr, CWnd::DefWindowProc, CWnd::SubclassWindow, CWnd::Attach

CWnd::SubclassWindow

BOOL SubclassWindow(HWND *hWnd*);

返回值

如果函数成功，则返回非零值；否则返回0。

参数

hWnd

窗口句柄。

说明

调用这个成员函数以动态子类化一个窗口，并将它与这个 CWnd **对象相连接。当窗口被动态子类化时，窗口消息将通过** CWnd **的消息映射，并首先调用** CWnd **类中的消息处理函数。发送给基类的消息将被传递给窗口的缺省消息处理函数。**

这个成员函数将 Windows **控件与** CWnd **对象连接起来，并替换窗口的** WndProc **和** AfxWndProc **函数。这个函数在** CWnd **对象中保存旧的** WndProc **的指针。**

请参阅 CWnd::GetSuperWndProcAddr, CWnd::DefWindowProc,

CWnd::SubclassDlgItem, CWnd::Attach, CWnd::PreSubclassWindow, CWnd::UnsubclassWindow

CWnd::UnlockWindowUpdate

```
void CWnd::UnlockWindowUpdate();
```

说明

调用这个成员函数以解锁用 CWnd::LockWindowUpdate **锁定的窗口。**

在同一时刻只能用 LockWindowUpdate **锁定一个窗口。有关锁定窗口的更多信息参见** CWnd::LockWindowUpdate **或** Win32 **函数** LockWindowUpdate**。**

CWnd::UnsubclassWindow

```
HWND UnsubclassWindow( );
```

返回值

被取消子类化的窗口的句柄。

说明

调用这个成员函数以将 WndProc **设为原来的值，并将** HWND **标识窗口与** CWnd **对象分离。**

请参阅

CWnd::SubclassWindow,

CWnd::PreSubclassWindow,CWnd::GetSuperWndProcAddr,

CWnd::DefWindowProc, CWnd::SubclassDlgItem, CWnd::Attach

CWnd::UpdateData

BOOL UpdateData(BOOL *bSaveAndValidate* = TRUE);

返回值

如果操作成功，则返回非零值；否则返回 0**。如果** *bSaveAndValidate* **为** TRUE**，则返回非零值意味着已成功地使数据有效。**

参数

bSaveAndValidate

指明是要初始化对话框（FALSE）还是获取数据（TRUE）的标志。

说明

调用这个成员函数以初始化对话框中的数据，或者获得并检验对话框数据。

当一个模式对话框被创建时，框架自动在 CDialog::OnInitDialog 的缺省实现中调用 UpdateData，bSaveAndValidate 被设为 FALSE。这个函数在对话框可见之前被调用。CDialog::OnOK 的缺省实现令 bSaveAndValidate 为 TRUE 并调用这个成员函数以获得对话框中的数据，如果成功，将关闭对话框（如果在对话框中点击了 Cancel 按钮，则对话框将被关闭，并不获取数据）。

请参阅 CWnd::DoDataExchange

CWnd::UpdateDialogControls

void UpdateDialogControls(CCmdTarget* *pTarget*, BOOL *bDisableIfNoHndler*);

参数

pTarget

指向应用程序的主框架窗口，被用于转发更新消息。

bDisableIfNoHndler

指明没有更新处理函数的控件是否要被自动显示为禁止状态的标志。

说明

调用这个成员函数以更新对话框按钮和对话框或使用 ON_UPDATE_COMMAND_UI 机制的窗口中其它控件的状态。

如果子控件不具有处理函数，并且 bDisableIfNoHndler 为 TRUE，则子控件将被禁止。

框架在应用程序的空闲处理中为对话框条或工具条上的控件调用这个成员函数。

请参阅 CFrameW nd::m_bAutoMenuEnable

CWnd::UpdateWindow

void UpdateW indow( );

说明

如果更新区域不为空，则发送一条 WM_PAINT 消息以更新客户区域。UpdateWindow 成员函数直接发送一条 WM_PAINT 消息，越过应用程序队列。如果更新区域为空，则 WM_PAINT 不会被发送。

请参阅 ::UpdateWindow, CWnd::RedrawWindow

CWnd::ValidateRect

void ValidateRect(LPCRECT *lpRect*);

参数

lpRect

指向一个 CRect 对象或 RECT 结构，其中包含了要从更新区域中清除的矩形的客户坐标。如果 *lpRect* 为 NULL，则整个窗口都变为有效。

说明

这个函数从窗口的更新区域内清除给定的矩形，使给定矩形之内的客户区有效。BeginPaint 成员函数自动使整个客户区有效。如果在下一个 WM_PAINT 消息产生之前需要使一部分更新区域有效，则不应调用 ValidateRect 和 ValidateRgn 成员函数。

Windows 将继续产生 WM_PAINT 消息，直到当前的更新区域有效。

请参阅 CWnd::BeginPaint, ::ValidateRect, CWnd::ValidateRgn

CWnd::ValidateRgn

void ValidateRgn(CRgn* *pRgn*);

参数

pRgn

指向一个 CRgn 对象的指针，它标识了要从更新区域内清除的区域。如果这个参数为 NULL，则整个客户区都将被清除。

说明

这个函数从窗口的当前更新区域中清除给定的区域，使区域内的客户区有效。给定的区域必须是以前通过区域函数创建的。区域坐标被假设为使用客户区坐标。

BeginPaint 成员函数自动使整个客户区有效。如果在产生下一个 WM_PAINT 消息之前需要使更新区域的一部分有效,则不能调用 ValidateRect 和 ValidateRgn 成员函数。

请参阅 ::ValidateRgn, CWnd::ValidateRect

CWnd::WindowFromPoint

static CWnd* PASCAL WindowFromPoint(POINT *point*);

返回值

指向窗口对象的指针，点位于该窗口之内。如果在给定点不存在任何窗口，则返回NULL。返回的指针可能是临时的，不应被保存以供将来使用。

参数

point

指定了一个CPoint对象或POINT数据结构，它定义了要检查的点。

说明

这个函数获得包含指定点的窗口；point必须指定一个屏幕上用屏幕坐标表示的点。

WindowFromPoint不获取隐藏或禁止的窗口，即使该点位于窗口之内。应用程序应当使用ChildWindowFromPoint成员函数以实现没有限制的搜索。

请参阅 ::WindowFromPoint, CWnd::ChildWindowFromPoint

CWnd::WindowProc

virtual LRESULT WindowProc(UINT *message*, WPARAM *wParam*, LPARAM *lParam*);

返回值

返回值依赖于消息。

参数

message
　　指定了要处理的 Windows **消息。**

wParam
　　提供了可用于消息处理的附加信息。这个参数的值与消息有关。

lParam
　　提供了可用于消息处理的附加信息。这个参数的值与消息有关。

说明

这个函数为 CWnd **对象提供了** Windows **过程（**WindowProc**）。它通过窗口的消息映射分派消息。**

数据成员

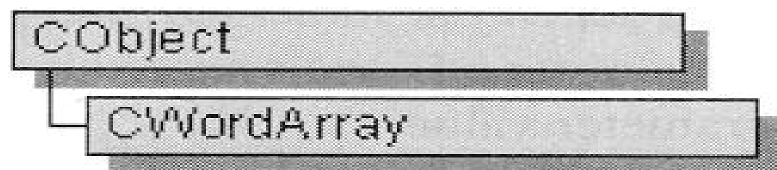
CWnd::m_hWnd

说明

与 CWnd 相连接的 Windows 窗口的句柄。m_hWnd 数据成员是一个 HWND 类型的公有变量。

请参阅 CWnd::Attach, CWnd::Detach, CWnd::FromHandle

CwordArray



CWordArray 类支持 16 位字的数组。

CWordArray 的成员函数与 CObArray 类的成员函数类似。因为这种相似性，你可以利用 CObArray 的参考文档找到成员函数的说明。每当你看到 CObject 指针被用作函数参数或返回值时，就用 WORD 来替换。

CObject* CObArray::GetAt(int <nIndex>) const;

例如，被转换为

WORD CWordArray::GetAt(int <nIndex>) const;

CWordArray 合并了 IMPLEMENT_SERIAL 宏以支持它的元素的串行化和转储。如果字数组被保存到档案中，可以通过重载的插入操作符或使用 CObject::Serialize 成员函数，则每个元素将依次被串行化。

注意 在使用数组之前，应使用 SetSize 来设置它的大小并为之分配内

存。如果你没有使用 SetSize，则在数组中加入元素时会引起频繁的重分配和拷贝。频繁的重分配和拷贝是效率低下的，会引起内存碎片。

如果你需要转储数组中的单个元素，则必须将转储环境的深度设为 1 或更大。

有关使用 CWordArray 的更多信息参见联机的《Visual C++ 程序员指南》中的文章“集合”。

#include <afxcoll.h>

CWordArray 类成员

构造

CWordArray	构造一个空的字数组

界限

GetSize	获得数组中元素的数目
GetUpperBound	返回最大的有效索引
SetSize	设置数组中将包含的元素的数目

操作

FreeExtra	释放在当前上界之上的没有使用的内存

续表

RemoveAll	从数组中清除所有的元素
访问元素	
GetAt	返回给定索引处的值
SetAt	为给定的索引设置值；数组不允许增长
ElementAt	返回对数组中元素指针的临时引用
GetData	允许访问数组中的元素。可以为 NULL
增长数组	
SetAtGrow	设置给定索引处的值；如果有必要则增长数组
Add	在数组的末尾加入元素；如果有必要则增长数组
Append	将一个数组追加在另一个数组后面；如果有必要则增长数组
Copy	将一个数组拷贝到另一个数组。如果有必要则增长数组
插入/清除	
InsertAt	在指定索引处插入一个元素（或其它数组中的全部元素）
RemoveAt	清除指定索引处的元素
操作符	
Operator [ ]	设置或获得指定索引处的元素

MFC 中的宏和全局函数、变量

微软基础类库可以分成两个主要部分：（1）MFC 类，（2）宏和全局函数、变量。如果一个函数或者变量不是某个类的成员，它就是全局的函数或变量。

MFC 类库和活动模板库（ATL）共用一些字符串转换宏。参看 ATL 文档中的“字符串转换宏”部分，其中包含了对这些宏的讨论。

MFC 中的宏和全局函数、变量提供了下列方面的功能：

General MFC

- 数据类型
- MFC 类对象的强制类型转换
- 运行时对象模型服务
- 诊断服务
- 异常处理
- 字符串格式化和消息框显示
- 消息映射
- 应用程序信息和管理

- 标准命令和窗口 ID
- 集合类帮助函数
- ClassWizard 注释限定符

Database

- 应用于 MFC 中 ODBC 类的记录字段交换（RFX）函数和成组记录字段交换（Bulk RFX）函数。
- 应用于 MFC 中 DAO 类的记录字段交换(DFX)函数。
- 应用于 CRecordView 和 CDaoRecordView（MFC 中的 ODBC 和 DAO 类）的对话框数据交换（DDX）函数。
- 应用于 OLE 控件的对话框数据交换（DDX）函数。
- 用于辅助直接调用开放数据库连接（ODBC）API 函数的宏和全局函数、变量。
- DAO 数据库引擎的初始化和结束函数。

Internet

- Internet 服务器 API（ISAPI）解析映射。
- Internet URL 解析全局函数。
- Internet 服务器 API（ISAPI）诊断宏。

OLE

- OLE 初始化。
- 应用程序控制。
- 调度映射。

另外，MFC 提供了一个函数，名为 AfxEnableControlContainer，它使任何一个用 MFC4.0 开发的 OLE 容器都能够完全支持复合 OLE 控件。

OLE 控件

- 可变参数类型常量。
- 类型库访问。
- 属性页。
- 事件映射。
- 事件接收映射。
- 连接映射。
- 注册 OLE 控件。
- 类工厂和注册。
- OLE 控件的持续性。

本章的第一部分简要地讨论了上面的每一个方面，列出了该方面的每个全局函数、变量和宏，同时还简要地描述了它们的功能。随后，按照字母顺序详细讨

论了 MFC 类库中的全局函数、全局变量和宏。

有关 MFC 中的宏和全局函数、变量的主要参考材料是《Visual C++程序员指南》。通常当你需要寻找有关宏和全局函数、变量的信息的时候，你首先会去查询这个地方。如果有必要，Visual C++ 程序员联机指南在描述函数或者宏的时候会提及适当的文章。

> **注意** 许多全局函数以前缀“Afx”开始——有一些例外，比如对话框数据交换函数（DDX）和一些数据库函数不符合这个约定。所有的全局变量都以“Afx”为前缀。宏没有什么特定的前缀，但是它们都是以大写形式出现的。

数据类型

本章列出了微软基础类库中最常见的数据类型。大多数数据类型是与 Windows 软件开发工具包（SDK）中一致的，也有一些是 MFC 独有的。

下面的数据类型是 Windows SDK 和 MFC 共用的：

- BOOL 布尔值。
- BSTR 32 位字符指针。
- BYTE 8 位无符号整数。

- COLORREF 用作颜色值的 32 位值。
- DWORD 32 位无符号整数，或者是段地址以及与之相关的偏移量。
- LONG 32 位带符号整数。
- LPARAM 32 位值，作为参数传递给一个窗口过程或者回调函数。
- LPCSTR 指向字符串常量的 32 位指针。
- LPSTR 指向字符串的 32 位指针。
- LPCTSTR 指向一个兼容 Unicode 和 DBCS 的字符串的 32 位指针。
- LPTSTR 指向一个兼容 Unicode 和 DBCS 的字符串的 32 位指针。
- LPVOID 指向一个未指定类型的 32 位指针。
- LRESULT 窗口过程或者回调函数返回的 32 位值。
- UINT 在 Windows 3.0 和 3.1 中表示 16 位的无符号整数，在 Win32 中表示 32 位的无符号整数。
- WNDPROC 指向一个窗口过程的 32 位指针。
- WORD 16 位无符号整数。
- WPARAM 作为参数传递给窗口函数或者回调函数的值：在 Windows 3.0 和 3.1 中为 16 位，在 Win32 中为 32 位。

微软基础类库中独有的数据类型如下：

- POSITION 用于标记集合中一个元素的位置的值；被 MFC 中的集合类所使用。
- LPCRECT 指向一个 RECT 结构体常量（不能修改）的 32 位指针。

在《Win32 SDK 程序员参考》的“数据类型”部分，有个不太通用的数据类型的列表。

MFC 类对象的强制类型转换

强制类型转换宏提供了一种方式，可以将一个给定的指针转换为指向特定类的对象的指针，并不需要检查这种转换是否合法。

下面的表格列出了 MFC 中的强制类型转换宏：

将指针强制转换为 MFC 类的对象的宏	
DYNAMIC_DOWNCAST	将一个指针转换为指向一个对象的指针，同时检查这种转换是否合法
STATIC_DOWNCAST	将指向一个类的对象的指针转换为另一个相关类型的指针。在调试版本中，如果该对象不能转换为目标类型，会引起 ASSERT

运行时对象模型服务

CObject和CRuntimeClass这两种类封装了一些对象服务，包括对运行时对象信息的访问，串行化以及动态对象创建。所有从CObject继承的类都继承了这些功能。

通过访问运行时对象信息，你可以在运行时决定对象的信息。当你需要额外的函数参数类型检查，或者你需要编写与类和对象有关的用于特定目的的代码时，这种在运行时决定对象信息的能力非常有用。C++语言本身并不直接支持运行时对象信息。

串行化是指向一个文件写入或者从一个文件读出对象的内容的过程。通过串行化，你即使在应用程序退出以后也还可以保存对象的信息。当应用程序重新启动以后，你可以从文件中读入对象的内容。这样的数据对象被称为是“永久性”的。

动态对象创建功能使你能够在运行时创建指定类型的对象。例如，文档、视和框架对象必须支持动态创建，因为应用框架必须动态地创建它们。

下面的表格列出了支持运行时类信息、串行化和动态创建的MFC宏。

如果需要关于运行时对象服务和串行化的更详细的信息，可以参考《Visual C++程序员指南》中的“CObject类：访问运行时类信息”部分。

运行时对象模式服务宏	
DECLARE_DYNAMIC	允许对运行时类信息进行访问（必须在类定义中使用）
DECLARE_DYNCREATE	允许动态创建和访问运行时类信息（必须在类定义中使用）
DECLARE_SERIAL	允许串行化和访问运行时类信息（必须在类定义中使用）
IMPLEMENT_DYNAMIC	允许对运行时类信息进行访问（必须在类的实现中使用）
IMPLEMENT_DYNCREATE	允许动态创建和访问运行时类信息（必须在类的实现中使用）
IMPLEMENT_SERIAL	允许串行化和访问运行时类信息（必须在类的实现中使用）
RUNTIME_CLASS	返回与指定名字的类对应的 CRuntimeClass 结构

OLE 经常需要在运行时动态地创建对象。例如，一个 OLE 服务器必须能够在响应客户的请求时动态地创建 OLE 项目。类似地，一个自动化服务器也必须能够在响应自动化客户的请求时动态地创建项目。

微软基础类库特别为 OLE 提供了两个宏。

OLE Objects 的动态创建	
DELCARE_OLECREATE	允许通过 OLE 自动化创建对象
IMPLEMENT_OLECREATE	允许由 OLE 系统创建对象

诊断服务

微软基础类库支持许多诊断服务功能，这会使你的调试工作更加简单。这些诊断服务包括一些宏和全局函数，它们允许你跟踪程序的内存分配，在运行时转储对象的内容及在运行时打印调试信息。这些用于诊断服务的宏和全局函数可以组成以下几个类别：

- 通用诊断宏。
- 通用诊断函数和变量。
- 对象诊断函数。

在 MFC 的调试和发行版本中，所有从 CObject 继承的类都可以使用这些宏和函数。但是，除了 DEBUG_NEW 和 VERIFY 以外，其他的在发行版本中都不起任何作用。

在调试版本的类库中，所有被分配的内存块都包含一部分“保护字节”。如果

这些字节被非正常的内存读写所扰乱，诊断例程就会报告出问题。如果你在程序文件中包含了如下的程序行：

```
#define new DEBUG_NEW
```

所有对 new 的调用将保存了产生内存分配的文件名和行号。函数 CMemoryState::Dump- AllObjectsSince 将会显示这些附加信息，使你能够识别内存泄漏。同时还可以参考 CDumpContext 类在诊断输出中给出的附加信息。

另外，C 运行时库同样支持一个诊断函数集，你可以利用它来调试程序。如果需要获得更多的信息，请参阅《Microsoft Visual C++ 6.0 库参考》中的《Microsoft Visual C++ 6.0 运行库参考》一卷中的“诊断例程”一文。

MFC 通用调试宏	
ASSERT	在调试版本的库中，如果指定的表达式计算结果为 FALSE，则打印出一条消息，然后退出程序
ASSERT_KINDOF	测试一个对象是否属于一个指定的类或者从指定类继承而来的类
ASSERT_VALID	调用对象的 AssertValid 成员函数测试对象的内部完整性；通常由 CObject 继承而来
DEBUG_NEW	在调试模式下提供所有对象分配的文件名和行号以帮助发现内存泄漏
TRACE	在类库的调试版本中提供了类似 printf 的功能
TRACE0	与 TRACE 类似，但是接受的格式字符串不包括参数

续表

TRACE1	与 TRACE 类似，但是接受的格式字符串只包括一个参数
TRACE2	与 TRACE 类似，但是接受的格式字符串只包括两个参数
TRACE3	与 TRACE 类似，但是接受的格式字符串只包括三个参数
VERIFY	与 ASSERT 类似，但是在 Release 版本中也象 Debug 版本一样计算表达式的值

MFC 通用诊断变量和函数

afxDump	全局变量，将 CdumpContext 信息发送的调试器输出窗口或者调试终端
afxMemDF	全局变量，控制着调试内存分配器的特性
afxTraceEnabled	全局变量，用于开放或者禁止 TRACE 宏的输出
afxTraceFlags	全局变量，用于打开 MFC 内建的报告特性
AfxCheckError	全局变量，用于测试传送的 SCODE，检查是否是错误，如果是，抛出适当的错误
AfxCheckMemory	检查当前分配的所有内存的完整性
AfxDump	如果在调试器内调用，则在调试时转储对象的状态

续表

AfxDumpStack	生成当前栈的一个图象。这个函数通常被静态连接
AfxEnableMemoryTracking	打开或关闭内存跟踪
AfxIsMemoryBlock	检验一个内存块是否被正确地分配
AfxIsValidAddress	检验一个内存地址是否属于程序的地址范围
AfxIsValidString	检验一个指向字符串的指针是否有效
AfxSetAllocHook	允许在每次进行内存分配时调用一个函数
MFC 对象诊断函数	
AfxDoForAllClasses	对所有从 CObject 继承的支持运行时类型检查的类执行一个指定的功能
AfxDoForAllObjects	对所有从 CObject 继承的用 new 分配内存的对象执行一个指定的功能

异常处理

在程序执行的过程中，可能会发生一些称为“异常”的非正常状态或者错误。这些异常可能包括内存耗尽，资源分配错误或是找不到文件等。

微软基础类库中采用的异常处理模式和 ANSI 标准化委员会建议的 C++异常处

理方式很接近。异常处理函数必须在调用可能发生不正常状态的函数之前建立。如果这个函数遇到了不正常的状态，它就抛出一个异常，并且将控制权转移给异常处理函数。

微软基础类库中包含的某些宏可以建立异常处理函数。另有一些全局函数有助于抛出异常并在必要时终止程序。这些宏和全局函数可以分为以下几类：

- 异常宏，生成异常处理函数的结构。
- 异常抛出函数，生成特定类型的异常。
- 终止函数，使程序终止。

如果需要示例和更多的细节，请参阅《Visual C++程序员指南》中的“异常”部分。

异常宏	
TRY	声明一段代码为异常处理
CATCH	声明一段代码，用于捕捉前面的 TRY 块产生的一个异常
CATCH_ALL	声明一段代码，用于捕捉前面的 TRY 块产生的所有异常
AND_CATCH	声明一段代码，用于捕捉前面的 TRY 块产生的其他类型的异常
AND_CATCH_ALL	声明一段代码，用于捕捉前面的 TRY 块抛出的所有其他类型的异常

续表

END_CATCH	结束上一个 CATCH 或 AND_CATCH 块
END_CATCH_ALL	结束上一个 CATCH_ALL 代码块
THROW	抛出一个指定的异常
THROW_LAST	抛出当前处理的异常，交给后面的处理函数

异常抛出函数	
AfxThrowArchiveException	抛出一个档案异常
AfxThrowFileException	抛出一个文件异常
AfxThrowMemoryException	抛出一个内存异常
AfxThrowNotSupportedException	抛出一个不支持的异常
AfxThrowResourceException	抛出一个 Windows 的未找到资源异常
AfxThrowUserException	在用户初始化的程序动作中抛出一个异常

MFC 特别为 OLE 异常提供了两个异常抛出函数。

OLE 异常函数	
AfxThrowOleDispatchException	在 OLE 自动化函数内抛出一个异常
AfxThrowOleException	抛出一个 OLE 异常

为了支持数据库异常，数据库类提供了两个异常类，CDBException 和 CDaoException，还有一些全局函数用于支持异常类型：

DAO 异常函数	
AfxThrowDAOException	从你自己的代码中抛出一个 CDaoException 异常
AfxThrowDBException	从你自己的代码中抛出一个 CDBException 异常

MFC 提供了下列终止函数：

终止函数	
AfxAbort	当发生了致命错误时用于结束应用程序

请参阅 CException

CString 格式化和消息框显示

有一些函数被用于格式化和处理 CString 对象。不论何时你需要操作 CString 对象，你都可以利用这些函数，但是在格式化将被显示在消息框中的文本时，这些函数特别有用。

这组函数中同时还包括了用于显示消息框的全局例程。

CString 函数	
AfxFormatString1	用一个字符串替换给定字符串中的格式字符“%1”

续表

AfxFormatString2	用两个字符串替换给定字符串中的两个格式字符“%1”和“%2”
AfxMessageBox	显示一个消息框

请参阅 CString

应用程序信息和管理

当你编写一个应用程序的时候，要创建一个从 CWinApp 继承而来的对象。此时，可能你希望从这个对象的外部获取它的有关信息。

微软基础类库提供了下列全局函数，用以帮助你完成这些任务：

应用程序信息和管理函数	
AfxFreeLibrary	减少已调入内存的动态链接库（DLL）模块的引用计数，当引用计数减到 0 时，这个模块就会被释放
AfxGetApp	返回应用程序中唯一的 CWinApp 对象的指针
AfxGetAppName	返回一个包含应用程序名字的字符串
AfxGetInstanceHandle	返回一个代表应用程序实例的 HINSTANCE 变量

续表

AfxGetMainWnd	返回指向非 OLE 应用程序的当前主窗口的指针，或者是服务器程序的现场框架窗口
AfxGetResourceHandle	返回代表应用程序的缺省资源的 HINSTANCE 变量。可以利用它来直接访问应用程序的资源
AfxInitRichEdit	为应用程序初始化 RichEdit 控件，如果公共控件库还没有被初始化，则初始化公共控件库
AfxLoadLibrary	调入一个 DLL 模块，同时返回一个句柄，可以通过它来得到 DLL 函数的地址
AfxRegisterWndClass	注册一个 Windows 的窗口类，用它来替换 MFC 自动注册的窗口类
AfxSocketInit	在 CwinApp::InitInstance 的重载函数中调用，用它来初始化 Windows 的 Socket 协议
AfxSetResourceHandle	设置指向应用程序调入的缺省资源的句柄
AfxRegisterClass	在使用 MFC 的 DLL 中注册一个窗口类
AfxBeginThread	创建一个新的线程
AfxEndThread	结束当前线程
AfxGetThread	获得指向当前 CwinThread 对象的指针
AfxWinInit	由 MFC 提供的 WinMain 函数调用，是基于 GUI 的应用程序中 CwinApp 初始化的一部分，用于初始化 MFC。在使用 MFC 的控制台程序中必须直接调用

请参阅 CwinApp

标准命令和窗口 Ids

微软基础类库在 AFXRES.H 中定义了一系列标准的命令和窗口 ID。这些 ID 通常在资源编辑器中使用，ClassWizard 也使用它们来将消息映射到你的处理函数。所有的标准命令都带有 ID_前缀。例如，当你使用菜单编辑器的时候，你通常会将文件菜单中的打开菜单项与标准的 ID_FILE_OPEN 命令 ID 绑定在一起。

对于大多数标准命令，应用程序不需要引用其命令 ID，因为应用框架本身在它的框架类(CwinThread, CwinApp, Cview, Cdocument 等等)的消息映射中处理了这些命令。

除了这些标准的命令 ID 以外，还定义了一些以 AFX_ID 为前缀的标准 ID。这些 ID 包括标准的窗口 ID（以 AFX_IDW 为前缀），字符串 ID（以 AFX_IDS_为前缀），以及其他几种类型。

以 AFX_ID 为前缀的 ID 很少由程序员使用，但是当你需要重载那些引用了 AFX_ID 的框架函数的时候，你可能需要引用这些 ID。

这儿并不单独描述这些 ID。在技术注释文档的第 20，21 和 22 篇中，你可以找

到更详细的有关信息。

注意　FXWIN.H 中并不直接包含头文件 AFXRES.H。你必须在你的应用程序的资源描述文件（.RC）中明确地包含下列语句：

```
#include afxres.h
```

集合类帮助函数

集合类 Cmap，Clist 以及 Carray 使用了一些全局的模板函数来完成诸如构造、析构和串行化元素等功能。在你实现由 Cmap，Clist 以及 Carray 继承而来的类时，你必须重载这些函数以适应在你的映射、链表、数组中保存的数据类型。如果需要获得有关重载 ConstructElements，DestructElements 以及 SerializeElements 的更多信息，请参阅下述文档《Visual C++ 程序员指南》中的“如何生成类型安全的集合”。

微软基础类库中提供了下列全局函数来帮助你定制集合类：

集合类帮助函数	
CompareElements	指出元素是否相同
ConstructElements	当生成一个元素时完成必要的动作
CopyElements	将元素从一个数组中复制到另一个数。
DestructElements	当销毁一个元素时完成必要的动作

续表

DumpElements	提供了面向流的诊断输出
HashKey	计算一个 Hash 键
SerializeElements	将元素保存在档案中，或者从档案中获得元素

请参阅　CMap, CList, CArray

记录字段交换函数

这个主题列出了记录字段交换函数(RFX, Bulk RFX, 以及 DFX)，它们被用于自动完成在记录集对象和它的数据源之间的数据传送，同时还能执行对数据的一些操作。

如果你正在使用基于 ODBC 的类，并且你已经实现了多行记录成组获取，你一定手动重载了 CRecordset 的成员函数 DoBulkFieldExchange，在其中对与数据源中的每一个列相对应的数据成员调用了成组 RFX 函数。

如果你的基于 ODBC 的类中还没有实现多行记录成组获取，或者你使用的是基于 DAO 的类，那么 ClassWizard 将会重载 CRecordset 或 CDaoRecordset 的成员函数 DoFieldExchange，在其中对你的记录集中的每个数据字段成员调用 RFX 函数（如果是 ODBC 类）或者 DFX 函数（如果是 DAO 类）。

当框架每一次调用DoFieldExchange或者DoBulkFieldExchange的时候，记录字段交换函数就传送一次数据。每个函数传送一种特定类型的数据。

如果需要获得有关如何使用这些函数的更详细的说明，可以参阅“记录字段交换：RFX如何工作”；或者“DAO记录字段交换：DFX如何工作”。如果需要有关成组记录获取的更详细的信息，可以参阅下述文章：“成组记录获取（ODBC）”。这些文章可以在《Visual C++程序员指南》中找到。

对于动态绑定的数据列，你也可以自己调用RFX或DFX函数，而不是利用ClassWizard，在“Recordset：动态绑定数据列（ODBC）”和“DAO：动态绑定记录”中对此作了解释。这些文章可以在《Visual C++程序员指南》中找到。注意，在DAO中动态绑定数据与ODBC中的动态绑定有所不同。另外，你也可以编写自己的RFX或DFX例程，在技术注释文档的第43篇（针对ODBC）和第53篇（针对DAO）中对此作了解释。

如果需要获得在DoFieldExchange和DoBulkFieldExchange中使用RFX和DFX函数的例子，可以参看RFX_Text和RFX_Text_Bulk。DFX函数与RFX函数十分类似。

RFX 函数(ODBC)	
RFX_Binary	传送CByteArray类型的字节数
RFX_Bool	传送布尔数据
RFX_Byte	传送单个的字节数据

续表

RFX_Date	传送 CTime 或 TIMESTAMP_STRUCT 类型的时间和日期数据
RFX_Double	传送双精度浮点数据
RFX_Int	传送整型数据
RFX_Long	传送长整型数据
RFX_LongBinary	通过 CLongBinary 类的对象传送二进制大对象（BLOB）数据
RFX_Single	传送浮点数据
RFX_Text	传送字符串数据

Bulk RFX 函数（ODBC）	
RFX_Binary_Bulk	传送二进制数据的数组
RFX_Bool_Bulk	传送布尔数据的数组
RFX_Byte_Bulk	传送字节数据的数组
RFX_Date_Bulk	传送 TIMESTAMP_STRUCT 类型数据的数组
RFX_Double_Bulk	传送双精度浮点数据的数组
RFX_Int_Bulk	传送整型数据的数组
RFX_Long_Bulk	传送长整型数据的数组
RFX_Single_Bulk	传送浮点数据的数组
RFX_Text_Bulk	传送 LPSTR 类型数据的数组

DFX 函数(DAO)	
DFX_Binary	传送 CByteArray 类型的字节数组
DFX_Bool	传送布尔型数据
DFX_Byte	传送单个字节数据
DFX_Currency	传送 COleCurrency 类型的货币数据
DFX_DateTime	传送 COleDateTime 类型的时间和日期数据
DFX_Double	传送双精度浮点数据
DFX_Long	传送长整型数据
DFX_LongBinary	通过 CLongBinary 类的对象传送大二进制对象（BLOB）数据，对于 DAO，推荐你使用 DFX_Binary
DFX_Short	传送短整型数据
DFX_Single	传送浮点数据
DFX_Text	传送字符串数据。

请参阅

Recordset::DoFieldExchange, CRecordset::DoBulkFieldExchange,CDaoRecordset::DoFieldExchange

CRecordView 和 CDaoRecordView 的对话框数据交换函数

这个主题列出了用于在 CRecordset 和 CRecordView 表格之间或是 CDaoRecordset 与 CDaoRecordView 表格之间交换数据的 DDX_Field 函数。

重要 DDX_Field 函数与 DDX 函数类似，它们都是与表格中的控件交换数据。但是与 DDX 函数不同的是，它们与和视窗相关的记录集对象的字段交换数据，而不是与记录视窗本身的字段进行交换。如果要获得更多的信息，请参阅 CRecordView 和 CDaoRecordView 类，或者参考《Visual C++程序员指南》中的“ClassWizard：将表格控件映射到记录集的字段”。

DDX_Field 函数	
DDX_FieldCBIndex	在记录集字段的数据成员和 CRecordView 或 CDaoRecordView 中组合框的当前选择索引之间交换整数数据
DDX_FieldCBString	在记录集字段的数据成员和 CrecordView 或 CDaoRecordView 中组合框的编辑控件之间交换 CString 数据。当从记录集向控件传送数据的时候，这个函数将选择组合框中以给定字符串中的字符开头的项

续表

DDX_FieldCBStringExact	在记录集字段的数据成员和 CRecordView 或 CDaoRecordView 中组合框的编辑控件之间交换 CString 数据。当从记录集向控件传送数据的时候，这个函数将选择组合框中严格匹配给定字符串的项
DDX_FieldCheck	在记录集字段的数据成员和 CRecordView 或 CdaoRecordView 中复选框之间交换布尔型数据
DDX_FieldLBIndex	在记录集字段的数据成员和 CRecordView 或 CDaoRecordView 中列表框的当前选择索引之间交换整型数据
DDX_FieldLBString	管理在记录集的字段数据成员和列表框控件之间的 CString 数据的传送。当数据是从记录集传送到控件时，这个函数将选择列表框中以给定字符串中字符开头的项
DDX_FieldLBStringExact	管理在记录集的字段数据成员和列表框控件之间的 CString 数据的传送。当数据是从记录集传送到控件时，这个函数将选择列表框中严格匹配给定字符串的项
DDX_FieldRadio	在记录集的字段数据成员和 CRecordView or CDaoRecordView 中的一组单选控件之间传送整数数据

续表

DDX_FieldScroll	设置或获得 CrecordView or CDaoRecordView 中滚动条控件的位置。在你的 DoFieldExchange 函数中调用
DDX_FieldText	对于在记录集的字段数据成员和 CRecordView or CDaoRecordView.中的编辑框之间传送数据，可以获得对应各种类的重载函数，包括 int 整型，UINT，long 型，DWORD，CString，float，double，short，COleDateTime 以及 COleCurrency 等

OLE 控件中的对话框数据交换

这个主题列出了用于在对话框、表格视或控制视对象中的 OLE 控件的属性与对话框、表格视或控制视对象的数据成员之间交换数据的 DDX_OC 函数。

DDX_OC 函数	
DDX_OCBool	管理在 OLE 控件的属性与布尔型数据成员之间的布尔数据传送
DDX_OCBoolRO	管理在 OLE 控件的只读属性与布尔型数据成员之间的布尔数据传送
DDX_OCColor	管理在 OLE 控件的属性与 OLE_COLOR 数据成员之间的 OLE_COLOR 数据传送

续表

DDX_OCColorRO	管理在 OLE 控件的只读属性与 OLE_COLOR 数据成员之间的 OLE_COLOR 数据传送
DDX_OCFloat	管理在 OLE 控件的属性与 float 浮点（或 double 双精度）数据成员之间的浮点（或双精度）数据传送
DDX_OCFloatRO	管理在 OLE 控件的只读属性与浮点（或双精度）数据成员之间的浮点（或双精度）数据传送
DDX_OCInt	管理在 OLE 控件的属性与 int 整型（或 long 长整型）数据成员之间的整型（或长整型）数据传送
DDX_OCIntRO	管理在 OLE 控件的只读属性与整型（或长整型）数据成员之间的整型（或长整型）数据传送
DDX_OCShort	管理在 OLE 控件的属性与 short 短整型数据成员之间的短整型数据传送
DDX_OCShortRO	管理在 OLE 控件的只读属性与短整型数据成员之间的短整型数据传送
DDX_OCText	管理在 OLE 控件的属性与 CString 数据成员之间的 CString 数据传送
DDX_OCTextRO	管理在 OLE 控件的只读属性与 CString 数据成员之间的 Cstring 数据传送

数据库宏和全局函数、变量

这里列出的宏使用于基于 ODBC 的数据库应用程序。它们不应在基于 DAO 的应用程序中使用。

直到 MFC4.2 之前，AFX_SQL_ASYNC 和 AFX_SQL_SYNC 宏使异步操作有机会向其它操作让出时间。从 MFC4.2 开始，这些宏的实现方式有了变化，因为现在 MFC 的 ODBC 类只使用同步操作。MFC4.2 中的 AFX_ODBC_CALL 宏是新的。

EDDX_OC 函数

数据库宏	
AFX_ODBC_CALL	利用这个宏来调用一个返回 SQL_STILL_EXECUTING 的 ODBC API，它会反复调用这个函数直到它不再返回 SQL_STILL_EXECUTING
AFX_SQL_ASYNC	仅仅是简单地调用 AFX_ODBC_CALL
AFX_SQL_SYNC	利用这个宏来调用一个不返回 SQL_STILL_EXECUTING 的 ODBC API

数据库全局函数

AfxGetHENV	利用这个全局函数来获得 MFC 当前使用的 ODBC 环境的句柄。你可以在直接 ODBC 调用时使用这个句柄

DAO 数据库引擎的初始化和终止

当你使用 MFC 的 DAO 对象时，必须首先初始化 DAO 数据库引擎，在你的应用程序或者 DLL 退出前，必须终止数据库引擎。有两个函数 AfxDaoInit 和 AfxDaoTerm 可以完成这个任务。

DAO 数据库引擎初始化和终止函数

AfxDaoInit	初始化 DAO 数据库引擎
AfxDaoTerm	终止 DAO 数据库引擎

OLE 初始化

在你的应用程序可以使用 OLE 系统服务之前，必须首先初始化 OLE 系统 DLL，并且确定 DLL 的版本。AfxOleInit 可以初始化 OLE 系统 DLL。

OLE 初始化	
AfxOleInit	初始化 OLE 库

应用程序控制

OLE 需要实质性地控制应用程序和它的对象。OLE 的系统 DLL 必须能够自动地启动、释放应用程序，协调它们的产物以及对象的修改，等等。这个主题中的函数适应了这些需要。另外，除了被 OLE 系统 DLL 调用以外，有时这些函数必须由应用程序自己调用。

应用程序控制	
AfxOleCanExitApp	指出应用程序是否能够结束
AfxOleGetMessageFilter	获得应用程序当前的消息过滤器
AfxOleGetUserCtrl	获得当前的用户控制标志
AfxOleSetUserCtrl	设置或清除用户控制标志
AfxOleLockApp	增加应用程序中活动对象的全局计数
AfxOleUnlockApp	减小应用程序中活动对象的全局计数
AfxOleRegisterServerClass	在 OLE 系统注册表中注册一个服务器
AfxOleSetEditMenu	实现 typename Object 命令的用户接口

调度映射

OLE 自动化提供了在应用程序之间调用方法和访问属性的途径。由微软基础类库提供的调度这些请求的机制称为“调度映射”，它分配对象函数和属性的内部、外部名字，同时还分配属性本身和函数参数的数据类型。

调度映射	
DECLARE_DISPATCH_MAP	声明将使用一个调度映射来揭示一个类的方法和属性（必须用于类声明中）
BEGIN_DISPATCH_MAP	开始一个调度映射的定义
END_DISPATCH_MAP	结束一个调度映射的定义
DISP_FUNCTION	用于调度映射中，以定义一个 OLE 自动化函数
DISP_PROPERTY	定义一个 OLE 自动化属性
DISP_PROPERTY_EX	定义一个 OLE 自动化属性并且命名“get”和“set”函数
DISP_PROPERTY_NOTIFY	为一个 OLE 自动化属性定义通知消息
DISP_PROPERTY_PARAM	定义一个带参数的 OLE 自动化属性，并且命名了“get”和“set”函数
DISP_DEFVALUE	将一个现存的属性设置为一个对象的缺省值

Variant参数类型常量

这个主题列出了一些新的常量，它们指出了被设计与微软基础类库中 OLE 控件类一起使用的 variant 参数类型。

下面是类常量的一个列表：

Variant 数据常量

- VTS_COLOR 用于表示 RGB 颜色值的 32 位整数
- VTS_FONT 指向一个 OLE 字体对象的 IFontDisp 接口的指针
- VTS_HANDLE 一个 Windows 句柄值
- VTS_PICTURE 指向一个 OLE 图形对象的 IPictureDisp 接口的指针
- VTS_OPTEXCLUSIVE 一个 16 位值，用于成组使用的控件，比如单选按钮。这个类型告诉容器是否在一组控件中有一个具有 TRUE 值，而其它的都是 FALSE
- VTS_TRISTATE 一个 16 位带符号整数，用于可能具有三个可能值（选中、未选中、变灰）的属性，例如，复选框
- VTS_XPOS_HIMETRIC 一个 32 位无符号整数值，用于表示 x 轴上以 HIMETRIC 为单的位置值

- VTS_YPOS_HIMETRIC 一个 32 位的无符号整数值，用于表示 y 轴上以 HIMETRIC 为单位的位置值
- VTS_XPOS_PIXELS 一个 32 位无符号整数值，用于表示 x 轴上以像素为单位的位置值
- VTS_YPOS_PIXELS 一个 32 位无符号整数值，用于表示 y 轴上以像素为单位的位置值
- VTS_XSIZE_PIXELS 一个 32 位无符号整数值，用于表示以像素为单位的屏幕对象的宽度
- VTS_YSIZE_PIXELS 一个 32 位无符号整数值，用于表示以像素为单位的屏幕对象的高度
- VTS_XSIZE_HIMETRIC 一个 32 位无符号整数值，用于表示以 HIMETRIC 为单位的屏幕对象的宽度
- VTS_YSIZE_HIMETRIC 一个 32 位无符号整数值，用于表示以 HIMETRIC 为单位的屏幕对象的高度

注意 另外还为所有的 variant 类型定义了一些 variant 常量，除了 VTS_FONT 和 VTS_PICTURE，它们提供了一个指向 variant 数据常量的指针。这些常量以 VTS_*Pconstantname* 约定来命名。例如，VTS_PCOLOR 是一个指向 VTS_COLOR 常量的指针。

访问类型库

类型库揭示了 OLE 控件与其它具有 OLE 能力的应用程序之间的接口。如果 OLE 控件需要提供一个或多个接口，它必须具有一个类型库。

下面列出的宏使 OLE 控件能够提供对它自己的类型库的访问。

类型库访问	
DECLARE_OLETYPELIB	定义了 OLE 控件的 GetTypeLib 成员函数（必须在类定义中使用）
IMPLEMENT_OLETYPELIB	实现了 OLE 控件的 GetTypeLib 成员函数（必须在类实现中使用）

属性页

属性页用可定制的图形接口显示特定 OLE 控件的属性的当前值，通过支持基于对话框数据交换（DDX）的数据映射机制提供了对它们的显示和编辑手段。这种数据映射机制将属性页控件映射到 OLE 控件的每个属性。控件属性的值反映了属性页的状态或内容。属性页控件和属性之间的这种映射由在属性页的 DoDataExchange 成员函数中调用的 DDP_函数指定。下面是用于交换你的属性

页控件中输入的数据的 DDP_函数的列表：

属性页数据交换	
DDP_CBIndex	利用这个函数将组合框中被选中的字符串的索引和控件的属性连接起来
DDP_CBString	利用这个函数将组合框中被选中的字符串和控件的属性连接起来。选中的字符串可以是以与属性值相同的字母开始的，但是并不需要完全匹配
DDP_CBStringExact	利用这个函数将组合框中被选中的字符串与控件的属性连接起来。选中的字符串与属性的字符串值必须是严格匹配的
DDP_Check	利用这个函数将控件的属性页中的复选框与控件的属性连接起来
DDP_LBIndex	利用这个函数将列表框中被选中字符串的索引与控件的属性连接起来
DDP_LBString	利用这个函数将列表框中被选中字符串与控件的属性连接起来。选中的字符串可以是以与控件属性值相同的字母开始的，但是并不需要完全匹配
DDP_LBStringExact	利用这个函数将列表框中被选中的字符串与控件的属性连接起来。被选中的字符串和属性的字符串值必须严格地匹配
DDP_PostProcessing	利用这个函数来结束你的控件的属性值的传送

续表

DDP_Radio	利用这个函数将控件的属性页中的一组单选按钮与控件的属性连接起来
DDP_Text	利益这个函数将控件的属性页中的控件与控件的属性连接起来。这个函数处理一些不同类型的属性，比如双精度值，短整型值，BSTR 值以及长整型值等

如果需要获得有关 DoDataExchange 函数和属性页的详细信息，请参阅《Visual C++程序员指南》中的“ActiveX 控件：属性页”一文。

下面是用来为 OLE 控件创建和管理属性页的宏的列表：

属性页	
BEGIN_PROPPAGEIDS	开始一个属性页 ID 的列表
END_PROPPAGEIDS	结束一个属性页 ID 的列表
PROPPAGEID	声明一个控件类的属性页

事件映射

不论何时，如果一个控件想要通知它的容器，有些动作（取决于控件的设计者）已经发生（比如按键、鼠标点击，或者控件状态的变化等），它就调用一个事件引发的函数。这个函数通过产生相应的事件来通知控件的容器一些重要的动

作已经产生了。

微软基础类库提供了一种为事件引发而优化的编程模式。在这种模式中，使用了“事件映射”来指定对于一个特定的控件，哪个函数引发哪个事件。事件映射中为每个事件包含了一个宏。例如，一个引发点击事件的事件映射可能是这样的：

```
BEGIN_EVENT_MAP(CSampleCtrl, COleControl)
	//{{AFX_EVENT_MAP(CSampleCtrl)
	EVENT_STOCK_CLICK( )
	//}}AFX_EVENT_MAP
END_EVENT_MAP()
```

EVENT_STOCK_CLICK 宏指明这个控件每检测到一次鼠标点击就引发一个点击事件。如果要获得其它预定事件的详细列表，请参阅《Visual C++程序员指南》中的“ActiveX 控件事件”一文。还要一些宏可以用来指明自定义的事件。

尽管事件映射宏很重要，通常你并不会直接插入它们。这是因为当你使用事件映射将事件引发的函数与事件联系起来时，ClassWizard 会在你的源代码中自动地创建一个事件映射入口。不论何时你想编辑或者添加事件映射入口，你都可以使用 ClassWizard。

为了支持事件映射，MFC 提供了下列宏：

事件映射定义和分界

DECLARE_EVENT_MAP	定义一个事件映射，它将被用于一个类中，将事件映射到一个事件引发的函数（必须在类定义中使用）
BEGIN_EVENT_MAP	开始一个事件映射的定义（必须在类的实现中使用）
END_EVENT_MAP	结束一个事件映射的定义（必须在类的实现中使用）

事件映射宏

EVENT_CUSTOM	指明哪个事件引发函数将引发特定的事件
EVENT_CUSTOM_ID	指明哪个事件引发函数将引发特定的具有指定调度 ID 的事件

消息映射宏

ON_OLEVERB	指明 OLE 控件处理的一个自定义的动词
ON_STDOLEVERB	重载一个 OLE 控件的标准动词映射

事件接收映射

当一个嵌入的 OLE 控件引发一个事件时，该控件的容器通过一个 MFC 提供的称为“事件接收映射”的机制接收事件。这种事件接收映射为每个特定的事件分配处理函数，包括这些事件的参数。如果需要获得有关事件接收映射的更多的信息，请参阅《Visual C++程序员指南》中的“ActiveX 控件容器”一文。

事件接收映射	
BEGIN_EVENTSINK_MAP	开始一个事件接收映射的定义
DECLARE_EVENTSINK_MAP	定义一个事件接收映射
END_EVENTSINK_MAP	结束一个事件接收映射的定义
ON_EVENT	为特定的事件指定一个事件处理函数
ON_EVENT_RANGE	为一系列 OLE 控件引发的事件定义一个事件处理函数
ON_EVENT_REFLECT	在被控件的容器处理之前接收控件所引发的事件
ON_PROPNOTIFY	定义一个处理函数以处理 OLE 控件产生的属性通知
ON_PROPNOTIFY_RANGE	定义一个处理函数以处理一个 OLE 控件集合产生的属性通知

续表

ON_PROPNOTIFY_REFLECT	在被控件的容器处理之前接收控件发出的属性通知

连接映射

OLE 控件可以向别的应用程序提供接口。这些接口仅允许容器对控件进行访问。如果一个 OLE 控件希望访问其它 OLE 对象的外部接口必须建立一个连接点。这个连接点允许一个控件访问外部的调度映射，比如事件映射或通知函数。

微软基础类库提供了支持这种连接点的编程模式。在这种模式中，利用“连接映射”来为 OLE 控件指派接口（或者连接点）。连接映射中为每个连接点包含了一个宏。如果要获得有关连接映射的更多信息，参见 CConnectionPoint 类。

典型的情况是，一个控件将仅仅支持两个连接点：一个用于事件，一个用于属性通知。这是由 COleControl 基类实现的，并且不需要控件的编写者再作什么额外的工作。你希望在你的类中实现的任何附加的连接点都必须手动添加。为了支持连接映射和连接点，MFC 提供了下列宏：

连接映射定义和分界

BEGIN_CONNECTION_PART	声明一个嵌入类，它实现了附加的连接点（必须在类定义中使用）

续表

END_CONNECTION_PART	结束一个连接点的定义（必须在类定义中使用）
CONNECTION_IID	指定控件的连接点的接口 ID
DECLARE_CONNECTION_MAP	声明一个连接映射将被用在一个类中（必须在类定义中使用）
BEGIN_CONNECTION_MAP	开始一个连接映射的定义（必须在类实现中使用）
END_CONNECTION_MAP	结束一个连接映射的定义（必须在类实现中使用）
CONNECTION_PART	指定控件的连接映射中的连接点

下面的函数帮助建立和断开一个使用连接点的连接：

连接点的初始化/终止	
AfxConnectionAdvise	在源端和接收端之间建立一个连接
AfxConnectionUnadvise	中断源端和接收端之间的一个连接

注册 OLE 控件

OLE 控件与其它 OLE 服务器对象一样，可以被其它具有 OLE 能力的应用程序

所访问。这是通过注册控件的类型库和类来实现的。

下面的函数使你能够在 Windows 的注册数据库中增添或删除控件类，属性页以及类型库等：

注册 OLE 控件	
AfxOleRegisterControlClass	在注册数据库中增添控件类
AfxOleRegisterPropertyPageClass	在注册数据库中增添控件的属性页
AfxOleRegisterTypeLib	在注册数据库中增添控件的类型库
AfxOleUnregisterClass	从注册数据库中删除控件类或属性页类
AfxOleUnregisterTypeLib	从注册数据库中删除控件的类型库

AfxOleRegisterTypeLib 通常由控件的 DLL 中 DllRegisterServer 的实现部分来调用。类似地，AfxOleUnregisterTypeLib 由 DllUnregisterServer. 调用。AfxOleRegisterControlClass，AfxOleRegisterPropertyPageClass 以及 AfxOleUnregisterClass 通常由控件的类工厂或属性页的成员函数 UpdateRegistry 来调用。

类工厂与许可

为了创建你的 OLE 控件的一个实例，容器应用程序调用控件的类工厂的成员

函数。因为你的控件是一个实际的 OLE 对象，由类工厂负责创建你的控件的实例。每个 OLE 控件都必须具有一个类工厂。

OLE 控件的另一个重要特征就是它们的强迫许可的能力。ControlWizard 使你能够在创建控件的时候将许可功能合并进去。如果需要获得有关控件许可的更详细的信息，请参阅 Visual C++ 程序员联机指南中“ActiveX 控件：许可一个 ActiveX 控件”一文。

下面的表格列出了一些宏和函数，它们可以用来声明并实现你的控件的类工厂，也可以许可你的控件。

类工厂与许可	
DECLARE_OLECREATE_EX	为 OLE 控件或属性页声明一个类工厂
IMPLEMENT_OLECREATE_EX	实现控件的 GetClassID 函数并声明类工厂的一个实例
BEGIN_OLEFACTORY	开始许可函数的声明
END_OLEFACTORY	结束许可函数的声明
AfxVerifyLicFile	校验一个控件是否经许可在特定计算机上使用

OLE 控件的持久性

OLE 控件的一个能力即是它的属性的持久性（或是串行化），这个能力使得 OLE 控件可以对一个文件或流读、写属性值。容器应用程序可以利用串行化来存储控件的属性值，即使应用程序已经销毁了这个控件。当以后创建了控件的一个新的实例时，OLE 控件的属性值可以从文件或流中读取。

OLE 控件的持久性	
PX_Blob	交换控件的存储二进制大对象（BLOB）数据的属性
PX_Bool	交换控件的 BOOL 类型的属性
PX_Color	交换控件的颜色属性
PX_Currency	交换控件的 CY 类型的属性
PX_DataPath	交换控件的 CDataPathProperty 类型的属性
PX_Double	交换控件的 double 型的属性
PX_Font	交换控件的字体属性
PX_Float	交换控件的 float 类型的属性
PX_IUnknown	交换控件的未定义类型的属性
PX_Long	交换控件的 long 型属性
PX_Picture	交换控件的图形属性
PX_Short	交换控件的 short 型属性

续表

PX_Ulong	交换控件的 ULONG 类型的属性
PX_Ushort	交换控件的 USHORT 类型的属性
PX_String	交换控件的字符串属性
PX_VBXFontConver	在 VBX 控件的与字体有关的属性与 OLE 控件的字体属性之间交换

另外，全局函数 AfxOleTypeMatchGuid 可以测试给定的 GUID 与 TYPEDESC 是否匹配。

Internet 服务器 API 解析映射

Internet 服务器 API 是一个扩展的开放 API 集合，它向你提供了为你的 Microsoft Internet 信息服务器创建附加内容、运行 Internet 服务器程序的能力。当客户向 Internet 服务器发送一个请求时，服务器处理这个请求，将它发送给"解析映射"中的一系列分析宏。解析映射将客户的请求映射为一个从 ChttpServer 继承的类的函数和参数。

ISAPI 解析映射

BEGIN_PARSE_MAP	开始一个分析映射的定义
ON_PARSE_COMMAND	分析客户命令

续表

ON_PARSE_COMMAND_PARAMS	为 CHttpServer 对象定义一个来自客户的命令
DEFAULT_PARSE_COMMAND	调用由 FnName 参数指定的页面
END_PARSE_MAP	结束一个分析映射的定义

Internet URL 解析全局函数

Internet 服务器 API 是一个扩展的开放 API 集合，它向你提供了为你的 Microsoft Internet 信息服务器创建附加内容、运行 Internet 服务器程序的能力。当客户向 Internet 服务器发送一个请求时，你可以使用一些全局的 URL 处理函数来析取客户的信息。

Internet URL 解析全局函数	
AfxParseURL	分析一个 URL 字符串，返回服务的类型及其内容
AfxParseURLEx	分析一个 URL 字符串，返回服务的类型及其内容，同时还提供客户的名字和密码

Internet服务器API（ISAPI）诊断宏

微软的Internet信息服务器需要一些与MFC程序相同的诊断服务。但是，为Internet服务器编写的程序不需要MFC。下面描述的ISAPI宏为MFC程序和非MFC程序提供了相同层次的调试功能。

ISAPI 诊断宏	
ISAPIASSERT	提供了ASSERT功能
ISAPITRACE	提供了TRACE功能
ISAPITRACE0	提供了TRACE0功能
ISAPITRACE1	提供了TRACE1功能
ISAPITRACE2	提供了TRACE2功能
ISAPITRACE3	提供了TRACE3功能
ISAPIVERIFY	提供了VERIFY功能

宏、全局函数和全局变量

这个部分描述了 MFC 库中的全局函数、全局变量以及宏。

> **注意** 许多全局函数都以“Afx”前缀开始——除了一些例外，如对话框数据交换（DDX）函数和许多数据库函数以外，都遵从这个约定。所有的全局变量都以“Afx”前缀开始。宏没有任何特定的前缀，但是它们都是大写的。

MFC 库和活动模板库（ATL）共用一些字符串转换宏。参考 ATL 文档中的“字符串转换宏”，在那儿讨论了这些宏。

如果需要有关 C 运行库的调试版本以及诊断函数的信息，请参阅《Microsoft Visual C++6.0 库参考》的《Microsoft Visual C++6.0 运行库》一卷中的“调试例程”一文。

AfxAbort

```
void AfxAbort( );
```

说明

这是 MFC 提供的缺省的终止函数。AfxAbort 是当发生了一个致命错误，如无法处理的没有捕捉到的异常时，由 MFC 的成员函数调用的。在个别情况下，如果你遇到了无法恢复的灾难性错误，你也可以自己调用 AfxAbort。

AfxBeginThread

CWinThread* AfxBeginThread(AFX_THREADPROC *pfnThreadProc*, LPVOID *pParam*,
int *nPriority* = THREAD_PRIORITY_NORMAL, UINT *nStackSize* = 0, DWORD *dwCreateFlags* = 0, LPSECURITY_ATTRIBUTES *lpSecurityAttrs* = NULL);
CWinThread* AfxBeginThread(CRuntimeClass* *pThreadClass*,
int *nPriority* = THREAD_PRIORITY_NORMAL, UINT *nStackSize* = 0, DWORD *dwCreateFlags* = 0, LPSECURITY_ATTRIBUTES *lpSecurityAttrs* = NULL);

返回值

指向新创建的线程对象的指针。

参数

pfnThreadProc

指向工作线程的控制函数。不能是 NULL。这个函数必须按下面的方式定义：

```
UINT MyControllingFunction( LPVOID pParam );
```

pThreadClass

从 CWinThread 继承的对象的 RUNTIME_CLASS。

pParam

将要传送给控制函数的参数，如 *pfnThreadProc* 中定义的函数参数所示。

nPriority

设定的线程的优先级。如果为 0，则使用与创建它线程相同的优先级。在《Win32 程序员参考》的“SetThreadPriority”中有可用的优先级的完整列表和描述。

nStackSize

指定新线程使用的栈的以字节为单位的大小。如果为 0，则缺省的栈大小与创建它的线程的栈大小相同。

dwCreateFlags

指定控制线程的创建过程的附加标志。这个标志可以是两个值之一：

- CREATE_SUSPENDED 经过一个延迟后启动这个线程。这个线程将在调用 ResumeThread 以后才会启动。
- 0 创建后立即启动这个线程。

lpSecurityAttrs

指向一个 SECURITY_ATTRIBUTES 结构，它指定了线程的安全特性。如果为空，将使用与创建它的线程相同的安全特性。如果需要获得有关这个结构的详细信息，请参阅《Win32 程序员参考手册》。

说明

调用这个函数以创建一个新的线程。AfxBeginThread 的第一种形式创建了一个工作线程。第二种形式创建了一个用户界面线程。

AfxBeginThread 创建一个新的 CWinThread 对象，调用它的 CreateThread 函数以启动这个线程，并且返回这个线程的指针。整个过程都进行检查以保证如果创建失败，所有的对象都会被适当地释放。为了结束这个线程，可以在线程内调用 AfxEndThread，或者从工作线程的控制函数内返回。

关于 AfxBeginThread 的更多信息，请参阅《Visual C++程序员指南》中的“多线程：创建工作线程”和“多线程：创建用户界面线程”。

请参阅 AfxGetThread

AfxCheckError

```
void AFXAPI AfxCheckError(SCODE sc);
        throw CMemoryExcetion *
        throw COleException *
```

说明

这个函数检测传递的 SCODE 是否是个错误。如果它是个错误，这个函数将抛出一个异常。如果传递的 SCODE 是个 E_OUTOFMEMORY，它就调用 AfxThrowMemoryException 抛出一个 CMemoryException。否则，它调用 AfxThrowOleException 抛出一个 COleException。

这个函数可以用来检查你的应用程序中对 OLE 函数调用的返回值。通过测试应用程序中函数调用的返回值，你就可以用很少的代码正确地响应错误状态。

注意 这个函数在调试版本和非调试版本中具有相同的效果。

示例

```
LPDISPATCH pDisp = NULL;
AfxCheckError(CoCreateInstance(CLSID,
  NULL,CLSCTX_LOCAL_SERVER,IID_IDispatch,
```

```
    (LPVOID)&pDisp));
// 如果有错误，则已经抛出了一个异常
// 我们可以开始使用返回的指针
COleDispatchDriver disp(pDisp);
//等等...
```

AfxCheckMemory

```
BOOL AfxCheckMemory( );
```

返回值

如果没有内存错误，则为非零值；否则为 0。

说明

这个函数使自由内存池有效并在需要时输出错误信息。如果这个函数没有检测到内存冲突，它什么也不输出。

当前在堆中分配的所有内存块都会被检查，包括那些用 new 分配的内存，但是不包括那些用直接调用内存分配函数分配的内存，例如 malloc 函数或者 Windows 的 GlobalAlloc 函数。如果发现有内存块存在错误，就会在调试器上输出错误信息。

如果你在程序模块中包含了下面的程序行：

```
#define new DEBUG_NEW
```

后面对 AfxCheckMemory 的调用都会显示发生内存分配的文件名和行号。

> 注意　如果你的模块中包含了一个或多个串行化类的实现，那么你必须在最后一个 IMPLEMENT_SERIAL 宏之后包含#define 程序行。
>
> 这个函数仅在 MFC 的调试版本中起作用。

示例

```
// AfxCheckMemory 的例子
CAge* pcage = new CAge( 21 );    // CAge 是从 CObject.继承而来的
Age* page = new Age( 22 );       // Age 不是从 CObject.继承的
*(((char*) pcage) - 1) = 99;     // 破坏前面的保护字节
*(((char*) page) - 1) = 99;      // 破坏前面的保护字节
AfxCheckMemory();
```

程序的结果如下：

```
memory check error at $0067495F = $63, should be $FD
DAMAGE: before Non-Object block at $00674960
Non-Object allocated at file test02.cxx(48)
Non-Object located at $00674960 is 2 bytes long
```

```
memory check error at $00674905 = $63, should be $FD
DAMAGE: before Object block at $00674906
Object allocated at file test02.cxx(47)
Object located at $00674906 is 6 bytes long
```

AfxConnectionAdvise

BOOL AFXAPI AfxConnectionAdvise(LPUNKNOWN *pUnkSrc*,REFIID *iid*,
LPUNKNOWN *pUnkSink*,BOOL *bRefCount*,DWORD FAR**pdwCookie*);

返回值

如果建立了连接则为非零值，否则为0。

参数

pUnkSrc
指向调用接口的对象的指针。

pUnkSink
指向实现接口的对象的指针。

iid

连接的接口 ID。

bRefCount

如果为 TRUE，表明创建连接会引起 *pUnkSink* 的引用计数增加。如果为 FALSE，则表明应用计数不会增加。

pdwCookie

指向连接标识符返回的 DWORD 值的指针。当断开连接的时候，必须将这个值传递给 AfxConnectionUnadvise 的 *dwCookie* 参数。

说明

调用这个函数在 *pUnkSrc* 指明的源端和 *pUnkSink* 指明的接收端之间建立连接。

请参阅 AfxConnectionUnadvise

AfxConnectionUnadvise

```
BOOL AFXAPI AfxConnectionUnadvise(LPUNKNOWN pUnkSrc, REFIID iid,
        LPUNKNOWN pUnkSink, BOOL bRefCount, DWORD dwCookie);
```

返回值

如果断开了连接则为非零值，否则为 0。

参数

pUnkSrc

指向调用接口的对象的指针。

pUnkSink

指向实现接口的对象的指针。

iid

连接点接口的接口 ID。

bRefCount

如果为 TRUE，表明断开连接会引起 *pUnkSink* 的引用计数减小。如果为 FALSE，则表明应用计数不会减小。

pdwCookie

由 AfxConnectionAdvise 返回的连接标识符

说明

调用这个函数以断开 *pUnkSrc* 指明的源端和 *pUnkSink* 指明的接收端之间的连接。

请参阅 AfxConnectionAdvise

AfxDaoInit

```
void AfxDaoInit();
        throw (CDaoException);
```

说明

这个函数初始化 DAO **数据库引擎。在大多数情况下，你没有必要调用** AfxDaoInit**，因为应用程序会在有必要时自动调用。**

相关信息和 AfxDaoInit **的调用示例请参见** Visual C++**的联机文档中“技术注释** 54**”一文。**

请参阅 AfxDaoTerm

AfxDaoTerm

```
void AfxDaoTerm()
```

说明

这个函数终止 DAO **数据库引擎。通常你只需在常规的** DLL **中调用这个函数。应用程序会在必要时自动调用** AfxDaoTerm**。**

在常规 DLL 中，应该在所有的 MFC DAO 对象都被销毁以后，而在 ExitInstance 函数以前调用 AfxDaoTerm。

有关调用 AfxDaoTerm 的更多信息请参阅《Visual C++程序员指南》中的“DAO：在 DLL 中使用 DAO”一文。相关信息参见“Visual C++联机文档”中的“技术注释 54”。

请参阅 AfxDaoInit

AfxDbInitModule

```
void AFXAPI AfxDbInitModule();
#include <afxdll.h>
```

为了在 MFC 的自动连接的常规 DLL 中支持 MFC 数据库（或者 DAO），应该在你的常规 DLL 的 CWinApp::InitInstance 函数中增加对这个函数的调用以初始化 MFC 的数据库 DLL。确保这个调用出现在使用 MFC 数据库 DLL 的任何基类或附加代码之前。MFC 数据库 DLL 是一种扩展 DLL，为了使一个扩展 DLL 进入一个 CDynLinkLibrary 链，必须在每个可能使用它的模块的环境中创建一个 CDynLinkLibrary 对象。AfxDbInitModule 在你的常规 DLL 的环境中创建一个 CDynLinkLibrary 对象，使它能够进入常规 DLL 的 CDynLinkLibrary 链。

AfxDoForAllClasses

void AFXAPI AfxDoForAllClasses(void (**pfn*)(const CRuntimeClass* *pClass*, void* *pContext*), void* *pContext*);

参数

pfn

指向每个类都会调用的重复函数。这个参数是一个指向 CRuntimeClass 对象的指针以及指向调用者提供给函数的附加数据的 void 指针。

pContext

指向调用者提供给重复函数的可选数据的指针。这个指针可以是 NULL。

说明

在应用程序的内存空间中，为所有从 CObject 继承的可串行化的类调用指定的重复函数。从 CObject 继承的可串行化的类是以 DECLARE_SERIAL 宏继承的。每次调用指定的重复函数时，都会将在 *pContext* 中传递给 AfxDoForAllClasses 的指针传递给重复函数。

注意 这个函数仅在 MFC 的调试版本中起作用。

请参阅 DECLARE_SERIAL

AfxDoForAllObjects

void AfxDoForAllObjects(void (**pfn*)(CObject* *pObject*,void* *pContext*),void* *pContext*);

参数

pfn

　　指向每个对象都执行的重复函数。函数的参数是一个指向 CObject 的指针以及指向调用者提供给函数的附加数据的 void 指针。

pContext

　　指向调用者提供给函数的附加数据的指针。这个指针可以为 NULL。

说明

对每个用 new 分配的从 CObject 继承的对象执行指定的重复函数。栈、全局变量或嵌入对象不包括在内。每次调用指定的重复函数时，都会将在 *pContext* 中传递给 AfxDoForAllObjects 的指针传递给重复函数。

　　注意　这个函数仅在 MFC 的调试版本中起作用。

afxDump

CDumpContext afxDump;

说明

使用这个变量为你的应用程序提供基本的对象转储能力。afxDump 是预定义的 CDumpContext 对象，它使你能够将 CDumpContext 信息发送到调试器输出窗口或者调试终端。通常把 afxDump 作为 CObject::Dump 的一个参数。

在 Windows NT 和 Windows 95（以及 Windows 的早期版本）中，当你调试应用程序时，afxDump 输出被发送到 Visual C++的调试输出窗口。

这个变量仅在 MFC 的调试版本中定义。有关 afxDump 的更多信息请参见《Visual C++程序员指南》中的“MFC 调试支持”。Visual C++连接文档中的“技术注释 7”和“技术注释 12”中包含了其它一些信息。

注意 这个函数仅在 MFC 的调试版本中起作用。

示例

```
// afxDump 的示例
CPerson myPerson = new CPerson;
// 设置 CPerson 对象的一些域
```

```
//..
// 现在转储内容
#ifdef _DEBUG
afxDump << "Dumping myPerson:\n";
myPerson->Dump(afxDump);
afxDump <<"\n";
#endif
```

请参阅 Cobject::Dump AfxDump

AfxDump

void AfxDump(const CObject* *pOb*);

参数

pOb

　　指向由 CObject **继承的类的对象的指针。**

说明

在调试器中调用这个函数以在调试时转储对象的状态。AfxDump **调用一个对象的** Dump **函数并且将信息发送到** afxDump **变量指定的位置。**AfxDump **仅能在**

MFC 的调试版本中使用。

你的程序代码不应该调用 AfxDump，而是应该调用适当对象的 Dump 成员函数。

请参阅 CObject::Dump, afxDump

AfxDumpStack

```
void AFXAPI AfxDumpStack(DWORD dwTarget =
        AFX_STACK_DUMP_TARGET_DEFAULT);
```

参数

dwTarget

指出转储输出的目标。其取值可以用位或操作符（|）组合起来，可能值如下：

- AFX_STACK_DUMP_TARGET_TRACE 通过 TRACE 宏输出。TRACE 仅仅在调试版本中产生输出，在发行版本中不产生输出。同时，TRACE 可以被重定向到调试器以外的目标。
- AFX_STACK_DUMP_TARGET_DEFAULT 将转储输出发送到缺省目标。对于调试版本，输出发送给 TRACE 宏。在发行版本中，输出发送到剪贴板。

- AFX_STACK_DUMP_TARGET_CLIPBOARD 输出仅发送到剪贴板。数据将按 CF_TEXT 格式以普通文本的形式放在剪贴板上。
- AFX_STACK_DUMP_TARGET_BOTH 同时将输出发送到剪贴板和 TRACE 宏。
- AFX_STACK_DUMP_TARGET_ODS 通过 Win32 函数 OutputDebugString()直接将输出发送的调试器。如果连接了调试器，它在调试版本和发行版本中都会产生调试器输出。AFX_STACK_DUMP_TARGET_ODS 通常到达调试器（如果连接了调试器），并且不能被重定向。

说明

这个全局函数可以被用来生成当前栈的一个映象。下面的例子反映了 MFC 对话框应用程序中按钮处理函数调用 AfxDumpStack 所产生的单行调试输出：

```
=== begin AfxDumpStack output ===
...
BFF928E0: WINDOWS\SYSTEM\KERNERL32.DLL! UTUnRegister + 2492 bytes

=== end AfxDumpStack() output ===
```

下面的表格描述了上面的输出行：

输出	描述
BFF928E0：	最近一次函数调用的返回地址
WINDOWS\SYSTEM\KERNEL32.DLL!	包含函数调用的模块的完整路径名
UTUnRegister	调用的函数原型
+ 2492 bytes	以字节为单位的从函数原型地址（这个例子中为 UTUnregister）到返回地址（这个例子中为 BEF928E0）的偏移

AfxDumpStack 在 MFC 库的调试版本和非调试版本中都可以使用。但是，这个函数通常是静态连接的，即使你的可执行文件以共享 DLL 的方式使用 MFC。在共享库的实现中，可以在 MFCS42.LIB 库（以及它的变化形式）中找到这个函数。

为了成功地使用这个函数：

- 在你的路径中必须包含 IMAGEHLP.DLL 文件。如果你没有这个 DLL 文件，这个函数会显示一条错误信息。IMAGEHLP.DLL 是随 Win32 SDK 和 Windows 一起发售的可散发的 DLL。在 C:\[Windows]\system[32]下查找它。有关 IMAGEHLP 提供的函数集的介绍可以参考“可移植的可执行文件的操作”一文。
- 具有栈框架的模块必须包含调试信息。如果它不包含调试信息，这个函数仍然会生成对栈的跟踪，但是这种跟踪是很简略的。

请参阅 afxDump

AfxEnableControlContainer

void AfxEnableControlContainer();

说明

在你应用程序对象的 InitInstance **函数中调用这个函数，以便能够提供对** OLE **控件容器的支持。**

有关 OLE **控件（现在被叫做** ActiveX **控件）的更多信息请参阅《**Visual C++**程序员指南》中的“** ActiveX **控件主题”。**

AfxEnableMemoryTracking

BOOL AfxEnableMemoryTracking(BOOL *bTrack*);

返回值

以前的跟踪允许状态设置。

参数

bTrack

将这个值设为 TRUE 时就打开了内存跟踪特性。如果是 FALSE 则将其关闭。

说明

诊断内存跟踪通常在 MFC 的调试版本中有效。利用这个函数对你的代码中正确分配内存的部分禁止跟踪。

有关 AfxEnableMemoryTracking 的更多信息请参见《Visual C++程序员指南》中的“MFC 调试支持”。

注意 这个函数仅在 MFC 的调试版本中起作用。

AfxEndThread

void AfxEndThread(UINT *nExitCode*);

参数

nExitCode

指定线程的退出代码。

说明

调用这个函数以结束当前运行的线程。必须在要终止的线程内部调用。

有关 AfxEndThread 的更多信息请参见《Visual C++程序员指南》中的“多线程：终止线程”。

请参阅 AfxBeginThread

AFX_EXT_CLASS

说明

扩展 DLL 使用 AFX_EXT_CLASS 宏以引出类。与扩展 DLL 连接的可执行程序使用这个宏以引入类。利用 AFX_EXT_CLASS 宏，创建扩展 DLL 时使用的头文件可以被与该 DLL 连接的可执行文件使用。

在你的 DLL 的头文件中，在类定义中加入 AFX_EXT_CLASS 关键字，如下所示：

```
class AFX_EXT_CLASS CMyClass : public CDocument
{
// 类的主体
};
```

更多的信息参见“使用 AFX_EXT_CLASS 来引入和引出”。

AfxFormatString1

void AxFormatString1(CString& *rString*, UINT *nIDS*, LPCTSTR *lpsz1*);

参数

rString

对 CString 对象的引用，在替换之后它将包含结果字符串。

nIDS

模板字符串的资源 ID，替换将在模板字符串上发生。

lpsz1

将替换模板字符串中格式字符“%1”的字符串。

说明

调入指定的字符串资源并将字符“%1”替换为 *lpsz1* 指向的字符串。新形式的字符串保存在 *rString* 中。例如，如果字符串表中的字符串是“File %1 not fount”，而 *lpszl* 代表“C:\MYFILE.TXT”,则 *rString* 包含的字符串为“File C:\MYFILE.TXT no found”。这个函数在格式化要向消息框或其它窗口发送的字符串时是非常

有用的。

如果字符串中格式字符“%1”出现了不止一次，那么会进行多次替换。

请参阅 AxFormatString2

AfxFormatString2

void AxFormatString2(CString& *rString*, UINT *nIDS*, LPCTSTR *lpsz1*, LPCTSTR *lpsz2*);

参数

rString

对 CString 对象的引用，在替换之后它将包含结果字符串。

nIDS

模板字符串的资源 ID，替换将在模板字符串上发生。

lpsz1

将替换模板字符串中格式字符“%1”的字符串。

lpsz2

将替换模板字符串中格式字符“%2”的字符串。

说明

调入指定的字符串资源并将字符“%1”和“%2”替换为 *lpsz1* 和 *lpsz2* 指向的字符串。新形式的字符串保存在 *rString* 中。例如，如果字符串表中的字符串是“File %1 not fount in %2”，而 *lpsz1* 指向“C:\MYFILE.TXT”，*lpsz2* 指向“C:\MYDIR”，则 *rString* 包含的字符串为“File C:\MYFILE.TXT no found in C:\MYDIR”。

如果字符串中格式字符“%1”和“%2”出现了不止一次，那么会进行多次替换。它们不必按照数字顺序排列。

请参阅 AxFormatString1

AfxFreeLibrary

BOOL AFXAPI AfxFreeLibrary(HINSTANCE *hInstLib*);

返回值

如果函数执行成功则为 TRUE，否则为 FALSE。

参数

hInstLib

已调入的库模块的句柄。AfxLoadLibrary 返回这个句柄。

说明

AfxFreeLibrary 和 AfxLoadLibrary 都为每一个调入的库模块维护着引用计数。AfxFreeLibrary 减小调入的动态链接库（DLL）模块的引用计数。当引用计数减小到 0 时，这个模块将从调用进程的地址空间中除去，其句柄也不再有效。每次调用 AfxLoadLibrary 就会增加这个引用计数。

在卸载一个库模块之前，系统使 DLL 能够从使用它的进程分离出来。这样就给 DLL 一个机会以清除当前进程分配的资源。当入口点函数返回之后，库模块就从当前进程的地址空间中移去了。

使用 AfxLoadLibrary 来映射一个 DLL 模块。

如果你的程序使用了多线程，确保你使用的是 AfxFreeLibrary 和 AfxLoadLibrary（而不是 Win32 函数 FreeLibrary 和 LoadLibrary）。当调入或卸载扩展 DLL 时，使用 AfxLoadLibrary 和 AfxFreeLibrary 来启动和关闭执行代码可以保证 MFC 的全局状态不被破坏。

请参阅 AfxLoadLibrary

AfxGetApp

CWinApp* AfxGetApp();

返回值

指向应用程序的单一 CWinApp 对象的指针。

说明

这个函数返回的指针可以被用来访问应用程序的信息，如主消息调度代码以及顶层窗口等。

AfxGetAppName

LPCTSTR AfxGetAppName();

返回值

包含应用程序名字的以 null 字符结尾的字符串。

说明

这个函数返回的字符串可以被用于诊断消息，或是用作一个临时字符串名的根。

AfxGetHENV

HENV AFXAPI AfxGetHENV();

返回值

当前 MFC **使用的** ODBC **环境的句柄。如果现在没有任何** CDatabase **对象或者** CRecordset **对象正被使用，则返回** SQL_HENV_NULL。

说明

你可以在直接调用用 ODBC **时使用返回的句柄，但是你不能关闭句柄，也不能假定在现存的从** CDatabase **或** CRecordset **继承的对象被销毁后句柄依然有效。**

AfxGetInstanceHandle

HINSTANCE AfxGetInstanceHandle( );

返回值

代表应用程序的当前实例的 HINSTANCE 值。如果是从与 MFC 的 USRDLL 版本连接的 DLL 内调用的，则返回代表 DLL 的 HINSTANCE 值。

说明

这个函数使你能够获得当前应用程序的实例句柄。AfxGetInstanceHandle 总是返回代表你的可执行文件（.EXE）的 HINSTANCE 值，除非它从与 MFC 的 USRDLL 版本连接的 DLL 内调用的。在这种情况下，它返回的是 DLL 的 HINSTANCE 值。

请参阅 AfxGetResourceHandle, AfxSetResourceHandle

AfxGetInternetHandleType

DWORD AFXAPI AfxGetInternetHandleType(HINTERNET *hQuery*)

返回值

在 W ININET.H 中定义的任何 Internet 服务类型。这些 Internet 服务的列表参见说明部分。如果这个句柄是 NULL 或者不能被识别，则函数返回 AFX_INET_SERVICE_UNK。

参数

hQuery

执行一个查询的句柄。

说明

使用这个全局函数来决定一个Internet句柄的类型。

下面的列表包含了AfxGetInternetHandleType可能返回的Internet类型。

- INTERNET_HANDLE_TYPE_INTERNET
- INTERNET_HANDLE_TYPE_CONNECT_FTP
- INTERNET_HANDLE_TYPE_CONNECT_GOPHER
- INTERNET_HANDLE_TYPE_CONNECT_HTTP
- INTERNET_HANDLE_TYPE_FTP_FIND
- INTERNET_HANDLE_TYPE_FTP_FIND_HTML
- INTERNET_HANDLE_TYPE_FTP_FILE
- INTERNET_HANDLE_TYPE_FTP_FILE_HTML
- INTERNET_HANDLE_TYPE_GOPHER_FIND
- INTERNET_HANDLE_TYPE_GOPHER_FIND_HTML
- INTERNET_HANDLE_TYPE_GOPHER_FILE
- INTERNET_HANDLE_TYPE_GOPHER_FILE_HTML

• INTERNET_HANDLE_TYPE_HTTP_REQUEST

注意 为了调用这个函数，你的项目必须包含AFXINET.H。

请参阅 AfxParseURL

AfxGetMainWnd

CWnd* AfxGetMainWnd( );

返回值

如果服务器具有一个可以在容器内现场激活的对象，并且这个容器是活动的，则这个函数返回一个指向包含这个现场活动文档的框架窗口对象的指针。

如果没有可以在容器内现场激活的对象，或者你的应用程序不是一个OLE服务器，这个函数仅返回你的应用程序对象的m_pMainWnd。

如果AfxGetMainWnd被应用程序主线程调用，它根据以上规则返回应用程序的主窗口。如果该函数被应用程序的次线程调用，该函数返回与引起该调用线程连接的主窗口。

说明

如果你的应用程序是一个OLE服务器，应该调用这个函数以获得应用程序的

活动主窗口指针，而不是直接引用应用程序对象的 m_pMainWnd 成员。

如果你的应用程序不是 OLE 服务器，那么调用这个函数与直接引用应用程序对象的 m_pMainWnd 成员是等价的。

请参阅 CWinThread::m_pMainWnd

AfxGetResourceHandle

HINSTANCE AfxGetResourceHandle( );

返回值

应用程序调入缺省资源的实例的 HINSTANCE 句柄。

说明

利用这个函数返回的 HINSTANCE 句柄来直接访问应用程序的句柄，例如，在调用 Windows 函数 FindResource 时使用。

请参阅 AfxGetInstanceHandle, AfxSetResourceHandle

AfxGetStaticModuleState

AFX_MODULE_STATE* AFXAPI AfxGetStaticModuleState( );

返回值

指向一个 AFX_MODULE_STATE 结构的指针。

说明

调用这个函数以在初始化之前设置模块状态，或者在清除后恢复原来的模块状态。AFX_MODULE_STATE 结构中包含了该模块的全局数据，这是被 push 或 pop 的模块状态的一部分。

在缺省情况下，MFC 利用主应用程序的资源句柄来调入资源模板。如果你在一个 DLL 中具有一个输出函数，例如有个函数在 DLL 中启动一个对话框，这个模板将被保存于 DLL 模块中。你可能需要改变模块的状态以便使用正确的句柄。你可以在函数的开头加入下面的代码以实现上述功能：

```
AFX_MANAGE_STATE(AfxGetStaticModuleState( ));
```

这会交换当前模块的状态和 AfxGetStaticModuleState 返回的状态，直到当前作用域结束。

有关模块状态和 MFC 的更多信息请参见《Visual C++程序员指南》的“创建新

的文档、窗口和视窗”中的“管理 MFC 模块的状态数据”，以及 Visual C++ 联机文档中的“技术注释 58”。

请参阅 AFX_MANAGE_STATE

AfxGetThread

```
CWinThread* AfxGetThread( );
```

返回值

指向当前执行的线程的指针。

说明

调用这个函数以获得代表当前执行的线程的 CWinThread 对象指针。必须在要求的线程内部调用。

请参阅 AfxBeginThread

AfxInitExtensionModule

BOOL AFXAPI AfxInitExtensionModule(AFX_EXTENSION_MODULE& *state*,

```
        HMODULE hModule );
```

返回值

如果成功地初始化了扩展DLL，则返回TRUE；否则返回FALSE。

参数

state

对初始化后包含了扩展DLL模块状态的AFX_EXTENSION_MODULE结构的引用。这个状态中包含了扩展DLL在进入DllMain之前作为一般静态对象构造过程的一部分而初始化的运行类对象的拷贝。

hModule

扩展DLL模块的句柄。

说明

在扩展DLL的DllMain中调用此函数以初始化DLL。例如：

```
static AFX_EXTENSION_MODULE extensionDLL;
extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID)
{
    if (dwReason == DLL_PROCESS_ATTACH)
```

```
    {
        // 扩展 DLL 一次性初始化
        if (!AfxInitExtensionModule(extensionDLL, hInstance))
            return 0;
...
```

AfxInitExtensionModule 生成 DLL 的 HMODULE 的一个拷贝，并捕捉 DLL 的运行类（CRuntimeClass 结构）和它的对象工厂（COleObjectFactory 对象）以供创建 CDynLinkLibrary 时使用。

MFC 的扩展 DLL 必须在它们的 DllMain 函数中做两件事情：

- 调用 AfxInitExtensionModule 并检验返回值。
- 如果 DLL 要引出 CRuntimeClass 对象或者具有它自己的资源，则创建一个 CDynLinkLibrary 对象。

当每个进程与扩展 DLL 分离时（这发生在进程退出，或者因为调用了 AfxFreeLibrary，DLL 被卸载时），你可以调用 AfxTermExtensionModule 来清除扩展 DLL。

请参阅　AfxTermExtensionModule

AfxInitRichEdit

BOOL AFXAPI AfxInitRichEdit( );

说明

调用这个函数来为应用程序初始化带格式编辑控件。如果该进程还没有初始化公共控件库，则它还初始化公共控件库。如果你在 MFC 应用程序内直接使用带格式编辑控件，你必须调用这个函数以确保 MFC 正确地初始化了带格式编辑控件。如果你通过 CRichEditCtrl，CRichEditView 或 CRichEditDoc 使用带格式控件，那么你不需要调用这个函数。

AfxIsMemoryBlock

BOOL AfxIsMemoryBlock(const void* *p*, UINT *nBytes*, LONG* *plRequestNumber* = NULL);

返回值

如果内存块是现在分配的，并且其长度也是正确的，则返回非零值。否则为 0。

参数

p

指向将被测试的内存块。

nBytes

包含了以字节为单位的内存块长度。

plRequestNumber

指向一个长整数,它将被设为内存块的分配系列号码。由 plRequestNumber 指向的这个变量只有当 AfxIsMemoryBlock 返回非零值时才会被填充。

说明

检测一个内存地址，确保它代表了一个由 new 的诊断版本分配的活动的内存块。它同时也检验指定的大小是否与最初分配的大小相符。如果这个函数返回非零值，分配的系列号码将在 *plRequestNumber* 中返回。这个号码代表了这个内存块相对于其它所有内存分配的顺序。

示例

```
// AfxIsMemoryBlock 的例子
CAge* pcage = new CAge( 21 ); // CAge is derived from CObject.
ASSERT( AfxIsMemoryBlock( pcage, sizeof( CAge ) ) )
```

请参阅 AfxIsValidAddress

AfxIsValidAddress

BOOL AfxIsValidAddress(const void* *lp*, UINT *nBytes*, BOOL *bReadWrite* = TRUE);

返回值

如果指定的内存块完全包括在程序的内存空间内，则返回非零值；否则返回 0。

参数

lp

指向将要被检测的内存地址。

nBytes

包含了要被测试的内存的字节数。

bReadWrite

指明这块内存是可以读写（TRUE**）还是只读（**FALSE**）。**

说明

测试任意的内存地址，确保它们完全处于程序的内存空间内。该地址并不仅限于用 new 分配的内存块。

请参阅 AfxIsMemoryBlock, AfxIsValidString

AfxIsValidString

BOOL AfxIsValidString(LPCSTR *lpsz*, int *nLength* = –1);

返回值

如果给定的指针指向一个给定大小的字符串则返回非零值，否则返回 0。

参数

lpsz

要测试的指针。

nLength

指定要测试的字符串的长度，以字节为单位。如果值为 –1，表示字符串是以 null 结尾的。

说明

使用这个函数来确定指向字符串的指针是否有效。

请参阅 AfxIsMemoryBlock, AfxIsValidAddress

AfxLoadLibrary

HINSTANCE AFXAPI AfxLoadLibrary(LPCTSTR *lpszModuleName*);

返回值

如果函数成功执行，返回值为模块的句柄。如果函数执行失败，返回值为 NULL。

参数

lpszModuleName

指向一个以 null 结尾的字符串，其中包含了模块的名字（DLL 或 EXE 文件）。指定的名字为模块的文件名。

如果该字符串指定了一个路径但是在指定的目录中不存在该文件，这个函数执行失败。

如果没有指定路径并且省略了文件的扩展名，将会附加缺省的扩展

名.DLL。但是，文件名字符串可以在尾部包含一个点字符（.），用以指示模块名没有扩展名。如果没有指定路径，这个函数将按如下顺序搜索文件：

- 应用程序的启动路径。
- 当前目录。
- Windows 95：Windows 系统目录。Windows NT： 32 位 Windows 系统目录。目录的名字为 SYSTEM32.
- 仅适用于 Windows NT：16 位 Windows 系统目录。没有 Win32 函数可以获得这个目录的路径，但是将会搜索这个目录。这个目录的名字是 SYSTEM。
- Windows 目录。
- 在 PATH 环境变量中列出的路径。

说明

使用 AfxLoadLibrary 来载入一个 DLL 模块。它返回一个句柄，可被用于 GetProcAddress 以获得 DLL 函数的地址。AfxLoadLibrary 还可以用来载入其它可执行模块。

每个进程都为载入的库模块维护这一个引用计数。这个引用计数在调用 AfxLoadLibrary 时增加，而在调用 AfxFreeLibrary 时减小。当引用计数减小到零时，该模块就被从调用进程的地址空间中移去，句柄也不再有效。

如果你使用多线程，确保你使用AfxLoadLibrary和AfxFreeLibrary(而不是Win32函数LoadLibrary和FreeLibrary)。当调入或卸载扩展DLL时使用AfxLoadLibrary和AfxFreeLibrary来启动和关闭执行代码可以保证MFC的全局状态不被破坏。
请参阅　AfxFreeLibrary

AFX_MANAGE_STATE

AFX_MANAGE_STATE(AFX_MODULE_STATE* pModuleState)

参数

pModuleState
　　指向一个AFX_MODULE_STATE结构的指针。

说明

调用这个宏以保护DLL中的引出函数。调用了这个宏以后，*pModuleState*就是立即包含域的剩余部分的实际模块状态。在脱离作用域之前，原来的实际模块状态将被自动保存。

AFX_MODULE_STATE结构中包含了有关模块的全局数据，这是被push或pop的模块状态的一部分。

在缺省情况下，MFC 利用主应用程序的资源句柄来载入资源模板。如果你在一个 DLL 中有一个输出函数，例如在 DLL 中启动一个对话框的函数，这个模板实际保存在 DLL 模块中。你需要为将要使用的正确的句柄切换模块状态。你可以在函数的开头加上如下代码以实现这个目的：

```
AFX_MANAGE_STATE(AfxGetStaticModuleState( ));
```

这会交换当前模块的状态和 AfxGetStaticModuleState 返回的状态，直到当前作用域结束。

有关模块状态和 MFC 的更多信息请参见《Visual C++程序员指南》的“创建新的文档、窗口和视窗”中的“管理 MFC 模块的状态数据”，以及 Visual C++联机文档中的“技术注释 58”。

请参阅 AfxGetStaticModuleState

afxMemDF

```
int afxMemDF;
```

说明

这个变量在调试器或你的程序中都可以访问，它允许你调整分配诊断。它可以具有枚举变量 afxMemDF 指定的下列值：

- allocMemDF 打开调试分配函数（调试库中的缺省设置）。
- delayFreeMemDF 延迟释放内存。当你的应用程序释放一个内存块的时候，分配函数并不将此内存返还给操作系统。这将带给你的程序最大的内存压力。
- checkAlwaysMemDF 每次分配或释放内存时调用 AfxCheckMemory。这会显著地减慢分配和收集内存的过程。

示例

```
// afxMemDF 的例子
afxMemDF = allocMemDF | checkAlwaysMemDF;
```

AfxMessageBox

int AfxMessageBox(LPCTSTR *lpszText*, UINT *nType* = MB_OK, UINT *nIDHelp* = 0);
int AFXAPI AfxMessageBox(UINT *nIDPrompt*, UINT *nType* = MB_OK, UINT *nIDHelp* = (UINT) –1);

返回值

如果没有足够的内存来显示消息框就返回零，否则可能是下列值之一：

- IDABORT 选择了 Abort 按钮。
- IDCANCEL 选择了 Cancel 按钮。
- IDIGNORE 选择了 Ignore 按钮。
- IDNO 选择了 No 按钮。
- IDOK 选择了 OK 按钮。
- IDRETRY 选择了 Retry 按钮。
- IDYES 选择了 Yes 按钮。

如果消息框具有 Cancel 按钮，那么当按下了 ESC 键或者是选择了 Cancel 按钮时，就会返回 IDCANCEL 值。如果消息框没有 Cancel 按钮，按下 ESC 键没有任何效果。

在格式化消息框中显示的字符串时，函数 AfxFormatString1 和 AfxFormatString2 非常有用。

参数

lpszText

指向一个 CString 对象或者以 null 结尾的字符串，包含了要在消息框中显示的信息。

nType

消息框的风格。对对话框应用任何一种消息框风格。

nIDHelp

消息的帮助上下文ID。0表示将使用应用程序的缺省帮助上下文。

nIDPrompt

用于引用字符串表中的字符串的唯一的ID。

说明

在屏幕上显示一个消息框，这个重载函数的第一种形式在消息框中显示由lpszText指向的文本字符串，并使用*nIDHelp*来描述帮助上下文。当用户按下了帮助键（通常是F1）时，帮助上下文被用来跳转到相关的帮助主题。

函数的第二种形式使用*nIDPrompt*所代表的字符串资源来在消息框中显示一个消息。相关的帮助页面通过*nIDHelp*的值来查找。如果使用了*nIDHelp*的缺省值（－1），则帮助上下文将使用字符串资源ID，即*nIDPrompt*。有关定义帮助上下文的更多信息参见《Visual C++程序员指南》中的“帮助主题”一文以及Visual C++联机文档中的“技术注释28”。

请参阅 CWnd::MessageBox

AfxNetInitModule

```
void AFXAPI AfxNetInitModule( );
```

```
#include <afxdll.h>
```

说明

为了在与 MFC 动态连接的常规 DLL 中增加 MFC 的 Sockets 支持，可以在你的常规 DLL 的 CWinApp::InitInstance 函数中加入对这个函数的调用以初始化 MFC 的 SocketsDLL。MFC 的 Sockets DLL 是一种扩展 DLL，为了使扩展 DLL 加入一个 CDynLinkLibrary 链，必须在每个使用它的模块的环境中创建一个 CDynLinkLibrary 对象。AfxNetInitModule 在你的常规 DLL 的环境中创建了一个 CDynLinkLibrary 对象，这样它就可以进入常规 DLL 的 CDynLinkLibrary 对象链了。

AFX_ODBC_CALL

AFX_ODBC_CALL(*SQLFunc*)

参数

SQLFunc

一个 ODBC API 函数。有关 ODBC API 函数的更多信息请参见《ODBC SDK 程序员参考》。

说明

使用这个宏来调用那些返回 SQL_STILL_EXECUTING 的 ODBC API 函数。AFX_ODBC_CALL 会重复调用这个函数，直到它们不再返回 SQL_STILL_EXECUTING。

在使用 AFX_ODBC_CALL 之前，你必须声明一个 RETCODE 类型的变量 nRetCode。你可以在调用了这个宏后使用 CRecordset::Check 函数来检查 nRetCode 的值。

注意现在 MFC 的 ODBC 类仅使用同步处理。如果要实现异步操作，你必须调用 ODBC 的 API 函数 SQLSetConnectOption。有关的更多信息请参见《ODBC SDK 程序员参考》中的“异步执行函数”主题。

示例

这个例子使用 AFX_ODBC_CALL 来调用 ODBC 的 API 函数 SQLColumns，它返回由 strTableName 列名的表中的数据列的列表。注意，声明了 nRetCode，并且将记录集数据成员作为参数传递给该函数。这个例子同时还演示了如何用 CRecordset 类的成员函数 Check 来检查结果。变量 prs 是指向 CRecordset 对象的指针，可以在别处声明。

```
// AFX_ODBC_CALL 的例子
RETCODE nRetCode;
```

```
AFX_ODBC_CALL( ::SQLColumns( prs->m_hstmt,
            (UCHAR *)NULL, SQL_NTS, (UCHAR *)NULL,
            SQL_NTS, (UCHAR *)(constchar*)strTableName,
            SQL_NTS, (UCHAR *)NULL, SQL_NTS ) );
if ( !prs->Check( nRetCode ) )
{
    AfxThrowDBException( nRetCode, prs->m_pdb,
                         prs->m_hstmt );
    TRACE( "SQLColumns failed\n" );
}
```

请参阅 AFX_SQL_ASYNC, AFX_SQL_SYNC

AfxOleCanExitApp

```
BOOL AFXAPI AfxOleCanExitApp( );
#include <afxdisp.h>
```

返回值

如果应用程序可以退出，则返回非零值；否则为 0。

说明

指明一个应用程序是否可以退出。如果还存在着对应用程序对象的引用，它就不能退出。全局函数 AfxOleLockApp 以及 AfxOleUnlockApp 分别增加和减小对应用程序对象的应用计数。在这个计数不为零时，应用程序就不能退出。如果这个计数为非零值，那么当用户选择了系统菜单中的 Close 或者 File 菜单中的 Exit 时，应用程序的主窗口就被隐藏（不会被销毁）。应用框架会在 CFrameWnd::OnClose 中调用这个函数。

请参阅 AfxOleLockApp, AfxOleUnlockApp

AfxOleGetMessageFilter

```
COleMessageFilter* AFXAPI AfxOleGetMessageFilter( );
#include <afxwin.h>
```

返回值

指向当前消息过滤器的指针。

说明

获得应用程序当前的消息过滤器。调用这个函数可以访问当前从

COleMessageFilter继承的对象，正如你调用 AfxGetApp 可以访问当前的应用程序对象一样。

示例

```
COleMessageFilter* pFilter = AfxOleGetMessageFilter();
ASSERT_VALID(pFilter);
pFilter->BeginBusyState();
// 完成一些需要 busy 状态的操作
pFilter->EndBusyState();
```

请参阅 COleMessageFilter, AfxGetApp

AfxOleGetUserCtrl

```
BOOL AFXAPI AfxOleGetUserCtrl( );
#include <afxdisp.h>
```

返回值

如果用户正控制应用程序，则返回非零值；否则为 0。

说明

获得当前的用户控制标志。当用户打开或创建了一个新文档时，就意味着该用户控制着应用程序。如果应用程序不是由 OLE 系统 DLL 启动的，换句话说，如果用户通过系统外壳启动了应用程序，也认为是用户控制了应用程序。

请参阅　AfxOleSetUserCtrl

AfxOleInit

```
BOOL AFXAPI AfxOleInit( );
#include <afxdisp.h>
```

返回值

如果函数执行成功则返回非零值。如果初始化失败则为 0，可能是因为安装了版本不正确的 OLE 系统 DLL。

说明

初始化 OLE 系统 DLL。

AfxOleInitModule

```
void AFXAPI AfxOleInitModule( );
#include <afxdll.h>
```

说明

为了在与 MFC 动态连接的常规 DLL 中增加 MFC 的 OLE 支持，可以在你的常规 DLL 的 CWinApp::InitInstance 函数中加入对这个函数的调用以初始化 MFC 的 OLE DLL。MFC 的 OLE DLL 是一种扩展 DLL，为了使扩展 DLL 加入一个 CDynLinkLibrary 链，必须在每个使用它的模块的环境中创建一个 CDynLinkLibrary 对象。AfxNetInitModule 在你的常规 DLL 的环境中创建了一个 CDynLinkLibrary 对象，这样它就可以进入常规 DLL 的 CDynLinkLibrary 对象链了。

如果你正在创建一个 OLE 控件并使用了 COleControlModule，你就不用调用 AfxOleInitModule，因为 COleControlModule 的成员函数 InitInstance 以及调用了 AfxOleInitModule。

AfxOleLockApp

```
void AFXAPI AfxOleLockApp( );
```

```
#include <afxdisp.h>
```

说明

增加框架对应用程序的活动对象的全局计数。

框架保存对应用程序的活动对象的计数。AfxOleLockApp **和** AfxOleUnlockApp **分别增加和减小这个计数值。**

当用户试图关闭仍然具有活动对象的应用程序----**这样的应用程序的活动对象计数为非零值**----**时，框架将会把应用程序从用户的视野中隐藏起来，而不是完全地关闭它。**AfxOleCanExitApp **函数指明了应用程序是否可以退出。**

如果一个引出 OLE **接口的对象不希望在被客户应用程序使用时被销毁，就为它调用** AfxOleLockApp **函数。同时，对于在构造函数中调用了** AfxOleLockApp **的对象，应当在其析构函数内调用** AfxOleUnlockApp **函数。在缺省情况下，**COleDocument**（及其派生类）自动地锁定或者解锁应用程序。**

请参阅 AfxOleUnlockApp, AfxOleCanExitApp, COleDocument

AfxOleLockControl

```
BOOL AFXAPI AfxOleLockControl( REFCLSID clsid );
BOOL AFXAPI AfxOleLockControl( LPCTSTR lpszProgID );
```

```
#include <afxwin.h>
```

返回值

如果控件的类被成功地锁定，则返回非零值；否则返回 0。

参数

clsid

控件的唯一的类 ID。

lpszProgID

控件的唯一程序 ID。

说明

锁定了指定控件的类工厂以后，与该控件有关的动态生成的数据就保留在内存中。这样能够显著地加速控件的显示过程。例如，一旦你在对话框中创建了一个控件并且用 AfxOleLockControl **锁定了它，每次对话框显示或销毁的时候，你就不需要再创建或销毁这个控件了。如果用户重复地打开和关闭对话框，你的控件将会显著地提高性能。当你准备销毁控件时，调用** AfxOleUnlockControl **函数。**

请参阅 AfxOleUnlockControl

AfxOleRegisterControlClass

```
BOOL AFXAPI AfxOleRegisterControlClass( HINSTANCE hInstance, REFCLSID clsid, LPCTSTR pszProgID, UINT idTypeName, UINT idBitmap, int nRegFlags, DWORD dwMiscStatus, REFGUID tlid, WORD wVerMajor, WORD wVerMinor );
```

#include <afxctl.h>

返回值

如果注册了控件类，则返回非零值；否则返回 0。

参数

hInstance

与控件类相关的模块的实例句柄。

clsid

控件的唯一的类 ID。

pszProgID

控件的唯一的程序 ID。

idTypeName

字符串的资源 ID，该字符串中包含了用户可读的控件的类型名。

idBitmap

位图的资源 ID，该位图用于在工具条或调色板中表示 OLE 控件。

nRegFlags

包含了下列标志的一个或多个：

- afxRegInsertable 允许控件在 OLE 对象的插入对象对话框中出现。
- afxRegApartmentThreading 将注册表中的线程模式设置为 ThreadingModel=Apartment.

注意 在 MFC4.2 以前的 MFC 版本中，整型的 nRegFlags 参数是 BOOL 类型的。由 bInsetable 来决定是否可以通过插入对象对话框来加入控件。

dwMiscStatus

包含了下列状态标志中的一个或多个（对这些标志的详细描述请参见《OLE 程序员参考》中列举的 OLEMISC）。

- OLEMISC_RECOMPOSEONRESIZE
- OLEMISC_ONLYICONIC
- OLEMISC_INSERTNOTREPLACE
- OLEMISC_STATIC
- OLEMISC_CANTLINKINSIDE
- OLEMISC_CANLINKBYOLE1

- OLEMISC_ISLINKOBJECT
- OLEMISC_INSIDEOUT
- OLEMISC_ACTIVATEWHENVISIBLE
- OLEMISC_RENDERINGISDEVICEINDEPENDENT
- OLEMISC_INVISIBLEATRUNTIME
- OLEMISC_ALWAYSRUN
- OLEMISC_ACTSLIKEBUTTON
- OLEMISC_ACTSLIKELABEL
- OLEMISC_NOUIACTIVATE
- OLEMISC_ALIGNABLE
- OLEMISC_IMEMODE
- OLEMISC_SIMPLEFRAME
- OLEMISC_SETCLIENTSITEFIRST

tlid

控件类的唯一的 ID。

wVerMajor

控件类的主版本号码。

wVerMinor

控件类的次版本号码。

说明

通过 Windows 的注册数据库来注册控件类。这就使得控件可以被具有 OLE 控件能力的容器所使用。AfxOleRegisterControlClass 用控件的名字及其在系统中的位置来更新系统注册表，同时在注册表中设置控件支持的线程模式。有关的更多信息请参见《Visual C++联机文档》中的“技术注释 64”，名为“OLE 控件中的 Apartment 模式”，以及《Win32 SDK》中的“进程与线程”。

示例

```
// COleObjectFactory::UpdateRegistry 类成员函数的实现
//
BOOL CMyApartmentAwareCtrl::CApartmentCtrlFactory::UpdateRegistry(BOOL bRegister)
{
// 目的：检验你的控件是否遵循 Apartment 模式线程规则。
// 更多的信息参考 MFC 技术注释 64。
// 如果你的控件不遵循 Apartment 模式规则，那么你必须修改下面的代码
// 将第六个参数从 afxRegInsertable | afxRegApartmentThreading
// 变为 afxRegInsertable.
    if (bRegister)
        return AfxOleRegisterControlClass()
```

```
            AfxGetInstanceHandle(),
            m_clsid,
            m_lpszProgID,
            IDS_APARTMENT,
            IDB_APARTMENT,
            afxRegInsertable | afxRegApartmentThreading,
            _dwApartmentOleMisc,
            _tlid,
            _wVerMajor,
            _wVerMinor);
    else
        return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
```

上面的例子演示了如何将代表可插入的标志和代表 Apartment 模式的标志通过或运算组合成第六个参数来调用 AfxOleregisterControlClass 函数。

```
    afxRegInsertable | afxRegApartmentThreading,
```

对于允许的容器，该控件将出现在插入对象对话框中，同时它将会适用 Apartment 模式。适用于 Apartment 模式的控件必须确保静态的类数据通过锁定得到保护，因此当 Apartment 中的控件访问静态数据时，在它结束之前不会被调度程序禁止，相同类的其它对象也不会开始使用相同的静态数据。对静态数据的任何访问都必须间接地通过临界区代码进行。

请参阅

AfxOleRegisterPropertyPageClass, AfxOleRegisterTypeLib, AfxOleUnregisterClass, AfxOleUnregisterTypeLib

AfxOleRegisterPropertyPageClass

BOOL AFXAPI AfxOleRegisterPropertyPageClass(HINSTANCE hInstance, REFCLSID clsid, UINT *idTypeName*, int *nRegFlags*)

#include <afxctl.h>

返回值

如果注册了控件类，则返回非零值，否则返回 0。

参数

hInstance

与属性页类相关的模块的实例句柄。

clsid

属性页的唯一的类 ID。

idTypeName

字符串的资源 ID，**该字符串包含了用户可读的属性页的名字。**

nRegFlags

可能包含标志：

- afxRegApartmentThreading 将注册表中的线程模式设置为 ThreadingModel=Apartment.

注意 在 MFC4.2 以前的 MFC 版本中，没有整型的 nRegFlags 参数。同时 afxRegInsertable 标志对于属性页也是无效的，如果设置了该标志，可能会在 MFC 中引起 ASSERT。

说明

通过 Windows 的注册数据库来注册控件类。这就使得属性页可以被具有 OLE 控件能力的容器所使用。AfxOleRegisterPropertyPageClass 用属性页的名字及其在系统中的位置来更新系统注册表，同时在注册表中设置控件支持的线程模式。有关的更多信息请参见 Visual C++联机文档中的“技术注释 64”，名为“OLE 控件中的 Apartment 模式”，以及《Win32 SDK》中的“进程与线程”。

请参阅 AfxOleRegisterControlClass, AfxOleRegisterTypeLib

AfxOleRegisterServerClass

BOOL AFXAPI AfxOleRegisterServerClass(REFCLSID *clsid*, LPCTSTR *lpszClassName*, LPCTSTR *lpszShortTypeName*, LPCTSTR *lpszLongTypeName*,

OLE_APPTYPE *nAppType* = OAT_SERVER, LPCTSTR* *rglpszRegister* = NULL, LPCTSTR* *rglpszOverwrite* = NULL);

#include <afxdisp.h>

返回值

如果成功注册了服务器类，则返回非零值；否则返回 0。

参数

clsid

对服务器的 OLE **类** ID **的引用。**

lpszClassName

指向包含服务器对象的类名的字符串的指针。

lpszShortTypeName

指向包含服务器对象类型的短名字的字符串的指针，如：“Chart”。

lpszLongTypeName

指向包含服务器对象类型的长名字的字符串的指针，如：“Microsoft Excel 5.0 Chart”。

nAppType

取值于 OLE_APPTYPE 列表中的一个值，指定了 OLE 应用程序的类型，可能的取值如下：

- OAT_INPLACE_SERVER 服务器具有完整的服务器接口。
- OAT_SERVER 服务器只支持嵌入。
- OAT_CONTAINER 容器支持与嵌入的连接。
- OAT_DISPATCH_OBJECT 具有 IDispatch 能力的对象。

rglpszRegister

字符串指针数组，代表 OLE 系统注册表的键和值，当这些键现在没有值时将被加入 OLE 系统注册表。

rglpszOverwrite

字符串指针数组，代表 OLE 系统注册表的键和值，当这些键现在已经有值时将被加入 OLE 系统注册表。

说明

这个函数使你能够在 OLE 系统注册表中注册你自己的服务器。多数应用程序可以使用 COleTemplateServer::Register 来注册应用程序的文档类型。如果你的应用程序的系统注册表格式不符合通常的形式，你可以使用 AfxOleRegisterServerClass 来获得更多的控制。

注册表中包含了一系列的键和值。*rglpszRegister* 和 *rglpszOverwrite* 参数都是字符串指针的数组，每一个字符串都包含了一个键和值，用 NULL 字符（'\n'）隔开。每一个字符串都有可以替换的参数，它们的位置用字符“%1”到“%5”来标记。

符号按照下列方式填充：

符号	取值
%1	类 ID，作为字符串格式化
%2	类名
%3	可执行文件的路径
%4	短类型名
%5	长类型名

请参阅 COleTemplateServer::UpdateRegistry

AfxOleRegisterTypeLib

```
BOOL AfxOleRegisterTypeLib(HINSTANCE hInstance, REFGUID tlid,
        LPCTSTR pszFileNam=NULL,LPCTSTR pszHelpDir = NULL);
```

返回值

如果注册了类型库，则返回非零值；否则返回 0。

参数

hInstance

与类型库相关的应用程序的实例句柄。

tlid

类型库的唯一的 ID。

pszFileName

指向控件的本地化类型库文件（.TLB）的可选文件名。

pszHelpDir

类型库的帮助文件所在目录的名字。如果为 NULL，就假定帮助文件位于与类型库本身相同的目录。

说明

在 Windows 系统注册表中注册类型库，允许其它具有 OLE 控件能力的容器使用类型库。这个函数用类型库的名字及其在系统中的位置来更新注册表。

请参阅 AfxOleUnregisterTypeLib, AfxOleRegisterControlClass, AfxOleUnregisterClass

AfxOleSetEditMenu

void AFXAPI AfxOleSetEditMenu(COleClientItem* *pClient*, CMenu* *pMenu*, UINT *iMenuItem*, UINT *nIDVerbMin*, UINT *nIDVerbMax* = 0, UINT *nIDConvert* = 0);

#include <afxole.h>

参数

pClient

指向客户 OLE 项的指针。

pMenu

指向要更新的菜单对象的指针。

iMenuItem

要更新的菜单项的索引。

nIDVerbMin

响应起始动词的命令 ID。

nIDVerbMax

响应最后的动词的命令 ID。

nIDConvert

Convert 菜单项的 ID。

说明

该函数实现了 *typename* Object命令的用户接口。如果服务器只能识别起始的动词，菜单项就变为“verb *typename* Object”，当用户选择了命令时，将会发送 *nIDVerbMin* 命令。如果服务器能够识别几个命令，菜单项就变为“*typename* Object”，当用户选择命令时，有一个子菜单列出了所有可能的动词。当用户从子菜单中选择了一个动词以后，如果用户选择的是第一个动词，则发送 *nIDVerbMin*，如果选择了第二个动词，则发送 *nIDVerbMin+1*，依次类推。缺省的 COleDocument的实现中自动处理了这个特性。

你必须在客户应用程序的资源描述（.RC）文件中包含下面的语句：

```
#include <afxolecl.rc>
```

请参阅 COleDocument

AfxOleSetUserCtrl

```
void AFXAPI AfxOleSetUserCtrl( BOOL bUserCtrl );
#include <afxdisp.h>
```

参数

bUserCtrl

指定用户控制标志是要被设置还是被清除。

说明

该函数设置或清除用户控制标志，在AfxOleGetUserCtrl的参考中对之有解释。当用户创建或载入一个文档时，框架会调用这个函数，但是不包括文档被间接的动作，比如在容器应用程序中载入一个嵌入对象，打开或创建的情况。

如果你的应用程序中有其它动作需要使用户控制应用程序时，调用这个函数。

请参阅 AfxOleGetUserCtrl

AfxOleTypeMatchGuid

BOOL AfxOleTypeMatchGuid(LPTYPEINFO *pTypeInfo*, TYPEDESC FAR* *pTypeDesc*,
REFGUID *guidType*, ULONG *cIndirectionLevels*);

返回值

如果成功地匹配，则返回非零值；否则返回0。

参数

pTypeInfo

指向类型信息对象的指针，可以从该对象获得 *pTypeDesc*。

pTypeDesc

指向一个 TYPEDESC 结构的指针。

guidType

类型的唯一的 ID。

cIndirectionLevels

间接层次数。

说明

调用这个函数以确定一个类型描述符（从类型信息中获得）是否描述了 *guidType* 指示的类型。

示例

检验 *typedesc* 是否代表一个 IFontDisp 指针：

```
AfxOleTypeMatchGuid( ptypeinfo, &typedesc, IID_IFontDisp, 1);
```

这里 IID_IFontDisp 表明类型类型，间接层次数为 1（因为这个例子是检验一个

简单的指针的）。

AfxOleUnlockApp

```
void AFXAPI AfxOleUnlockApp( );
#include <afxdisp.h>
```

说明

减小框架对应用程序的活动对象的计数。更进一步的信息参见AfxOleLockApp。

当活动对象计数达到0时，就会调用AfxOleOnReleaseAllObjects。

请参阅 AfxOleLockApp, CCmdTarget::OnFinalRelease

AfxOleUnlockControl

```
BOOL AFXAPI AfxOleUnlockControl( REFCLSID clsid );
BOOL AFXAPI AfxOleUnlockControl( LPCTSTR lpszProgID );
#include <afxwin.h>
```

返回值

如果成功地解锁了控件的类工厂，则返回非零值，否则返回0。

参数

clsid
　　控件的唯一的类ID。

lpszProgID
　　控件的唯一的程序ID。

说明

该函数解锁指定控件的类工厂。控件用AfxOleLockControl锁定后，与之相关的动态生成的数据就会保留在内存中。这样可以显著地提高控件的显示速度，因为控件不必在每次显示的时候创建和销毁。当你准备销毁控件的时候，应该调用AfxOleUnlockControl。

请参阅　AfxOleLockControl

AfxOleUnregisterClass

BOOL AFXAPI AfxOleUnregisterClass(REFCLSID *clsID*, LPCSTR *pszProgID*);

返回值

如果成功地注销了控件类或属性页类，则返回非零值。否则返回 0。

参数

clsID

  控件或属性页的唯一的类 ID。

pszProgID

  控件或属性页的唯一的程序 ID。

说明

该函数从 Windows **的注册数据库中移去控件类或属性页类的入口。**

请参阅 AfxOleRegisterPropertyPageClass**，**AfxOleRegisterControlClass**，**AfxOleRegisterTypeLib

AfxOleUnregisterTypeLib

BOOL AFXAPI AfxOleUnregisterTypeLib(REFGUID *tlID*);

返回值

如果成功地注销了类型库，则返回非零值。否则返回 0。

参数

tlID

类型库的唯一的 ID。

说明

调用这个函数以从 Windows **的注册数据库移去类型库的入口。**

请参阅 AfxOleUnregisterClass, AfxOleRegisterTypeLib

AfxParseURL

BOOL AFXAPI AfxParseURL(LPCTSTR *pstrURL*, DWORD& *dwServiceType*, CString& *strServer*, CString& *strObject*, INTERNET_PORT& *nPort*);

返回值

如果成功地解析了 URL**，则返回非零值。如果** URL **为空或它不包含已知的** Internet **服务类型，则为** 0。

参数

pstrURL

指向包含了要解析的 URL 的字符串的指针。

dwServiceType

指明了 Internet 服务的类型。可能的取值如下：

- AFX_INET_SERVICE_FTP
- AFX_INET_SERVICE_HTTP
- AFX_INET_SERVICE_HTTPS
- AFX_INET_SERVICE_GOPHER
- AFX_INET_SERVICE_FILE
- AFX_INET_SERVICE_MAILTO
- AFX_INET_SERVICE_NEWS
- AFX_INET_SERVICE_NNTP
- AFX_INET_SERVICE_TELNET
- AFX_INET_SERVICE_WAIS
- AFX_INET_SERVICE_MID
- AFX_INET_SERVICE_CID
- AFX_INET_SERVICE_PROSPERO
- AFX_INET_SERVICE_AFS
- AFX_INET_SERVICE_UNK

strServer

URL 中服务类型后的第一个部分。

strObject

URL 涉及的对象（可能为空）。

nPort

如果存在，则从 URL 的服务器或对象部分搜索出来。

说明

这个全局函数被用于 CInternetSession::OpenURL。它解析一个 URL 字符串，返回服务的类型以及其它的内容。

例如，AfxParseURL 解析一个如下形式的 URL：service://server/dir/dir/object.ext:port，返回的内容如下：

strServer = = "server"
strObject = = "/dir/dir/object/object.ext"
nPort = = #port
dwServiceType = = #service

注意 为了调用这个函数，你必须在项目中包含 AFXINET.H。

请参阅 AfxGetInternetHandleType

AfxParseURLEx

BOOL AFXAPI AfxParseURLEx(LPCTSTR *pstrURL*, DWORD& *dwServiceType*, CString& *strServer*, CString& *strObject*, INTERNET_PORT& *nPort*, CString& *strUsername*, CString& *strPassword*, DWORD *dwFlags* = 0);

返回值

如果成功地解析了 URL**，则返回非零值。如果** URL **为空或它不包含已知的** Internet **服务类型，则为** 0。

参数

pstrURL

指向包含了要解析的 URL **的字符串的指针。**

dwServiceType

指明了 Internet **服务的类型。可能的取值如下：**

- AFX_INET_SERVICE_FTP
- AFX_INET_SERVICE_HTTP
- AFX_INET_SERVICE_HTTPS
- AFX_INET_SERVICE_GOPHER
- AFX_INET_SERVICE_FILE

- AFX_INET_SERVICE_MAILTO
- AFX_INET_SERVICE_NEWS
- AFX_INET_SERVICE_NNTP
- AFX_INET_SERVICE_TELNET
- AFX_INET_SERVICE_WAIS
- AFX_INET_SERVICE_MID
- AFX_INET_SERVICE_CID
- AFX_INET_SERVICE_PROSPERO
- AFX_INET_SERVICE_AFS
- AFX_INET_SERVICE_UNK

strServer

URL **中服务类型后的第一个部分。**

strObject

URL **涉及的对象（可能为空）。**

nPort

如果存在，则从 URL **的服务器或对象部分搜索出来。**

strUserName

对包含用户名字的 CString **对象的引用。**

strPassword

对包含了用户密码的 CString 对象的引用。

dwFlags

控制 URL 的解析方式的标志。可能是下列值的组合：

取值	含义
ICU_DECODE	把%XX 转义序列转换为字符
ICU_NO_ENCODE	不把不安全的字符转换为转义序列
ICU_NO_META	不把 URL 中的 meta 序列(如“\.”和“\..”)移去
ICU_ENCODE_SPACES_ONLY	仅解码空间
ICU_BROWSER_MODE	不对‘#’和‘?’后面的字符进行编码或解码，并且也不把'?'后面的空白字符移去。如果没有指定这个值，将会对整个 URL 进行编码，后面的空白字符也会被移去

如果你使用 MFC 的缺省条件，则没有设置标志，这个函数将把所有不安全的字符和 meta 序列（如\.，\..和\...）转换为转义序列。

说明

这个全局函数是 AfxParseURL 的扩展版本，被用于 CInternetSession::OpenURL。它解析一个 URL 字符串，返回服务的类型以及其它的内容，同时提供用户的名字和密码。标志指明了如何处理不安全的字符。

注意　为了调用这个函数，你必须在项目中包含 AFXINET.H。

请参阅　AfxGetInternetHandleType

AfxRegisterClass

```
BOOL AFXAPI AfxRegisterClass( WNDCLASS* lpWndClass );
```

返回值

如果成功地注册了这个类，则返回 TRUE，否则返回 FALSE。

参数

lpWndClass

指向一个 WNDCLASS 结构，它包含了要注册的窗口类的信息。关于这个结构的更多的信息，参看 Win32 SDK 文档。

说明

利用这个函数在使用 MFC 的 DLL 中注册一个窗口类。如果你使用了这个函数，则当 DLL 卸载时，这个类会自动注销。

在非 DLL 项目中，AfxRegisterClass 标识符被定义为一个宏，映像到 Windows

函数 RegisterClass，因为应用程序中注册的类是自动注销的。如果你用 AfxRegisterClass 来代替 RegisterClass，那么你的代码不管是用于应用程序还是用于 DLL 都不用于改变。

AfxRegisterWndClass

LPCTSTR AFXAPI AfxRegisterWndClass(UINT *nClassStyle*, HCURSOR *hCursor* = 0, HBRUSH *hbrBackground* = 0, HICON *hIcon* = 0);

返回值

一个以 null 结尾的字符串，其中包含了类名。你可以将这个类名传递给 CWnd 或其派生类的成员函数 Create 以创建一个窗口。这个名字是由微软基础类库生成的。

> 注意　返回值是指向一个静态缓冲区的指针。如果要保存这个字符串，将它赋给一个 CString 变量。

参数

nClassStyle

指定 Windows 的类风格或通过位或（|）操作符生成的风格的组合，用于

窗口类类风格的列表参见 Win32 SDK 文档中的 WNDCLASS 结构。如果这个值为 NULL，缺省的风格如下：

- 将鼠标风格设为 CS_DBLCLICKS，当用户双击鼠标时，将向窗口过程发送双击消息。
- 将鼠标光标风格设为 Windows 的标准风格 IDC_ARROW。
- 将背景刷子设为 NULL，因此窗口将不会擦去它的背景。
- 将图标设为标准的波浪标志的 Windows 徽标。

hCursor

指定了一个鼠标光标资源句柄，将被用于该窗口类所创建的每个窗口。

如果你使用缺省值 0，你将得到标准的 IDC_ARROW 光标。

hbrBackground

指定了一个刷子资源句柄，将被用于该窗口类所创建的每个窗口。

如果你使用缺省值 0，你将获得一个 Null 背景刷子，同时在 WM_ERASEBKGND 进程中，窗口将不会释放其背景。

hIcon

指定了一个图标资源句柄，将被用于该窗口类所创建的每个窗口。
如果你使用缺省值 0，你将得到标准的波浪标志的 Windows 徽标。

说明

微软基础类库自动注册了一些标准的窗口类。如果你希望注册自己的窗口类，可以调用这个函数。

AfxRegisterWndClass 为类注册的名字仅与参数有关。如果你用相同的参数多次调用了 AfxRegisterWndClass，它仅在第一次调用时才注册类。随后用相同的参数进行调用仅简单地返回以及注册的类名。

如果你用相同的参数为多个 CWnd 的派生类调用了 AfxRegisterWndClass，而不是为每个类生成一个独立的窗口类，那么所有的类共用相同的窗口类。如果使用了 CS_CLASSDC 风格，这就会引起问题。你将获得一个 CS_CLASSDC 窗口类，而不是多个 CS_CLASSDC 窗口类，并且使用这个类的所有的 C++窗口都共用相同的 DC。为了避免这个问题，可以调用 AfxRegisterClass 来注册类。

请参阅 CWnd::Create, CWnd::PreCreateWindow, WNDCLASS AfxRegisterClass

AfxSetAllocHook

```
AFX_ALLOC_HOOK AfxSetAllocHook( AFX_ALLOC_HOOK pfnAllocHook );
```

返回值

如果你希望允许分配，则返回非零值。否则返回 0。

参数

pfnAllocHook

指定要调用的函数名。参考关于分配函数的原型的说明。

说明

这个函数设置一个钩子，使每次分配内存之前都会调用一个指定的函数。微软基础类库中的调试内存分配函数能够调用一个用户定义的钩子函数，使用户能够监控内存分配并控制是否允许分配内存。内存分配的钩子函数的原型如下：

BOOL AFXAPI AllocHook(size_t nSize, BOOL bObject, LONG lRequestNumber);

nSize

计划分配的内存大小。

bObject

如果是要为一个 CObject 派生类对象分配内存则为 TRUE，否则为 FALSE。

lRequestNumber

内存分配的系列号。

注意，AFXAPI 调用约定意味着调用者必须从栈中清除参数。

AfxSetResourceHandle

void AfxSetResourceHandle(HINSTANCE *hInstResource*);

参数

hInstResource
应用程序调入资源的.EXE 或.DLL 文件的实例或模块句柄。

说明

使用这个函数来设置 HINSTANCE 句柄，决定应用程序的缺省资源从哪儿载入。

请参阅 AfxGetInstanceHandle, AfxGetResourceHandle

AfxSocketInit

BOOL AfxSocketInit(WSADATA* *lpwsaData* = NULL);

返回值

如果函数成功执行，则返回非零值，否则为 0。

参数

lpwsaData

指向 WSADATA 结构的指针。如果 *lpwsaData* 不等于 NULL，那么调用::WSAStartup 将填充 WSADATA 结构。这个函数同时也保证在应用程序之前调用::WSACleanup。

说明

在你重载的 CWinApp::InitInstance 函数中调用这个函数以初始化 Windows Sockets。

请参阅 CWinApp::InitInstance

AFX_SQL_ASYNC

AFX_SQL_ASYNC(*prs*, *SQLFunc*)

参数

prs

指向 CRecordset 对象或者 CDatabase 对象的指针。从 MFC4.2 开始，这个参数就被忽略。

SQLFunc

一个 ODBC 的 API 函数。有关 ODBC 的 API 函数的更多的信息请参见《ODBC SDK 程序员参考》。

说明

在 MFC4.2 中，这个宏的实现有了变化。现在 AFX_SQL_ASYNC 只是简单地调用 AFX_ODBC_CALL 宏，并且忽略了 *prs* 参数。在 MFC 的以前的版本中，AFX_SQL_ASYNC 被用于调用可能返回 SQL_STILL_EXECUTING 的 ODBC API 函数。如果一个 ODBC 的 API 函数返回了 SQL_STILL_EXECUTING，AFX_SQL_ASYNC 就会调用 prs>*OnWaitForDataSource*。

> **注意** 现在 MFC 的 ODBC 类仅使用同步处理。如果要实现异步操作，你必须调用 ODBC 的 API 函数 SQLSetConnectOption。相关的更多的信息请参考《ODBC SDK 程序员参考》中的主题：“异步执行函数。”

请参阅 AFX_ODBC_CALL, AFX_SQL_SYNC

AFX_SQL_SYNC

AFX_SQL_SYNC(*SQLFunc*)

参数

SQLFunc

一个 ODBC 的 API 函数。关于这些函数的更多的信息，参看《ODBC SDK 程序员参考》。

说明

AFX_SQL_SYNC 宏简单地调用函数 *SQLFunc*。利用这个宏来调用不会返回 SQL_STILL_EXECUTING 的 ODBC API 函数。

在调用 AFX_SQL_SYNC 之前，你必须声明一个 RETCODE 类型的变量 nRetCode。你可以在调用宏之后检查 nRetCode 的值。

注意在 MFC4.2 中 AFX_SQL_SYNC 的实现以及改变了。因为不再需要检查服务器的状态，AFX_SQL_SYNC 简单地将一个值赋给 nRetCode。例如，你不需要进行下面的调用调用：

```
AFX_SQL_SYNC( ::SQLGetInfo( .. ) )
```

而是可以简单地进行赋值：

```
nRetCode = ::SQLGetInfo( .. );
```

请参阅　AFX_SQL_ASYNC, AFX_ODBC_CALL

AfxTermExtensionModule

```
void AFXAPI AfxTermExtensionModule( AFX_EXTENSION_MODULE& state,
        BOOL bAll = FALSE );
```

参数

state

对一个AFX_EXTENSION_MODULE结构的引用，其中包含了扩展DLL模块的状态。

bAll

如果为TRUE，清除所有的扩展DLL模块。否则，只清除当前的DLL模块。

说明

调用这个函数，使每个进程与DLL分离（这在进程退出时发生，或者是由于调用了AfxFreeLibrary，DLL被卸载）的时候MFC可以清除扩展DLL。AfxTermExtensionModule将删除所有与模块相连的局部内容并从消息映射中除去所有的入口。例如：

```
static AFX_EXTENSION_MODULE extensionDLL;
extern "C" int APIENTRY
```

```
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID)
{
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        // 扩展 DLL 的一次性初始化
        if (!AfxInitExtensionModule( extensionDLL, hInstance))
            return 0;
        // 任务：执行其它的初始化工作
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        // 扩展 DLL 的每个进程结束
        AfxTermExtensionModule(extensionDLL);
      // 任务：执行其它的清除工作。
    }
    return 1;    // 完成
}
```

如果你的应用程序动态地载入、释放扩展 DLL，确保调用了 AfxTermExtensionModule。因为大多数扩展 DLL 不是动态载入（通常，它们与其引入库相连接）的，通常没有必要调用 AfxTermExtensionModule。

MFC 的扩展 DLL 需要在它们的 DllMain 中调用 AfxInitExtensionModule。如果

DLL 将输出 CRuntimeClass 对象或者具有它自己的资源，你也需要在 DllMain 中创建 CDynLinkLibrary 对象。

请参阅 AfxInitExtensionModule

AfxThrowArchiveException

void AfxThrowArchiveException(int *cause*, LPCTSTR *lpszArchiveName*);

参数

cause

指定一个整数，代表了异常的原因。其可能取值的列表参见 CArchiveExcepti- on::m_cause。

lpszArchiveName

指向一个字符串，其中包含了引起异常的 CArchive 对象的名字。

说明

抛出一个档案异常。

请参阅 CArchiveException, THROW

AfxThrowDaoException

void AFXAPI AfxThrowDaoException(int *nAfxDaoError* = NO_AFX_DAO_ERROR, SCODE *scode* = S_OK);

参数

nAfxDaoError

一个整数值，代表DAO的扩展错误码，其值可能是CDaoException::m_nAfxDaoError中列出的值之一。

scode

DAO产生的一个SCODE类型的OLE错误码。相关信息参见CdaoException::m_scode。

说明

调用这个函数从你的代码中抛出一个CDaoException类型的异常。框架还调用了AfxThrowDaoException。在你的调用中，你可以传递一个参数，也可以都传递。例如，如果你希望引出CDaoException::nAfxDaoError中定义的一个错误但是并不关心*scode*参数，可以传递一个有效的*nAfxDaoError*参数，同时接受缺省的scode值。

有关与MFC的DAO类相关的异常的更多信息可以参见本书中的CDaoException

类以及《Visual C++程序员指南》中的“异常：数据库异常”。

请参阅 **CException**

AfxThrowDBException

void AfxThrowDBException(RETCODE *nRetCode*, CDatabase* *pdb*, HSTMT *hstmt*);

参数

nRetCode

一个 RETCODE 类型的值，定义了引起被抛出异常的错误的类型。

pdb

指向 CDatabase 对象的指针，代表与异常相关的数据源连接。

hstmt

一个 ODBC 的 HSTMT 句柄，代表与异常相关的语句句柄。

说明

调用这个函数，从你自己的代码中抛出一个 CDBException 类型的异常。如果框架收到了一个 ODBC API 函数的 RETCODE，并且将其解释为异常而不是可

预见的错误，它就调用 AfxThrowDBException。例如，数据访问操作可能因为读磁盘错误而失败。

有关 ODBC 定义的 RETCODE 的信息参见《ODBC SDK 程序员参考》中的第八章“获得状态和错误信息”。关于 MFC 对这些代码的扩展参见 CDBException 类。

请参阅 CDBException::m_nRetCode

AfxThrowFileException

void AfxThrowFileException(int *cause*, LONG l*OsError* = –1, LPCTSTR *lpszFileName* = NULL);

参数

cause

指定了一个整数，它指明了异常的原因。其可能取值的列表参见 CfileExceptio- n::m_cause。

lOsError

包含了操作系统的错误号码（如果有的话），它描述了异常的原因。错误码的列表参见你的操作系统手册。

lpszFileName

指向一个字符串，包含了引起异常的文件的名字。

说明

这个函数抛出一个文件异常。你必须根据操作系统的错误码确定异常的原因。

请参阅 CFileException::ThrowOsError, THROW

AfxThrowInternetException

void AFXAPI AfxThrowInternetException(DWORD *dwContext*, DWORD *dwError* = 0);

参数

dwContext

引起错误的操作的环境标识符。*dwContext* 的缺省值由 CInternetSession 指定，并被传递给 CInternetConnection 和 CInternetFile 的派生类。对于针对连接或文件的特定操作，你通常会用自己的值覆盖缺省的 dwContext 值。这个值将被返回到 CinternetSession:: OnStatusCallback 以标识特定的操作状态。有关环境标识符的更多的信息请参见《Visual C++程序员指南》中的文章：“Internet 第一步：WinInet”。

dwError

引起异常的错误。

说明

这个函数抛出一个 Internet 异常。你必须根据操作系统的错误码来确定异常的原因。

注意 为了调用这个函数，你必须在项目中包含 AFXINET.H。

请参阅 CInternetException, THROW

AfxThrowMemoryException

void AfxThrowMemoryException( );

说明

这个函数抛出一个内存异常。如果调用系统的内存分配函数（例如 malloc 以及 Windows 函数 GlobalAlloc）失败了，就调用这个函数。你不需要为 new 调用这个函数，因为如果内存分配失败，new 会自动抛出一个内存异常。

请参阅 CMemoryException, THROW

AfxThrowNotSupportedException

void AfxThrowNotSupportedException( );

说明

抛出的异常是请求不支持的特性的结果。

请参阅 CNotSupportedException, THROW

AfxThrowOleDispatchException

void AFXAPI AfxThrowOleDispatchException(WORD *wCode*, LPCSTR *lpszDescription*, UINT *nHelpID* = 0);
void AFXAPI AfxThrowOleDispatchException(WORD *wCode*, UINT *nDescriptionID*, UINT *nHelpID* = –1);

#include <afxdisp.h>

参数

wCode

你的应用程序特有的错误码。

lpszDescription

错误的语言描述。

nDescriptionID

错误的语言描述的资源 ID。

nHelpID

你的应用程序的帮助文件（.HLP）的帮助环境。

说明

使用这个函数在一个 OLE 自动化函数内部抛出一个异常。为这个函数提供的信息可以被控制应用程序（Microsoft Visual Basic 或其它 OLE 自动化的客户程序）所显示。

请参阅 COleException

AfxThrowOleException

```
void AFXAPI AfxThrowOleException( SCODE sc );
void AFXAPI AfxThrowOleException( HRESULT hr );
```

```
#include <afxdisp.h>
```

参数

sc

指明异常原因的 OLE 状态代码。

hr

指明异常原因的结果代码的句柄。

说明

这个函数生成一个 COleException 类型的对象并抛出一个异常。接收 HRESULT 类型的参数的版本将结果代码转换为对应的 SCODE。有关 HRESULT 和 SCODE 的更多的信息参见《Platform SDK》中的“COM 错误代码的结构”。

请参阅 COleException, THROW

AfxThrowResourceException

```
void AfxThrowResourceException( );
```

说明

这个函数抛出一个资源异常。通常在不能载入 Windows 资源的时候调用这个函数。

请参阅 CResourceException, THROW

AfxThrowUserException

```
void AfxThrowUserException( );
```

说明

这个函数抛出一个异常，结束一个终止用户操作。通常在 AfxMessageBox **向用户报告了一个错误之后会立即调用这个函数。**

请参阅 CUserException, THROW, AfxMessageBox

afxTraceEnabled

```
BOOL afxTraceEnabled;
```

说明

这是一个全局变量，用于打开或关闭 TRACE **宏的输出。**

在缺省情况下，TRACE **宏的输出是被关闭的。如果你希望你的程序中的** TRACE **宏产生输出，则应将** afxTraceEnabled **设置为非零值。**

通常，afxTraceEnabled 是在你的 AFX.INI 设置的。与此同时，你可以通过 TRACER.EXE 工具设置 afxTraceEnabled 的值。更多的信息请参考“技术注释 7”。

请参阅 afxTraceFlags, TRACE

afxTraceFlags

int afxTraceFlags;

说明

用于打开微软基础类库的内建报告特性。

这个变量可以通过程序来设置也可以在调试器中设置。afxTraceFlags 的每一位选择了一个跟踪报告选项。你可以利用 TRACER.EXE 打开或关闭这些位中的任意一个。没有必要手动设置这些位。

下面是这些位模板以及相应的跟踪报告特性的列表：

- 0x01 多应用程序调试。这将会在每个 TRACE 输出之前加上应用程序的名字作为前缀，它既会影响你的应用程序的 TRACE 输出，也会影响下面描述的附加的报告特性。
- 0x02 主消息泵。报告在主 CWinApp 消息处理机制中接收到的每个消息。

列出了窗口句柄，消息的名字或号码，wParam 以及 lParam。

报告发生在对 Windows 函数 GetMessage 的调用之后，但是在任何对消息的翻译或分发之前。

动态数据交换消息将显示一些额外的数据，可以被 OLE 中的一些调试方式所使用。

这个标志仅显示接收到的消息，而不包括那些发出的消息。

- 0x04 与上面的 0x02 选项类似的主消息调度，但是适用于在 CWnd::WindowProc 中分发的消息，并且对接收到的和发送出去的消息都作处理。
- 0x08 WM_COMMAND 消息调度。一个特殊的情况，用于扩展 WM_COMMAND/On-Command 处理，报告命令路径机制的进度。

  同时报告哪个类接收了命令（如果有匹配的消息映射入口），以及什么时候类不接收命令（如果没有匹配的消息映射入口）。在跟踪多文档界面（MDI）应用程序的命令消息流的时候，这个报告特别有用。

- 0x10 OLE 跟踪。报告重要的 OLE 通知或请求。

  为 OLE 服务器或客户打开这个选项以跟踪 OLE DLL 和 OLE 应用程序之间的通讯。

- 0x20 数据库跟踪。报告 ODBC 类和 DAO 类的警告，对于 DAO 类还有

一些附加信息。如果你希望跟踪 MFC 的 ODBC 类和 DAO 类，则打开这个选项。对于 ODBC，你只能得到警告，例如在 DFX 调用中发生的类型不匹配。对于 DAO，你可以得到所有异常的信息，包括发生异常的 DAO 或 MFC 的 DAO 类中的线路和函数。

相关的更多的信息参见《Visual C++文档》中的“技术注释 7”。

请参阅 afxTraceEnabled, TRACE

AfxVerifyLicFile

BOOL AFXAPI AfxVerifyLicFile(HINSTANCE *hInstance*,LPCTSTR *pszLicFileName*,LPOLE- STR *pszLicFileContents*, UINT *cch* = –1);

返回值

如果存在许可文件并且是以 *pszLicFileContents* 中的字符序列开始的，则返回非零值；否则返回 0。

参数

hInstance

与许可控制相关的 DLL 的实例句柄。

pszLicFileName

指向一个以 null **结尾的字符串，包含了许可文件的名字。**

pszLicFileContents

指向一个字节序列，它必须与许可文件开头的序列匹配。

cch

pszLicFileContents **中的字符数目。**

说明

调用这个函数以检验由 *pszLicFileName* **指定文件名的许可文件是否对** OLE **控件有效。如果** cch **为－**1**，这个函数将使用：**
_tcslen(pszLicFileContents)
请参阅 COleObjectFactory::VerifyUserLicense

AfxWinInit

BOOL AFXAPI AfxWinInit(HINSTANCE *hInstance*, HINSTANCE *hPrevInstance*, LPTSTR *lpCmdLine*, int *nCmdShow*)

参数

hInstance

当前运行模块的句柄。

hPrevInstance

应用程序前一个实例的句柄。对于基于 Win32 的应用程序，这个参数总是 NULL。

lpCmdLine

指向一个以 null 结尾的字符串，指定了应用程序的命令行。

nCmdShow

指定了 GUI 应用程序的主窗口将如何显示。

说明

这个函数是由 MFC 提供的 WinMain 函数调用的，它是基于 GUI 的应用程序的 CWinApp 初始化的一部分，主要用来初始化 MFC。对于控制台应用程序，不使用 MFC 提供的 WinMain 函数，你必须直接调用 AfxWinInit 来初始化 MFC。

如果你自己调用 AfxWinInit，你必须声明一个 CWinApp 类的实例。对于控制台应用程序，你可以选择不从 CWinApp 基础你自己的类，而是直接使用 CWinApp 的实例。如果你决定在你的 main 函数的实现中将所有的工作留给你的应用程序，这样做是可以的。

例子程序 TEAR 演示了如何用 MFC 生成一个控制台应用程序。

示例

```
// this file must be compiled with the /GX and /MT options:
//            cl /GX /MT thisfile.cpp
#include <afx.h>
#include <afxdb.h>
#include <iostream.h>
int main()
{
    // 试图初始化 MFC
    if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
    {
        cerr << "MFC failed to initialize!" << endl;
        return 1;
    }
    // 试图与并不存在的 ODBC 数据库连接
    // （如果不初始化 MFC，这根本不会起作用。）
    CDatabase db;
    try
    {
        db.Open("This Databsae Doesn't Exist");
        // 我们不会真正到达这里。
```

```
        cout << "Successful!" << endl;
        cout << "Closing ... ";
        db.Close();
        cout << "Closed!" << endl;
    }
    catch (CDBException* pEx)
    {
        // 我们到达了一个异常。输出错误信息。
        // （如果不初始化 MFC，这根本不会起作用。）
        char sz[1024];
       cout << "Error: ";
        if (pEx->GetErrorMessage(sz, 1024))
            cout << sz;
        else
            cout << "No error message was available";
        cout << endl;
        pEx->Delete();
        return 1;
    }
    return 0;
}
```

请参阅 CWinApp, CWinApp: The Application Class, main, WinMain

AND_CATCH

AND_CATCH(*exception_class*, *exception_object_pointer_name*)

参数

exception_class
　　指定了要检测的异常类型。标准的异常类的列表参见 CException 类。

exception_object_pointer_name
　　这个宏将生成的异常对象的指针的名字。你可以使用这个指针名在 AND_CATCH 块中访问异常对象。这个变量是为你定义的。

说明

定义了一个代码块，用于捕捉前面的 TRY 块中抛出的附加的异常类型。使用 CATCH 宏来捕捉一个异常类型，然后用 AND_CATCH 宏来捕捉后面的类型。用 END_CATCH 宏来结束 TRY 块。

异常处理代码能够检验异常对象以获得有关异常原因的更多信息。在 AND_CATCH 块内调用 THROW_LAST 宏，这样可以将处理权移交给下一个

异常框架。AND_CATCH 标记了前面的 CATCH 或 AND_CATCH 块的结束。

> **注意** AND_CATCH 块被定义为一个 C++作用域（用大括号分界）。如果你在这个作用域内定义了一个变量，记住它们仅能在此作用域内访问。这也适用于 *exception_object_pointer_name* 变量。

请参阅 TRY, CATCH, END_CATCH, THROW, THROW_LAST, AND_CATCH_ALL, Cexception

AND_CATCH_ALL

AND_CATCH_ALL(*exception_object_pointer_name*)

参数

exception_object_pointer_name
这个宏将生成的异常对象的指针的名字。你可以使用这个指针名在 AND_CATCH_ALL 块中访问异常对象。这个变量是为你定义的。

说明

定义了一个代码块，用于捕捉前面的 TRY 块中抛出的附加的异常类型。使用 CATCH 宏来捕捉一个异常类型，然后用 AND_CATCH_ALL 宏来捕捉所有其

它类型。如果你适用了 AND_CATCH_ALL，用 END_CATCH_ALL 宏来结束 TRY 块。

异常处理代码能够检验异常对象以获得有关异常原因的更多信息。在 AND_CATCH_ALL 块内调用 THROW_LAST 宏，这样可以将处理权移交给下一个异常框架。AND_CATCH_ALL 标记了前面的 CATCH 或 AND_CATCH_ALL 块的结束。

> **注意** AND_CATCH_ALL 块被定义为一个 C++作用域（用大括号分界）。如果你在这个作用域内定义了一个变量，记住它们仅能在此作用域内访问。

请参阅

TRY,CATCH_ALL,END_CATCH_ALL,THROW,THROW_LAST,AND_CATCH,Cexception

ASSERT

ASSERT(*booleanExpression*)

参数

booleanExpression

指定了一个表达式（包含指针变量），其计算结果为非零值或 0。

说明

这个宏计算它的参数。如果结果为 0，**则输出一个调试信息并退出程序。如果结果为非零值，它什么也不做。**

诊断信息具有如下形式：

assertion failed in file <name> in line <num>

这里的 *name* **是源文件的名字，** *num* **是源文件中断言失败位置的行号。**

在 MFC **的发行版本中，** ASSERT **并不计算表达式的值，因而也不会中断程序。如果不管环境如何，表达式都必须被计算，用** VERIFY **宏来代替** ASSERT。

注意 这个函数仅在 MFC 的调试版本中有效。

示例

```
// ASSERT 的例子
CAge* pcage = new CAge( 21 ); // CAge is derived from CObject.
ASSERT( pcage!= NULL )
ASSERT( pcage->IsKindOf( RUNTIME_CLASS( CAge ) ) )
// 仅当 pcage 不是一个 CAge*指针时才结束程序。
```

请参阅 VERIFY

ASSERT_KINDOF

ASSERT_KINDOF(*classname*, *pobject*)

参数

classname

CObject 派生类的名字。

pobject

类对象的指针。

说明

这个宏断言指向的对象属于指定的类，或者属于从指定的类继承的类。*pobject* 参数应该是指向一个对象的指针，可以是 const 类型的。指向的对象和类必须支持 CObject 运行时类信息。作为一个例子，要确定 pDocument 是否是 CMyDocument 类或者它的任何派生类的对象的指针，你可以这么写：

```
ASSERT_KINDOF(CMyDocument, pDocument)
```

使用 ASSERT_KINDOF 宏的作用与下述代码完全相同：

```
ASSERT(pobject->IsKindOf(RUNTIME_CLASS(classname)));
```

这个函数仅对用 DECLARE_DYNAMIC 或 DECLARE_SERIAL 宏声明的类起作用。

注意　这个函数仅在 MFC 的调试版本中才有。

请参阅　ASSERT

ASSERT_VALID

ASSERT_VALID(pObject)

参数

pObject

指定了从 CObject 继承的类的对象，它具有 AssertValid 成员函数的重载版本。

说明

这个函数用于检验你对对象内部状态的有效性的假定。ASSERT_VALID 调用作为参数传递给它的对象的 AssertValid 成员函数。

在 MFC 的发行版本中，ASSERT_VALID 什么也不做。在调试版本中，它检验一个指针是否为 NULL，并且调用对象自己的 AssertValid 成员函数。如果这些测试中有些失败了，它就按照与 ASSERT 相同的方式显示一个警告信息。

注意　这个函数仅在 MFC 的调试版本中才有。

有关的更多信息以及示例参见《Visual C++程序员指南》中的"MFC 调试支持"。

请参阅　ASSERT, VERIFY, CObject, CObject::AssertValid

BASED_CODE

说明

在 Win32 下，这个宏展开后实际为空，主要是为向后兼容性提供的。在 16 位的 MFC 中，这个宏确保数据将被放在代码段而不是数据段中。其结果是你的

数据段不太紧凑了。

BEGIN_CONNECTION_MAP

BEGIN_CONNECTION_MAP(*theClass*, *theBase*)

参数

theClass
　　指定了连接映射所属的控件类的名字。

theBase
　　指定了 *theClass* 的基类的名字。

说明

在你的程序中每个从 COleControl 继承的类都能为你的控件支持的连接点提供一个连接映射。在你的类成员函数的实现文件（.CPP）中，用 BEGIN_CONNECTION_MAP 宏开始连接映射的定义，然后用 CONNECTION_PART 宏为每个连接点加入一个入口。最后，用 END_CONNECTION_MAP 宏结束连接映射。

请参阅　BEGIN_CONNECTION_PART, DECLARE_CONNECTION_MAP

BEGIN_CONNECTION_PART

BEGIN_CONNECTION_PART(*theClass*, *localClass*)

参数

theClass
　　指定了连接点所属的控件类的名字。

localClass
　　指定了实现连接点的本地类的名字。

说明

利用 BEGIN_CONNECTION_PART 宏来开始除事件和属性通知连接点之外的其它类型的连接点定义。
在你的类成员函数的声明文件（.H）中，用 BEGIN_CONNECTION_PART 宏来开始连接点定义，然后加入 CONNECTION_IID 宏以及你想实现的其它成员函数，最后用 END_CONNECTION_PART 宏结束连接点映射。

请参阅　BEGIN_CONNECTION_MAP, END_CONNECTION_PART, DECLARE_CONNECTION_MAP

BEGIN_DISPATCH_MAP

BEGIN_DISPATCH_MAP(*theClass*, *baseClass*)

#include <afxdisp.h>

参数

theClass

指定拥有调度映射的类的名字。

baseClass

指定 *theClass* **的基类的名字。**

说明

使用 BEGIN_DISPATCH_MAP **宏来声明你的调度映射定义。**

在你的类的成员函数的实现文件（.CPP**）中，用** BEGIN_DISPATCH_MAP **宏开始调度映射，为每个调度函数和属性加入一个入口，最后用** END_DISPATCH_MAP **宏结束调度映射。**

请参阅 Dispatch Maps, DECLARE_DISPATCH_MAP, END_DISPATCH_MAP, DISP_FUNCTION, DISP_PROPERTY, DISP_PROPERTY_EX, DISP_DEFVALUE

BEGIN_EVENT_MAP

BEGIN_EVENT_MAP(*theClass*, *baseClass*)

参数

theClass
指定事件映射所属的控件类的名字。

baseClass
指定 *theClass* 的基类的名字。

说明

使用 BEGIN_EVENT_MAP 宏开始你的事件映射的定义。

在你的类的成员函数的实现文件（.CPP）中，使用 BEGIN_EVENT_MAP 宏开始事件映射，然后为每个事件加入一个入口，最后用 END_EVENT_MAP 宏结束事件映射。

关于事件映射和 BEGIN_EVENT_MAP 宏的更多信息参见《Visual C++文档》中的“ActiveX 控件：事件”。

请参阅 DECLARE_EVENT_MAP, END_EVENT_MAP

BEGIN_EVENTSINK_MAP

BEGIN_EVENTSINK_MAP(*theClass*, *baseClass*)

参数

theClass

　　指定了事件接收映射所属的控件类的名字。

baseClass

　　指定了 *theClass* **的基类的名字。**

说明

使用 BEGIN_EVENTSINK_MAP **宏开始你的事件接收映射的定义。**

在你的类的成员函数的实现文件（.CPP**）中，使用** BEGIN_EVENTSINK_MAP **宏开始事件映射，然后为每个要识别的事件加入一个入口，最后用** END_EVENTSINK_MAP **宏结束事件接收映射。**

关于事件接收映射和 OLE **控件容器的更多信息参见《**Visual C++**程序员指南》中的“** ActiveX **控件容器”。**

请参阅 DECLARE_EVENTSINK_MAP, END_EVENTSINK_MAP

BEGIN_MESSAGE_MAP

BEGIN_MESSAGE_MAP(*theClass*, *baseClass*)

参数

theClass

指定消息映射所属的类的名字。

baseClass

指定 *theClass* **的基类的名字。**

说明

使用 BEGIN_MESSAGE_MAP **宏开始你的消息映射的定义。**

在你的类的成员函数的实现文件（.CPP**）中，使用** BEGIN_MESSAGE_MAP **宏开始消息映射，然后为每个消息处理函数加入一个入口，最后用** END_MESSAGE_MAP **宏结束消息映射。**

关于消息映射和 BEGIN_MESSAGE_MAP **的更多信息参见《**Visual C++**教程》中的“加入对话框”。**

示例

```
// BEGIN_MESSAGE_MAP 示例
BEGIN_MESSAGE_MAP( CMyWindow, CFrameWnd )
    //{{AFX_MSG_MAP( CMyWindow )
    ON_WM_PAINT()
    ON_COMMAND( IDM_ABOUT, OnAbout )
   //}}AFX_MSG_MAP
END_MESSAGE_MAP( )
```

请参阅 DECLARE_MESSAGE_MAP, END_MESSAGE_MAP

BEGIN_OLEFACTORY

BEGIN_OLEFACTORY(*class_name*)

参数

class_name

指定了类工厂所属的控件类的名字。

说明

在控件类的头文件中，用 BEGIN_OLEFACTORY 宏开始你的类工厂的定义。在 BEGIN_OLEFACTORY 后面必须立即开始类工厂许可函数的定义。

请参阅 END_OLEFACTORY, DECLARE_OLECREATE_EX

BEGIN_PARSE_MAP

BEGIN_PARSE_MAP(*theClass*, *baseClass*)

参数

theClass
　　指定拥有这个解析映射的类的名字。

baseClass
　　指定 *theClass* 的基类的名字。必须是从 CHttpServer 继承的类。

说明

使用 BEGIN_PARSE_MAP 宏来开始你的解析映射的定义。

当 CHttpServer 对象接收到一个客户命令时，解析映射就将此命令与其类成员

函数和参数关联起来。对于每个 CHttpServer 对象仅有一个解析映射。

在你的类成员函数的实现文件（.CPP）中，用 BEGIN_PARSE_MAP 宏开始一个解析映射，然后为你的每一个解析函数和属性加入一个入口，最后用 END_PARSE_MAP 宏来结束解析映射。

有关解析映射的例子参见 ON_PARSE_COMMAND。

请参阅 ON_PARSE_COMMAND, ON_PARSE_COMMAND_PARAMS, DEFAULT_PARSE_COMMAND, END_PARSE_MAP, CHttpServer

BEGIN_PROPPAGEIDS

BEGIN_PROPPAGEIDS(*class_name*, *count*)

参数

class_name

指定了属性页的控件类的名字。

count

控件类使用的属性页的数目。

说明

使用 BEGIN_PROPPAGEIDS 宏来开始你的控件的属性页 ID 列表的定义。

在你的类成员函数的实现文件（.CPP）中，用 BEGIN_PROPPAGEIDS 宏来开始属性页列表，然后为你的每一个属性页加入一个入口，最后用 END_PROPPAGEIDS 宏来结束属性页列表。

有关属性页的更多信息参见《Visual C++程序员指南》中的文章："ActiveX 控件：属性页"。

请参阅 END_PROPPAGEIDS, DECLARE_PROPPAGEIDS, PROPPAGEID

CATCH

CATCH(*exception_class*, *exception_object_pointer_name*)

参数

exception_class
指定了要检测的异常类型。有关标准异常类的列表参见 CException 类。

exception_object_pointer_name
指定了这个宏将创建的异常对象指针的名字。你可以在 CATCH 块内使

用这个指针名来访问异常对象。这个变量是为你定义的。

说明

使用这个宏来定义一个代码块，它将捕捉前面的 TRY 块抛出的第一个异常类型。异常处理代码能够检验异常对象以获得有关异常原因的详细信息。调用 THROW_LAST 宏可以将处理权移交给下一个异常框架。用 END_CATCH 来结束一个 TRY 块。

如果 *exception_class* 是 CException 类，那么所有的异常类型都会被捕捉。你可以使用 CObject::IsKindOf 成员函数来决定抛出什么类型的异常。捕捉几个类型的异常的更好的方式是使用一系列的 AND_CATCH 语句，每个针对一个不同的异常类型。

异常对象指针是由宏生成的。你不必自己声明它。

> 注意 CATCH 块被定义为一个 C++作用域（用大括号分界）。如果你在这个作用域内声明了变量，记住它只能在作用域内部访问。这也适用于 exception_object_point- er_name。

关于异常和 CATCH 宏的更多信息参见《Visual C++程序员指南》中的文章："异常"。

请参阅 TRY, AND_CATCH, END_CATCH, THROW, THROW_LAST, CATCH_ALL, CException

CATCH_ALL

CATCH_ALL(*exception_object_pointer_name*)

参数

exception_object_pointer_name

指定了这个宏生成的异常对象指针的名字。你可以在 CATCH_ALL **块中使用这个指针名来访问异常对象。这个变量是为你定义的。**

说明

使用这个宏来定义一块代码，它捕捉前面的 TRY **块中抛出的所有的异常类型。异常处理代码能够检验异常对象以获得关于异常原因的详细信息。调用** THROW_LAST **宏可以将处理权移交给下一个异常框架。如果你使用了** CATCH_ALL**，用** END_CATCH_ALL **宏来结束** TRY **块。**

注意 CATCH_ALL块被定义为一个C++作用域（用大括号分界）。如果你在此作用域内定义了一个变量，记住你只能在这个作用域内部访问它。

有关异常的更多的信息参见《Visual C++**程序员指南》中的文章：“异常”。**

请参阅 TRY, AND_CATCH_ALL, END_CATCH_ALL, THROW, THROW_LAST, CATCH, CException

CompareElements

template< class *TYPE*, class *ARG_TYPE* >
BOOL AFXAPI CompareElements(const *TYPE** *pElement1*,const *ARG_TYPE** *pElement2*);

返回值

如果 *pElement1* **所指向的对象与** *pElement2* **所指向的对象相等，则返回非零值。否则返回** 0。

参数

TYPE

要比较的第一个元素的类型。

pElement1

指向要比较的第一个元素的指针。

ARG_TYPE

要比较的第二个元素的类型。

pElement2

指向要比较的第二个元素的指针。

说明

这个函数由 CList::Find 直接调用，CMap::Lookup 和 CMap::operator[]会间接调用它。CMap 的调用使用了 CMap 的模板参数 *KEY* 和 *ARG_KEY*。

缺省的实现返回对**pElement1* 和**pElement2* 进行比较的结果。你可以重载这个函数以便它以一种适用于你的应用程序的方式进行比较。

C++语言为简单类型（char，int，float 等等）定义了比较操作符（==），但是没有为类和结构定义比较操作符。如果你想使用 CompareElements 或演示一个使用它的集合类，你就要定义一个比较操作符或者重载 CompareElements 以返回正确的类型。

请参阅 CList, CMap

ConstructElements

```
template< class TYPE >
void AFXAPI ConstructElements( TYPE* pElements, int nCount );
```

参数

TYPE

指定了要构造的元素的类型的模板参数。

pElements

指向元素的指针。

nCount

要构造的元素的数目。

说明

这个函数在构造新的数组、列表或映射时被调用。缺省的版本将新元素的所有位初始化为 0。

关于这个函数以及其它帮助函数的信息参见《Visual C++程序员指南》中的文章“集合：如何生成类型安全的集合”。

请参阅 CArray, CList, CMap

CopyElements

```
template< class TYPE >
void AFXAPI CopyElements( TYPE* pDest, const TYPE* pSrc, int nCount );
```

参数

TYPE

　　指定了要复制的元素类型的模板参数。

pDest

　　指向复制元素的目的的指针。

pSrc

　　指向要复制的元素源的指针。

nCount

　　要复制的元素的数目。

说明

这个函数由 CArray::Append **和** CArray::Copy **直接调用。缺省实现使用单一指定的操作符（**=**）来执行复制操作。如果被复制的类型没有过载操作符**=**，那么缺省的实现按位进行复制。**

关于这个函数以及其它帮助函数的实现参见《Visual C++**程序员指南》中的文章“如何生成类型安全的集合”。**

请参阅 CArray

CONNECTION_IID

CONNECTION_IID(*iid*)

参数

iid

连接点调用的接口的 ID。

说明

在 BEGIN_CONNECTION_PART 和 END_CONNECTION_PART 之间使用 CONNECTION- _IID 宏来为你的 OLE 控件支持的连接点定义接口 ID。*iid* 参数是用于识别连接点对其连接的接收方调用的接口的 ID。例如：

CONNECTION_IID(IID_ISinkInterface)

指定了调用 ISinkInterface 接口的连接点。

请参阅 BEGIN_CONNECTION_PART, DECLARE_CONNECTION_MAP, END_CONNECTION_PART

CONNECTION_PART

CONNECTION_PART(*theClass*, *iid*, *localClass*)

参数

theClass
　　指定了连接点所属的控件类的名字。

iid
　　连接点调用的接口的 ID。

localClass
　　指定了实现连接点的本地类的名字。

说明

使用 CONNECTION_PART **宏来把你的** OLE **控件的连接映射到一个指定的接口** ID。
例如：

```
BEGIN_CONNECTION_MAP(CSampleCtrl, COleControl)
CONNECTION_PART(CSampleCtrl, IID_ISinkInterface, MyConnPt)
END_CONNECTION_MAP()
```

实现了一个连接映射，其中有一个连接点，它调用了 IID_ISinkInterface 接口。

请参阅 BEGIN_CONNECTION_PART, DECLARE_CONNECTION_MAP, BEGIN_CONNECTION_MAP, CONNECTION_IID

DDP_CBIndex

void AFXAPI DDP_CBIndex(CDataExchange* *pDX*, int *id*, int& *member*, LPCTSTR *pszPropName*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。

id

与 *pszPropName* 指定的控件属性相关的组合框控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的组合框控件交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange **函数中调用这个函数以使整数属性的值与属性页中组合框的当前选择的索引同步。这个函数必须在相应的** DDX_CBIndex **函数之前调用。**

请参阅 DDP_CBString, DDP_Text, COleControl::DoPropExchange, DDX_CBIndex

DDP_CBString

void AFXAPI DDP_CBString(CDataExchange* *pDX*, int *id*, CString& *member*, LPCTSTR *pszPropName*);

参数

pDX
指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。**

id
与 *pszPropName* **指定的控件属性相关的组合框控件的资源** ID**。**

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的组合框控件交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使字符串属性的值与属性页中组合框的当前选择同步。这个函数必须在相应的 DDX_CBCString 函数之前调用。

请参阅 DDP_CBStringExact, DDP_CBIndex, COleControl::DoPropExchange, DDX_CBString

DDP_CBStringExact

void AFXAPI DDP_CBStringExact(CDataExchange* *pDX*, int *id*, CString& *member*, LPCTSTR *pszPropName*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据

交换的环境，包括它的方向。

id

与 *pszPropName* 指定的控件属性相关的组合框控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的组合框控件交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使属性页中组合框的当前选择和与之完全匹配的字符串属性的值同步。这个函数必须在相应的 DDX_CBCStringExtract 函数之前调用。

请参阅 DDP_CBString, DDP_CBIndex, COleControl::DoPropExchange, DDX_CBStringExact

DDP_Check

void AFXAPI DDP_Check(CDataExchange* *pDX*, int *id*,int &*member*,LPCSTR *pszPropName*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。

id

与 pszPropName 指定的控件属性相关的复选框控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的复选框控件交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使属性的值与相关属性页的复选框同步。这个函数必须在相应的 DDX_Check 函数之前调用。

请参阅 DDP_Radio, DDP_Text, COleControl::DoPropExchange, DDX_Check

DDP_LBIndex

Void AFXAPI DDP_LBIndex(CDataExchange* *pDX*, int *id*, int& *member*, LPCTSTR

pszPropName);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。

id

与 *pszPropName* 指定的控件属性相关的列表框控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的列表框字符串交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使整数属性的值与属性页中列表框的当前选择的索引同步。这个函数必须在相应的 DDX_LBIndex 函数之前调用。

请参阅 DDP_LBString, DDP_CBIndex, COleControl::DoPropExchange, DDX_LBIndex

DDP_LBString

void AFXAPI DDP_LBString(CDataExchange* *pDX*, int *id*, CString& *member*, LPCTSTR *pszPropName*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。

id

与 *pszPropName* 指定的控件属性相关的列表控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的列表框字符串交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使整数属性的值与属性页中列表框的当前选择同步。这个函数必须在相应的 DDX_LBString 函数之

前调用。

请参阅

DDP_LBStringExact,DDP_CBIndex,COleControl::DoPropExchange,DDX_LB-String

DDP_LBStringExact

void AFXAPI DDP_LBStringExact(CDataExchange* *pDX*, int *id*, CString& *member*, LPCTSTR *pszPropName*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。

id

与 *pszPropName* 指定的控件属性相关的列表框控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的列表框字符串交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使属性页中列表框的当前选择和与之完全匹配的字符串属性的值同步。这个函数必须在相应的 DDX_LBStringExact 函数之前调用。

请参阅 DDP_LBString, DDP_LBIndex, COleControl::DoPropExchange, DDX_LBStringExact

DDP_PostProcessing

void AFXAPI DDP_PostProcessing(CDataExchange **pDX*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数，使得当属性值被存储

以后停止在属性页到控件之间的数据交换。

这个函数必须在所有的数据交换函数都结束以后调用。例如：

```
void CSamplePage::DoDataExchange(CDataExchange* pDX)
{
    //{{AFX_DATA_MAP(CSpindialPropPage)
    DDP_Text(pDX, IDC_POSITIONEDIT, m_NeedlePosition,
                _T("NeedlePosition"));
    DDX_Text(pDX, IDC_POSITIONEDIT, m_NeedlePosition);
    DDV_MinMaxInt(pDX, m_NeedlePosition, 0, 3);
    //}}AFX_DATA_MAP
    DDP_PostProcessing(pDX);
}
```

请参阅 COleControl::DoPropExchange

DDP_Radio

void AFXAPI DDP_Radio(CDataExchange**pDX*, int *id*, int &*member*, LPCTSTR *pszPropName*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。

id

与 *pszPropName* 指定的控件属性相关的单选按钮控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的单选按钮控件交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使属性值与属性页中的单选按钮控件同步。这个函数必须在相应的 DDX_Radio 函数之前调用。

请参阅 DDP_Check, DDP_Text, COleControl::DoPropExchange, DDX_Radio

DDP_Text

void AFXAPI DDP_Text(CDataExchange**pDX*, int *id*, BYTE &*member*,

LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, int &*member*,
LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, UINT &*member*,
LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, long &*member*,
LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, DWORD &*member*,
LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, float &*member*,
LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, double &*member*,
LPCTSTR *pszPropName*);
void AFXAPI DDP_Text(CDataExchange* *pDX*, int *id*, CString &*member*,
LPCTSTR *pszPropName*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括它的方向。**

id

与 *pszPropName* 指定的控件属性相关的控件的资源 ID。

member

与 *id* 指定的属性页控件和 *pszPropName* 指定的属性相关的成员变量。

pszPropName

将要与 *id* 指定的控件交换数据的控件属性的名字。

说明

在你的属性页的 DoDataExchange 函数中调用这个函数以使属性值与属性页控件同步。这个函数必须在相应的 DDX_Text 函数之前调用。

请参阅 DDP_Check, DDP_Radio, COleControl::DoPropExchange, DDX_Text

DDV_MaxChars

void AFXAPI DDV_MaxChars(CDataExchange* *pDX*, CString const& *value*, int *nChars*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对对话框、表格视或控件视对象的成员变量的引用，将对其进行数据校验。

nChars

允许的最大字符数目。

说明

调用 DDV_MaxChars 以检验与 *value* 相关的控件的字符数是否超过 *nChars*。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxByte

void AFXAPI DDV_MinMaxByte(CDataExchange* *pDX*, BYTE *value*, BYTE *minVal*, BYTE *maxVal*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对对话框、表格视或控件视对象的成员变量的引用，将对其进行数据校验。

minVal

允许的最小值（BYTE 类型）。

maxVal

允许的最大值（BYTE 类型）。

说明

调用 DDV_MinMaxByte 以检验与 *value* 相关的控件中的值是否介于 *minVal* 和 *maxVal* 之间。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxDateTime

Void AFXAPI DDV_MinMaxDateTime (CDataExchange* *pDX*, CTime& *refvalue*, const
Ctime* *refMinRange*, Const Ctime* *refMaxRange*);
Void AFXAPIDDV_MinMaxDateTime (CDataExchange* *pDX*, ColeDateTime& *refValue*,
const ColeDateTime* *refMinRange*, const ColeDateTime* *refMaxRange*);

参数

pDX

指向 CDataExchange **对象的指针，框架提供了这个对象以建立数据交换的环境，包括其方向。你无需删除该对象。**

refValue

对与对话框、表格视图或控件视图对象的成品变量相关的 CTime **或** ColeDateTime **对象的引用。该对象包含了要被检验的数据。**

refMinRange

允许的最小的日期/时间值。

refMaxRange

允许的最大的日期/时间值。

说明

调用 DDV_MaxDateTime 以检验与 refValue 相关的日历控件（CDateTimeCtrl）中的时间/日期值是否介于 refMinRange 和 refMaxRange 之间。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++程序员教程》中的“加入对话框”。

请参阅 DDX_DateTimeCtrl, DDV_MinMaxMonth

DDV_MinMaxDouble

void AFXAPI DDV_MinMaxDouble(CDataExchange* *pDX*, double const& *value*, double *minVal*, double *maxVal*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对对话框、表格视图或控件视图对象的成员变量的引用，将对其进行数据校验。

minVal

允许的最小值（double 类型）。

maxVal

允许的最大值（double 类型）。

说明

调用 DDV_MinMaxDouble 以检验与 *value* 相关的控件中的值是否介于 *minVal* 和 *maxVal* 之间。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxDWord

void AFXAPI DDV_MinMaxDWord(CDataExchange* *pDX*, DWORD const& *value*, DWORD *minVal*, DWORD *maxVal*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对对话框、表格视图或控件视图对象的成员变量的引用，将对其进行数据校验。

minVal

允许的最小值（DWORD 类型）。

maxVal

允许的最大值（DWORD 类型）。

说明

调用 DDV_MinMaxDWord 以检验与 value 相关的控件中的值是否介于 minVal 和 maxVal 之间。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxFloat

void AFXAPI DDV_MinMaxFloat(CDataExchange* *pDX*, float *value*, float *minVal*, float *maxVal*);

参数

pDX

　　指向 CDataExchange **对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。**

value

　　对对话框、表格视图或控件视图对象的成员变量的引用，将对其进行数据校验。

minVal

　　允许的最小值（float **类型）。**

maxVal

　　允许的最大值（float **类型）。**

说明

调用 DDV_MinMaxInt **以检验与** value **相关的控件中的值是否介于** minVal **和** maxVal **之间。**

关于DDV的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxInt

void AFXAPI DDV_MinMaxInt(CDataExchange* *pDX*, int *value*, int *minVal*, int *maxVal*);

参数

pDX
　　指向CDataExchange对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value
　　对对话框、表格视或控件视对象的成员变量的引用，将对其进行数据校验。

minVal
　　允许的最小值（int类型）。

maxVal
　　允许的最大值（int类型）。

说明

调用 DDV_MinMaxInt 以检验与 value 相关的控件中的值是否介于 minVal 和 maxVal 之间。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxLong

void AFXAPI DDV_MinMaxLong(CDataExchange* *pDX*, long *value*, long *minVal*, long *maxVal*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对对话框、表格视或控件视对象的成员变量的引用，将对其进行数据校验。

minVal

允许的最小值（long类型）。

maxVal

允许的最大值（long类型）。

说明

调用DDV_MinMaxLong以检验与value相关的控件中的值是否介于minVal和maxVal之间。

关于DDV的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxMonth

void AFXAPI DDV_MinMaxMonth(CDataExchange* *pDX*,CTime& *refValue*, const CTime* *refMinRange*, const CTime* *refMaxRange*);
void AFXAPI DDV_MinMaxMonth(CDataExchange* *pDX*,COleDateTime& *refValue*, const COleDateTime* *refMinRange*, const COleDateTime* *refMaxRange*);

参数

pDX

指向CDataExchange对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

refValue

对与对话框、表格视或控件视对象的成员变量相关的CTime或COleDateTime对象的引用。该对象包含了要被校验的数据。当DDV_MinMaxMonth被调用时，MFC把这个引用传递给它。

refMinRange

允许的最小的日期/时间值。

refMaxRange

允许的最大的日期/时间值。

说明

调用DDV_MinMaxMonth以检验与refValue相关的日历控件（CMonthCalCtrl）中的时间/日期值是否介于refMinRange和refMaxRange之间。

关于DDV的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDV_MinMaxSlider

void AFXAPI DDV_MinMaxSlider(CDataExchange* *pDX*, DWORD *value*, DWORD *minVal*, DWORD *maxVal*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对要被校验的数据的引用。这个参数保存或设置滑块控件的当前位置。

minVal

允许的最小值。

maxVal

允许的最大值。

说明

调用 DDV_MinMaxSlider 以检验与 value 相关的控件中的值是否介于 minVal 和 maxVal 之间。

关于DDV的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关滑块控件的更多信息参见《Visual C++程序员指南》中的“使用CSliderCtrl”。

请参阅 DDX_Slider, DDX_FieldSlider

DDV_MinMaxUnsigned

void AFXAPI DDV_MinMaxUnsigned(CDataExchange* *pDX*, unsigned *value*, unsigned *minVal*, unsigned *maxVal*);

参数

pDX

指向CDataExchange对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

value

对对话框、表格视或控件视对象的成员变量的引用，将对其进行数据校验。

minVal

允许的最小值（unsigned类型）。

maxVal

允许的最大值（unsigned 类型）。

说明

调用 DDV_MinMaxUnsigned 以检验与 value 相关的控件中的值是否介于 minVal 和 maxVal 之间。

关于 DDV 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDX_CBIndex

void AFXAPI DDX_CBIndex(CDataExchange* *pDX*, int *nIDC*, int& *index*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

与控件属性相关的组合框控件的ID。

index

对对话框、表格视或控件视的成员变量的引用，将与该变量发生数据交换。

说明

DDX_CBIndex 函数管理着在对话框、表格视或控件视中的组合框控件与对话框、表格视或控件视的整数数据成员之间的整数数据交换。

当 DDX_CBIndex 被调用时，index 被设置为组合框的当前选择的索引。如果没有选择任何项，则 index 被设为 0。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

请参阅 DDP_CBIndex

DDX_CBString

void AFXAPI DDX_CBString(CDataExchange* *pDX*, int *nIDC*, CString& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

与控件属性相关的组合框控件的资源 ID。

value

对对话框、表格视或控件视的成员变量的引用，将与该变量发生数据交换。

说明

DDX_CBString 函数管理着在对话框、表格视或控件视中的组合框中的编辑控件与对话框、表格视或控件视的 CString 数据成员之间的 CString 数据交换。

当 DDX_CBString 被调用时，value 被设置为组合框的当前选择。如果没有选择任何项，则 index 被设为空字符串。

注意 如果组合框是一个下拉列表框，交换的值被限制在 255 个字符以内。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和

校验”和《Visual C++教程》中的“加入对话框”。

请参阅 DDP_CBString

DDX_CBStringExact

void AFXAPI DDX_CBStringExact(CDataExchange* *pDX*, int *nIDC*, CString& *value*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。**

nIDC

与控件属性相关的组合框控件的资源 ID**。**

value

对对话框、表格视或控件视的成员变量的引用，将与该变量发生数据交换。

说明

DDX_CBStringExact 函数管理着在对话框、表格视或控件视中的组合框中的编辑控件与对话框、表格视或控件视的 CString 数据成员之间的 CString 数据交换。

当 DDX_CBString 被调用时，value 被设置为组合框的当前选择。如果没有选择任何项，则 index 被设为空字符串。

> **注意** 如果组合框是一个下拉列表框，交换的值被限制在 255 个字符以内。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

请参阅 DDP_CBStringExact

DDX_Check

void AFXAPI DDX_Check(CDataExchange* *pDX*, int *nIDC*, int& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换

的环境，包括其方向。

nIDC

与控件属性相关的复选框控件的资源 ID。

value

对对话框、表格视或控件视的成员变量的引用，将与该变量发生数据交换。

说明

DDX_Check 函数管理着在对话框、表格视或控件视中的复选框控件与对话框、表格视或控件视的整数数据成员之间的整数数据交换。

当 DDX_Check 被调用时，value 被设置为复选框的当前状态。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

请参阅 DDP_Check

DDX_Control

void AFXAPI DDX_Control(CDataExchange* *pDX*, int *nIDC*, CWnd& *rControl*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

与控件属性相关的被子类化的控件的资源 ID。

rControl

对对话框、表格视或控件视的成员变量的引用，将与该变量发生数据交换。

说明

DDX_Control 函数管理着在对话框、表格视或控件视中被子类化的控件与对话框、表格视或控件视的 CWnd 数据成员之间的数据交换。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

DDX_DateTimeCtrl

void AFXAPI DDX_DateTimeCtrl(CDataExchange* *pDX*, int *nIDC*, CTime&

value);
void AFXAPI DDX_DateTimeCtrl(CDataExchange* *pDX*, int *nIDC*, COleDateTime& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。你不需要删除这个对象。

nIDC

与成员变量相关的日期/时间选取控件的资源 ID。

value

对对话框、表格视或控件视的 CTime 或 COleDateTime 成员变量的引用，将与该变量发生数据交换。

说明

DDX_DateTimeCtrl **函数管理着在对话框、表格视或控件视中的日期/时间选取控件（CDateTimeCtrl）与对话框、表格视或控件视的 CTime 或 COleDateTime 数据成员之间的日期和（或）时间数据交换。**
当 DDX_DateTimeCtrl 被调用时，value 被设置为日期/时间选取控件的当前状态，或者控件的状态被设为 value，这取决于数据交换的方向。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

请参阅 CDateTimeCtrl, CDateTimeCtrl::SetRange, CDateTimeCtrl::GetRange, DDV_MinMaxDateTime

DDX_FieldCBIndex

void AFXAPI DDX_FieldCBIndex(CDataExchange* *pDX*, int *nIDC*, int& *index*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldCBIndex(CDataExchange* *pDX*, int *nIDC*, int& *index*, CdaoRecord-set* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

CRecordView 或 CDaoRecordView 对象中控件的 ID。

index

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldCBIndex 函数使记录视中组合框控件中的列表框的当前选择项与记录视的相关记录集对象的整数数据成员同步。当数据是从记录集传送到控件时，这个函数将根据 index 指定的值设置控件的当前选择项。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，MFC 将 index 的值设为 0。对于从控件到记录集的数据传送，如果控件为空或没有选择任何项，则记录集的字段被设为 0。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。DDX_FieldCBIndex 的例子与此类似。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCBStringExact, DDX_FieldLBIndex, DDX_FieldScroll, DDX_CBIndex

DDX_FieldCBString

void AFXAPI DDX_FieldCBString(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldCBString(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。**

nIDC

CRecordView **或** CDaoRecordView **对象中控件的** ID**。**

index

对相关的 CRecordset **或** CDaoRecordset **对象的字段数据成员的引用。**

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldCBString 函数管理着记录视中组合框控件的编辑框与记录视的相关记录集的 CString 数据成员之间的 CString 数据交换。当数据是从记录集传送到控件时，这个函数将组合框控件的当前选择设为以 value 指定的字符串中字符开始的第一项。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，组合框的选择将被清除，其编辑框被设为空。对于从控件到记录集的数据传送，如果控件为空，则在允许的情况下，记录集的字段被设为 NULL。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。这个例子中包含了对 DDX_FieldCB- String 的调用。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCBStringExact

DDX_FieldCBStringExact

void AFXAPI DDX_FieldCBStringExact(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldCBStringExact(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

CRecordView 或 CDaoRecordView 对象中控件的 ID。

index

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldCBStringExact 函数管理着记录视中组合框控件的编辑框与记录视的相关记录集的 CString 数据成员之间的 CString 数据交换。当数据是从记录集传送到控件时，这个函数将组合框控件的当前选择设为第一个与 value 指定的字符串严格匹配的项。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，组合框的选择将被清除，其编辑框被设为空。对于从控件到记录集的数据传送，如果控件为空，记录集的字段被设为 NULL。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。对 DDX_FieldCBStringExact 的调用是类似的。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCBString

DDX_FieldCheck

void AFXAPI DDX_FieldCheck(CDataExchange* *pDX*, int *nIDC*, int& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldCheck(CDataExchange* *pDX*, int *nIDC*, int& *value*, CDaoRecordset* *pRecordset*);

参数

pDX
　　指向 CDataExchange **对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。**

nIDC
　　CRecordView **或** CDaoRecordView **对象中控件的** ID**。**

value
　　对相关的 CRecordset **或** CDaoRecordset **对象的字段数据成员的引用。**

pRecordset
　　指向 CRecordset **或** CDaoRecordset **对象的指针，将与其发生数据交换。**

说明

DDX_FieldCheck 函数管理着对话框、表格视或控件视中复选框控件与对话框、表格视或控件视的 int 数据成员之间的整数数据交换。

当 DDX_FieldCheck 被调用时，value 被设置为复选框控件的当前状态，也可能控件的状态被设为 value，这取决于数据传送的方向。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCBString

DDX_FieldLBIndex

void AFXAPI DDX_FieldLBIndex(CDataExchange* *pDX*, int *nIDC*, int& *index*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldLBIndex(CDataExchange* *pDX*, int *nIDC*, int& *index*, CdaoRecord-set* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

CRecordView 或 CDaoRecordView 对象中控件的 ID。

index

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldLBIndex 函数使记录视中列表控件的当前选择与记录视的相关记录集的 int 数据成员保持同步。当数据是从记录集传送到控件时，这个函数将根据 index 的值设置控件的当前选择项。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，MFC 将把 index 的值设为 0。对于从控件到记录集的数据传送，如果控件为空，记录集的字段被设为 0。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和

CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCBStringExact, DDX_FieldCBIndex, DDX_FieldScroll, DDX_LBIndex

DDX_FieldLBString

void AFXAPI DDX_FieldLBString(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldLBString(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换

的环境，包括其方向。

nIDC

CRecordView 或 CDaoRecordView 对象中控件的 ID。

value

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldLBString 函数将记录视中列表框的当前选择复制给记录视的相关记录集的 CString 数据成员。在相反的方向，这个函数将列表框控件的当前选择设为第一个以 value 指定的字符串中字符为开头的项。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，列表框的选择将被清除。对于从控件到记录集的数据传送，如果控件为空，记录集的字段被设为 NULL。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和

CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。对 DDX_FieldLBString 的调用是类似的。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBStringExact, DDX_FieldCBString, DDX_FieldCBStringExact, DDX_FieldCBIndex, DDX_FieldLBIndex, DDX_FieldScroll

DDX_FieldLBStringExact

void AFXAPI DDX_FieldLBStringExact(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldLBStringExact(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

CRecordView 或 CDaoRecordView 对象中控件的 ID。

value

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldLBStringExact 函数将记录视中列表框的当前选择复制给记录视的相关记录集的 CString 数据成员。在相反的方向，这个函数将列表框控件的当前选择设为第一个与 value 指定的字符严格匹配的项。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，列表框的选择将被清除。对于从控件到记录集的数据传送，如果控件为空，记录集的字段被设为 NULL。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和

CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。对 DDX_FieldLBStringExact 的调用是类似的。

请参阅 DDX_FieldText, DDX_FieldRadio, DDX_FieldLBString, DDX_FieldCBString, DDX-_FieldCBStringExact, DDX_FieldCBIndex, DDX_FieldLBIndex, DDX_FieldScroll

DDX_FieldSlider

void AFXAPI DDX_FieldSlider(CDataExchange* *pDX*, int *nIDC*, int& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldSlider(CDataExchange* *pDX*, int *nIDC*, int& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

滑块控件的资源 ID。

value

对要交换的值的引用。这个参数保存滑块控件的当前位置，或者被用于设置滑块控件的当前位置。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldSlider 函数使记录视中滑块控件的位置与记录视的相关记录集中的 int 数据成员（或者是你选择的任何整数变量）保持同步。当数据是从记录集传送到滑块控件时，这个函数将滑块控件的位置设为 value 指定的值。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，滑块控件的位置被设为 0。对于从控件到记录集的数据传送，如果控件为空，记录集的字段被设为 0。

DDX_FieldSlider 并不与滑块控件交换范围信息，否则就可以设置范围而不仅是位置。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO

的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的“记录视”。关于滑块控件的信息参见《Visual C++程序员指南》中的“使用 CSliderCtrl”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。对 DDX_FieldSlider 的调用是类似的。

请参阅 DDX_Slider, DDV_MinMaxSlider, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCBString, DDX_FieldCBStringExact, DDX_FieldCBIndex, DDX_Field-LBIndex

DDX_FieldRadio

void AFXAPI DDX_FieldRadio(CDataExchange* *pDX*, int *nIDC*, int& *value*, CRecordset* *pRecordset*);
void AFXAPI DDX_FieldRadio(CDataExchange* *pDX*, int *nIDC*, int& *value*,

CDaoRecordset* *pRecordset*);

参数

pDX

　　指向 CDataExchange **对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。**

nIDC

　　在 CRecordView **或** CDaoRecordView **对象中一组相邻的单选按钮（具有** WS_GROUP **风格）中的第一个的** ID**。**

value

　　对相关的 CRecordset **或** CDaoRecordset **对象的字段数据成员的引用。**

pRecordset

　　指向 CRecordset **或** CDaoRecordset **对象的指针，将与其发生数据交换。**

说明

DDX_FieldRadio **函数把记录视的记录集中从零开始的** int **型成员变量与记录视中一组单选按钮的当前选择项关联起来。当数据是从记录集字段传送到记录视时，这个函数打开第** n **个（从零开始）单选按钮并关闭其它的单选按钮。按相反的方向，这个函数将记录集的字段设为当前打开（被选中）的单选按钮的序**

号。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，没有一个按钮被选中。对于从控件到记录集的数据传送，如果没有控件被选中，如果允许，则记录集的字段被设为 NULL。

如果你正在使用基于 ODBC 的类,则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。对 DDX_FieldRadio 的调用是类似的。

请参阅 DDX_FieldText, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCB-String, DDX_FieldCBStringExact, DDX_FieldCBIndex, DDX_FieldLBIndex, DDX_FieldScroll

DDX_FieldScroll

void AFXAPI DDX_FieldScroll(CDataExchange* *pDX*, int *nIDC*, int& *value*,

CRecordset* *pRecordset*);
void AFXAPI DDX_FieldScroll(CDataExchange* *pDX*, int *nIDC*, int& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象以建立数据交换的环境，包括其方向。

nIDC

在 CRecordView 或 CDaoRecordView 对象中一组相邻的单选按钮（具有 WS_GROUP 风格）中的第一个的 ID。

value

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。

说明

DDX_FieldScroll 函数使记录视中滚动条控件的位置与记录视的相关记录集中的 int 数据成员（或者是你选择的任何整数变量）保持同步。当数据是从记录集传

送到滑块控件时，这个函数将滚动条控件的位置设为 value 指定的值。对于从记录集到控件的数据传送，如果记录集的字段为 NULL，滚动条控件的位置被设为 0。对于从控件到记录集的数据传送，如果控件为空，记录集的字段被设为 0。

如果你正在使用基于 ODBC 的类，则使用第一个版本。如果你正在使用基于 DAO 的类，则使用第二个版本。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的“记录视”。

示例

参看 DDX_FieldText 以获得一般的 DDX_Field 例子。对 DDX_FieldScroll 的调用是类似的。

请参阅 DDX_FieldText, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_Field-CBString, DDX_FieldCBStringExact, DDX_FieldCBIndex, DDX_FieldLBIndex, DDX_Scroll

DDX_FieldText

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, BYTE& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, int& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, UINT& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, long& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, DWORD& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, float& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, double& *value*, CRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, short& *value*, CDaoRecordset* *pRecordset*);

void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, BOOL& *value*,

CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, BYTE& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, long& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, DWORD& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, CString& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, float& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, double& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, COleDateTime& *value*, CDaoRecordset* *pRecordset*);
void AFXAPI DDX_FieldText(CDataExchange* *pDX*, int *nIDC*, COleCurrency& *value*, CDaoRecordset* *pRecordset*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象以建立数据交换**

的环境，包括其方向。

nIDC

在 CRecordView 或 CDaoRecordView 对象中控件的 ID。

value

对相关的 CRecordset 或 CDaoRecordset 对象的字段数据成员的引用。value 的数据类型与你使用的 DDX_FieldText 的版本有关。

pRecordset

指向 CRecordset 或 CDaoRecordset 对象的指针，将与其发生数据交换。这个指针使 DDX_FieldText 能够检测并设置 NULL 值。

说明

DDX_FieldText 函数管理着编辑框控件与记录集的字段数据成员之间的 int，short，long，DWORD，CString，float，double，BOOL 或 BYTE 类型的数据交换。对于 CDaoRecordset 对象，DDX_FieldText 函数还管理着 COleDateTime 和 COleCurrency 数值的传送。一个空的编辑控件代表了一个 NULL 值。对于从记录集到控件的数据传送，如果记录集的与为 NULL，则编辑框被设为空。对于从控件到记录集的数据传送，如果控件为空，则记录集的字段被设为 NULL。

如果你正在使用基于 ODBC 的类，则使用带 CRecordset 参数的版本。如果你

正在使用基于 DAO 的类，则使用带 CDaoRecordset 参数的类。

有关 DDX 的更多信息参见《Visual C++程序员指南》中的“对话框数据交换和校验”和《Visual C++教程》中的“加入对话框”。有关 CRecordView 和 CDaoRecordView 中的 DDX 的更多信息以及例子参见《Visual C++指南》中的文章“记录视”。

示例

下面针对 CRecordView 的包含了 DDX_FieldText 函数 DoDataExchange 是为三种类型调用的：IDC_COURSELIST 是一个组合框，其它两种控件是编辑框。对于 DAO 编程，m_pSet 参数是一个指向 CRecordset 或 CDaoRecordset 的指针。

```
//DDX_FieldText 的例子
void CSectionForm::DoDataExchange( CDataExchange* pDX )
{
	CRecordView::DoDataExchange( pDX );
	//{{AFX_DATA_MAP(CSectionForm)
	DDX_FieldCBString( pDX, IDC_COURSELIST,
	                        m_pSet->m_strCourseID, m_pSet);
	DDX_FieldText( pDX, IDC_ROOM, m_pSet->m_nRoomNo,
	                        m_pSet );
	DDX_FieldText( pDX, IDC_TUITION,
```

```
                  m_pSet->m_dwTuition, m_pSet );
    //}}AFX_DATA_MAP
}
```

请参阅 DDX_FieldRadio, DDX_FieldLBString, DDX_FieldLBStringExact, DDX_FieldCB-String, DDX_FieldCBStringExact, DDX_FieldCBIndex, DDX_FieldLBIndex, DDX_-FieldScroll

DDX_LBIndex

void AFXAPI DDX_LBIndex(CDataExchange* *pDX*, int *nIDC*, int& *index*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。**

nIDC

与控件属性相关的列表框控件的资源 ID**。**

index

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_LBIndex 函数管理着对话框、表格视或控件视对象中的列表框控件与对话框、表格视或控件视对象的 int 型数据成员之间的 int 型数据交换。

当 DDX_LBIndex 被调用的时候，index 被设为列表框的当前选择项的索引。如果没有选择任何项，则 index 被设为 0。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDP_LBIndex

DDX_LBString

void AFXAPI DDX_LBString(CDataExchange* *pDX*, int *nIDC*, CString& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

与控件属性相关的列表框控件的资源 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_LBString 函数管理着对话框、表格视或控件视对象中的列表框控件中编辑框与对话框、表格视或控件视对象的 CString 型数据成员之间的 CString 型数据交换。

当 DDX_LBString 被调用的时候，value 被设为列表框的当前选择项。如果没有选择任何项，则 index 被设为空字符串。

注意 如果该列表框是一个下拉列表框，交换的值被限制在 255 个字符以内。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDP_LBString

DDX_LBStringExact

void AFXAPI DDX_LBStringExact(CDataExchange* *pDX*, int *nIDC*, CString& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

与控件属性相关的列表框控件的资源 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_LBStringExact 函数管理着对话框、表格视或控件视对象中的列表框控件中编辑框与对话框、表格视或控件视对象的 CString 型数据成员之间的 CString 型数据交换。

当 DDX_LBStringExact 被调用的时候，value 被设为列表框的当前选择项。如果没有选择任何项，则 index 被设为空字符串。

> **注意**　如果该列表框是一个下拉列表框，交换的值被限制在 255 个字符以内。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

DDX_MonthCalCtrl

void AFXAPI DDX_MonthCalCtrl(CDataExchange* *pDX*, int *nIDC*, CTime& *value*);
void AFXAPI DDX_MonthCalCtrl(CDataExchange* *pDX*, int *nIDC*, COleDateTime& *value*);

参数

pDX
　　指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。你不需要自己删除这个对象。

nIDC
　　与成员变量相关的月历控件的资源 ID。

value

　　对对话框、表格视或控件视对象的 CTime 或 COleDateTime 成员变量的引用，将与其发生数据交换。

说明

DDX_MonthCalCtrl 函数管理着对话框、表格视或控件视对象中的月历控件与对话框、表格视或控件视对象的 CTime 或 COleDateTime 型数据成员之间的数据交换。

　　注意　该控件仅管理日期值。时间对象中的时间域被设为控件窗口的创建时间，或者是通过 CMonthCalCtrl::SetCurSel 设置的任意时间。

当 DDX_MonthCalCtrl 被调用的时候，value 被设为月历控件的当前状态。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅　DDX_DateTimeCtrl, CMonthCalCtrl, CMonthCalCtrl::GetCurSel, CmonthCalCtrl:: SetCurSel

DDX_OCBool

void AFXAPI DDX_OCBool(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*,

```
BOOL& value );
```

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCBool 函数管理着对话框、表格视或控件视对象中的 OLE 控件属性与对话框、表格视或控件视对象的 BOOL 型数据成员之间的 BOOL 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCBoolRO

DDX_OCBoolRO

void AFXAPI DDX_OCBoolRO(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, BOOL& *value*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。**

nIDC

对话框、表格视或控件视中的 OLE **控件的** ID**。**

dispid

控件属性的调度 ID**。**

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCBoolRO 函数管理着对话框、表格视或控件视对象中的 OLE 控件的只读属性与对话框、表格视或控件视对象的 BOOL 型数据成员之间的 BOOL 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCBool

DDX_OCColor

void AFXAPI DDX_OCColor(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, OLE_COLOR & *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCColor 函数管理着对话框、表格视或控件视对象中的 OLE 控件的属性与对话框、表格视或控件视对象的 OLE_COLOR 型数据成员之间的 OLE_COLOR 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCColorRO

DDX_OCColorRO

void AFXAPI DDX_OCColorRO(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, OLE_COLOR & *value*);

参数

pDX

指向CDataExchange对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的OLE控件的ID。

dispid

控件属性的调度ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCColorRO函数管理着对话框、表格视或控件视对象中的OLE控件的只读属性与对话框、表格视或控件视对象的OLE_COLOR型数据成员之间的OLE_COLOR型数据交换。

关于DDX的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCColor

DDX_OCFloat

void AFXAPI DDX_OCFloat(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, float& *value*);
void AFXAPI DDX_OCFloat(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, double& *value*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。**

nIDC

对话框、表格视或控件视中的 OLE **控件的** ID**。**

dispid

控件属性的调度 ID**。**

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交

换。

说明

DDX_OCFloat 函数管理着对话框、表格视或控件视对象中的 OLE 控件的属性与对话框、表格视或控件视对象的 float（或 double）型数据成员之间的 float（或 double）型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCFloatRO

DDX_OCFloatRO

void AFXAPI DDX_OCFloatRO(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, float& *value*);
void AFXAPI DDX_OCFloatRO(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, double& *value*);

参数

pDX

指向CDataExchange对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的OLE控件的ID。

dispid

控件属性的调度ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCFloatRO函数管理着对话框、表格视或控件视对象中的OLE控件的只读属性与对话框、表格视或控件视对象的float（或double）型数据成员之间的float（或double）型数据交换。

关于DDX的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅　DDX_OCFloat

DDX_OCInt

void AFXAPI DDX_OCInt(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, int& *value*);
void AFXAPI DDX_OCInt(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, long& *value*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。**

nIDC

对话框、表格视或控件视中的 OLE **控件的** ID**。**

dispid

控件属性的调度 ID**。**

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCInt 函数管理着对话框、表格视或控件视对象中的 OLE 控件的属性与对话框、表格视或控件视对象的 int（或 long）型数据成员之间的 int（或 long）型数据交换。
关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCIntRO

DDX_OCIntRO

void AFXAPI DDX_OCIntRO(CDataExchange* *pDX*, int nIDC, DISPID *dispid*, int& *value*);
void AFXAPI DDX_OCLongRO(CDataExchange* *pDX*, int nIDC, DISPID *dispid*, long& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCIntRO 函数管理着对话框、表格视或控件视对象中的 OLE 控件的只读属性与对话框、表格视或控件视对象的 int（或 long）型数据成员之间的 int（或 long）型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCInt

DDX_OCShort

void AFXAPI DDX_OCShort(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*,

short& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCShort 函数管理着对话框、表格视或控件视对象中的 OLE 控件的属性与对话框、表格视或控件视对象的 short 型数据成员之间的 short 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++

程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCShortRO

DDX_OCShortRO

void AFXAPI DDX_OCShortRO(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, short& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCShortRO 函数管理着对话框、表格视或控件视对象中的 OLE 控件的只读属性与对话框、表格视或控件视对象的 short 型数据成员之间的 short 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCShort

DDX_OCText

void AFXAPI DDX_OCText(CDataExchange* *pDX*, int *nIDC*, DISPID *dispid*, CString& *value*);

参数

pDX

　　指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

　　对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCText 函数管理着对话框、表格视或控件视对象中的 OLE 控件的属性与对话框、表格视或控件视对象的 CString 型数据成员之间的 CString 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCTextRO

DDX_OCTextRO

```
void AFXAPI DDX_OCTextRO( CDataExchange* pDX, int nIDC, DISPID dispid, CString& value );
```

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

对话框、表格视或控件视中的 OLE 控件的 ID。

dispid

控件属性的调度 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_OCTextRO 函数管理着对话框、表格视或控件视对象中的 OLE 控件的只读属性与对话框、表格视或控件视对象的 CString 型数据成员之间的 CString 型数据交换。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

请参阅 DDX_OCText

DDX_Radio

void AFXAPI DDX_Radio(CDataExchange* *pDX*, int *nIDC*, int& *value*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。**

nIDC

与控件属性相关的单选控件的资源 ID**。**

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_Radio **函数管理着对话框、表格视或控件视对象中的单选控件组与对话框、表格视或控件视对象的** int **型数据成员之间的** int **型数据交换。**

当 DDX_Radio 被调用时，将根据 value 设置单选控件组的当前状态。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

DDX_Scroll

void AFXAPI DDX_Scroll(CDataExchange* *pDX*, int *nIDC*, int& *value*);

参数

pDX

指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

与控件属性相关的滚动条控件的资源 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_Scroll 函数管理着对话框、表格视或控件视对象中的滚动条控件与对话框、表格视或控件视对象的 int 型数据成员之间的 int 型数据交换。

当 DDX_Scroll 被调用时，将根据 value 的值设置滚动条控件的当前位置。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。

DDX_Slider

void AFXAPI DDX_Slider(CDataExchange* *pDX*, int *nIDC*, int& *value*);

参数

pDX

　　指向 CDataExchange 对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。

nIDC

　　与控件属性相关的滑块控件的资源 ID。

value

对对话框、表格视或控件视对象的成员变量的引用，将与其发生数据交换。

说明

DDX_Slider 函数管理着对话框、表格视或控件视对象中的滑块控件与对话框、表格视或控件视对象的 int 型数据成员之间的 int 型数据交换。

当 DDX_Slider 被调用时，将根据 value 的值设置滑块控件的当前位置，也有可能是将 value 设为控件的当前位置，这取决于交换的方向。

关于 DDX 的更多信息参见《Visual C++教程》中的“加入对话框”和《Visual C++程序员指南》中的“对话框数据交换与校验”。有关滑块控件的更多信息参见《Visual C++程序员指南》中的“使用 CSliderCtrl”。

请参阅 DDX_FieldSlider, DDV_MinMaxSlider

DDX_Text

```
void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, BYTE& value );
void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, short& value );
void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, int& value );
void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, UINT& value );
void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, long& value );
```

void AFXAPI DDX_Text(CDataExchange* *pDX*, int *nIDC*, DWORD& *value*);
void AFXAPI DDX_Text(CDataExchange* *pDX*, int *nIDC*, CString& *value*);
void AFXAPI DDX_Text(CDataExchange* *pDX*, int *nIDC*, float& *value*);
void AFXAPI DDX_Text(CDataExchange* *pDX*, int *nIDC*, double& *value*);
void AFXAPI DDX_Text(CDataExchange* *pDX*, int *nIDC*, COleCurrency& *value*);
void AFXAPI DDX_Text(CDataExchange* *pDX*, int *nIDC*, COleDateTime& *value*);

参数

pDX

指向 CDataExchange **对象的指针。框架提供了这个对象，用于建立数据交换的环境，包括其方向。**

nIDC

对话框、表格视或控件视中编辑控件的 ID**。**

value

对对话框、表格视或控件视对象的成员变量的引用，其类型取决于你使用了 DDX_Text **的哪一个重载版本。**

说明

DDX_Text 函数管理着对话框、表格视或控件视对象中的编辑控件与对话框、表格视或控件视对象的 CString 型数据成员之间的 int，UINT，long，DWORD，CString，float 或 double 型数据交换。

关于 DDX 的更多信息参见 Visual C++联机教程中的“加入对话框”和 Visual C++程序员联机指南中的“对话框数据交换与校验”。

DEBUG_NEW

#define new DEBUG_NEW

说明

用于帮助发现内存泄漏。你可以在程序中原来使用 new 操作符来分配内存的任何地方使用 DEBUG_NEW。

在调试模式（当定义了_DEBUG 符号时）中，DEBUG_NEW 记录它所分配的每个对象的文件名和行号。然后，当你使用 CMemoryState::DumpAllObjectsSince 成员函数时，每个用 DEBUG_NEW 分配的对象将与发生分配的文件名和行号一起现实。

为了使用 DEBUG_NEW，在你的源文件中插入下面的指令：

```
#define new DEBUG_NEW
```

一旦你插入了这个指令，预处理程序就会在你使用 new 的地方插入 DEBUG_NEW，剩下的事情由 MFC 完成。当你编译你的程序的发行版本时，DEBUG_NEW 转变为简单的 new 操作，也不再生成文件名和行号信息。

> **注意** 在 MFC 的以前版本（4.1 或更早）中，你必须在调用 IMPLEMENT_DYNCREATE 或 IMPLEMENT_SERIAL 宏的语句后面加入#define 语句，现在不再需要这样做了。

关于 DEBUG_NEW 宏的更多信息参见"Visual C++程序员指南"中的"MFC 调试支持"。

DECLARE_CONNECTION_MAP

DECLARE_CONNECTION_MAP( )

说明

在你的程序中每一个 COleControl 的派生类都可以提供一个连接映射，以指定你的控制支持的附加的连接点。

如果你的控制支持附加的连接点，就在你的类声明的末尾使用

DECLARE_CONNECTI- ON_MAP宏。然后，在实现了类成员函数的.CPP文件中，加入BEGIN_CONNECTION_MAP宏，为控制的每个连接点加上CONNECTION_PART宏，最后用END_CONNECTION_MAP宏来声明连接映射的结束。

请参阅 BEGIN_CONNECTION_PART, BEGIN_CONNECTION_MAP, CONNECTION_IID

DECLARE_DISPATCH_MAP

DECLARE_DISPATCH_MAP( )

说明

如果你的程序中的CCmdTarget的派生类支持OLE自动化，这个类必须提供一个调度映射以公开它的方法和属性。在你的类声明的末尾使用DECLARE_DISPATCH_MAP宏。然后，在实现了类成员函数的.CPP文件中加入BEGIN_DISPATCH_MAP宏。然后为你的类的每个公开的方法和属性加入宏入口（DISP_FUNCTION，DISP_PROPERTY等）。最后，使用END_DISPATCH_MAP宏。

注意 如果你在DECLARE_DISPATCH_MAP之后定义了成员，你必须为它们

指定一个新的访问类型（public，private或protected）。

AppWizard 和 ClassWizard 帮助你创建自动化类并维护调度映射。参见关于“AppWizard”以及“ClassWizard：自动化支持”的文章。有关调度映射的更多信息参见“自动化服务器”。所有的文章都在“Visual C++程序员指南”中。

示例

```
// DECLARE_DISPATCH_MAP 的例子
class CMyDoc : public CDocument
{
	// 成员声明
	DECLARE_DISPATCH_MAP()
};
```

请参阅 Dispatch Maps, BEGIN_DISPATCH_MAP, END_DISPATCH_MAP, DISP_FUNCTION, DISP_PROPERTY, DISP_PROPERTY_EX, DISP_DEFVALUE

DECLARE_DYNAMIC

DECLARE_DYNAMIC(*class_name*)

参数

class_name

类的实际名字（不用引号括起来）。

说明

当你从 CObject 继承一个类的时候，这个宏加入了访问类的运行时信息的能力。

在类的头文件（.H）中加入 DECLARE_DYNAMIC 宏，然后在所有需要访问这个类的对象的.CPP 模块中包含这个头文件。

如果你按照前面描述的方式使用了 DECLARE_DYNAMIC 和 IMPLEMENT_DYNAMIC 宏，你就可以在运行时利用 RUNTIME_CLASS 宏和 CObject::IsKindOf 函数来确定对象所属的类。

如果在类声明中包含了 DECLARE_DYNAMIC，那么必须在类的实现中包含 IMPLEMENT_DYNAMIC 宏。

有关 DECLARE_DYNAMIC 宏的更多信息参见“Visual C++程序员指南”中的

“COBject类主题”。

请参阅 IMPLEMENT_DYNAMIC, DECLARE_DYNCREATE, DECLARE_SERIAL, RUNTIME_CLASS, CObject::IsKindOf

DECLARE_DYNCREATE

DECLARE_DYNCREATE(*class_name*)

参数

class_name

类的实际名字（不用引号括起来）。

说明

使用 DECLARE_DYNCREATE 宏可以使每个 CObject 的派生类的对象具有运行时动态创建的能力。框架利用这种能力来动态创建对象，例如，当它在串行化过程中从磁盘读取对象的时候。文档、视和框架类必须支持动态创建，因为框架需要动态地创建它们。

在类的.H 模块中加入 DECLARE_DYNCREATE 宏，然后在每个需要访问这个类的对象的.CPP 模块中包含这个模块。

如果在类声明中包含了 DECLARE_DYNCREATE，那么必须在类的实现中包含 IMPLEMENT_DYNCREATE 宏。

关于 DECLARE_DYNCREATE 宏的更多信息参见“Visual C++程序员指南”中的“CObject 类主题”。

请参阅 DECLARE_DYNAMIC, IMPLEMENT_DYNAMIC, IMPLEMENT_DYNCREATE, RUNTIME_CLASS, CObject::IsKindOf

DECLARE_EVENT_MAP

DECLARE_EVENT_MAP( )

说明

你的程序中的每个 COleControl 的派生类都能够提供一个事件映射，以指定你的控制可能引发的事件。在你的类声明的末尾使用 DECLARE_EVENT_MAP 宏。然后，在实现类成员函数的.CPP 文件中使用 BEGIN_ENENT_MAP 宏，再加入每个控制事件的宏入口，最后用 END_EVENT_MAP 宏来声明事件列表的结束。

关于事件映射的更多信息参见“Visual C++程序员指南”中的文章“ActiveX 控制：事件”。

请参阅 BEGIN_EVENT_MAP, END_EVENT_MAP, EVENT_CUSTOM, EVENT_CUSTOM_ID

DECLARE_EVENTSINK_MAP

DECLARE_EVENTSINK_MAP( )

说明

OLE 容器能够提供事件接收映射以指定你的容器能够接收的事件。在你的类声明的末尾使用 DECLARE_EVENTSINK_MAP 宏。然后在实现了类成员函数的.CPP 文件中使用 BEGIN_EVENTSINK_MAP 宏，再加入可以接收的事件的宏入口，最后用 END_EVENTSINK_MAP 宏来声明事件接收列表的结束。

关于事件接收映射的更多信息参见“Visual C++ 程序员指南”中的文章“ActiveX 控制容器”。

请参阅 BEGIN_EVENTSINK_MAP, END_EVENTSINK_MAP, ON_EVENT, ON_PROPNOTIFY

DECLARE_MESSAGE_MAP

DECLARE_MESSAGE_MAP( )

说明

你的程序中的每一个 CCmdTarget 的派生类都可以提供一个消息映射以处理消息。在你的类声明的末尾使用 DECLARE_MESSAGE_MAP 宏。然后，在实现了类成员函数的.CPP 文件中加入 BEGIN_MESSAGE_MAP 宏，再加入每个消息处理函数的宏入口，最后使用 END_MESSAGE_MAP 宏。

> **注意**　如果你在 DECLARE_MESSAGE_MAP 之后定义了成员，那么你必须为它们指定新的访问类型（public, private 或 protected）。

关于消息映射和 DECLARE_MESSAGE_MAP 宏的更多信息参见“Visual C++程序员指南”中的“消息处理”和“映射主题”。

示例

```
// DECLARE_MESSAGE_MAP 的例子
class CMyWnd : public CFrameWnd
{
    // 成员声明
    DECLARE_MESSAGE_MAP( )
};
```

请参阅　BEGIN_MESSAGE_MAP, END_MESSAGE_MAP

DECLARE_OLECREATE

DECLARE_OLECREATE(*class_name*)

#include <afxdisp.h>

参数

class_name

类的实际名字（不用引号括起来）。

说明

DECLARE_OLECREATE宏使得CCmdTarget的派生类的对象能够在OLE自动化的过程中被创建。这个宏使具有OLE能力的应用程序可以创建这种类型的对象。

在类的.H模块中加入DECLARE_OLECREATE宏，然后在每个需要访问这个类对象的.CPP模块中包含这个模块。

如果在类的声明中包含了DECLARE_OLECREATE宏，那么必须在类实现中加入IMPLEMENT_OLECREATE宏。使用了DECLARE_OLECREATE的类声明也必须使用DECLARE_DYNCREATE或DECLARE_SERIAL。

请参阅 IMPLEMENT_OLECREATE, DECLARE_DYNCREATE,

DECLARE_SERIAL

DECLARE_OLECREATE_EX

DECLARE_OLECREATE_EX(*class_name*)

参数

class_name
控制类的名字。

说明

定义了控制类的类工厂和 GetClassID **成员函数。在不支持许可的控制类的头文件中使用这个宏。**

注意这个宏与下面例子代码的作用相同：

```
BEGIN_OLEFACTORY(CSampleCtrl)
END_OLEFACTORY(CSampleCtrl)
```

请参阅 BEGIN_OLEFACTORY, END_OLEFACTORY

DECLARE_OLETYPELIB

DECLARE_OLETYPELIB(*class_name*)

参数

class_name

与类型库相关的控制类的名字。

说明

定义了控制类的 GetTypeLib 成员函数。在控制类的头文件中使用这个宏。

请参阅 IMPLEMENT_OLETYPELIB

DECLARE_PROPPAGEIDS

DECLARE_PROPPAGEIDS(*class_name*)

参数

class_name

拥有属性页的控制类的名字。

说明

OLE控制能够提供一个属性页列表以显示它的属性。在你的类声明的末尾使用DECLARE_PROPPAGEIDS宏。然后，在定义了类成员函数的.CPP文件中使用BEGIN_PROPPAGEIDS宏，再加入控制的属性页的入口，最后用END_PROPPAGEIDS宏来声明属性页列表的结束。

关于属性页的更多信息参见“Visual C++程序员指南”中的文章“ActiveX控制：属性页”。

请参阅 BEGIN_PROPPAGEIDS, END_PROPPAGEIDS

DECLARE_SERIAL

DECLARE_SERIAL(*class_name*)

参数

class_name

类的实际名字（不用引号括起来）。

说明

DECLARE_SERIAL为可以串行化的CObject的派生类生成了必要的C++代码。

串行化是指将对象的内容写入文件或从文件读入对象的内容的过程。

在.H 模块中使用 DECLARE_SERIAL 宏，然后在所有需要访问这个类的对象的.CPP 模块中包含这个模块。

如果在类声明中包含了 DECLARE_SERIAL 宏，那么必须在类实现中包含 IMPLEMENT_SERIAL 宏。

DECLARE_SERIAL 宏包含了 DECLARE_DYNAMIC 和 DECLARE_DYNCREATE 的所有功能。

你可以用 AFX_API 宏为那些使用了 DECLARE_SERIAL 和 IMPLEMENT_SERIAL 宏的类自动引出 CArchive 提取操作符。用下面的代码将类声明（在.H 文件中）括起来：

```
#undef AFX_API
#define AFX_API AFX_EXT_CLASS
<这里是你的类声明>
#undef AFX_API
#define AFX_API
```

关于 DECLARE_SERIAL 宏的更多信息参见“Visual C++程序员指南”中的“CObject 类主题”。

请参阅 DECLARE_DYNAMIC, IMPLEMENT_SERIAL, RUNTIME_CLASS, CObject::IsKindOf

DEFAULT_PARSE_COMMAND

DEFAULT_PARSE_COMMAND(*FnName, mapClass*)

参数

FnName
　　成员函数的名字。也是命令的名字。

mapClass
　　函数映射的目的类的名字。

说明

如果一个客户向 CHttpServer **对象提出的请求不包括命令，**DEFAULT_PARSE_COMMAND **宏就指示框架调入** *FnName* **参数指定的缺省页面。**

DEFAULT_PARSE_COMMAND **宏可以出现在解析映射的任何位置。**

解析映射的例子参看 ON_PARSE_COMMAND。

请参阅 BEGIN_PARSE_MAP, ON_PARSE_COMMAND, ON_PARSE_COMMAND_PARAMS, END_PARSE_MAP, CHttpServer

DestructElements

```
template< class TYPE >
void AFXAPI DestructElements( TYPE* pElements, int nCount );
```

参数

TYPE

　　指定了要销毁的元素类型的模板参数。

pElements

　　指向元素的指针。

nCount

　　要销毁的元素的数目。

说明

当元素要被销毁时，CArray**，**CList **和** CMap **的类成员调用这个函数。**

缺省的实现不做任何操作。有关这个函数以及其它帮助函数的实现的信息参见 Visual C++**程序员指南中的文章“集合：如何生成类型安全的集合”。**

请参阅 CArray, CList, CMap

DFX_Binary

void AFXAPI DFX_Binary(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, CByteArray& *value*, int *nPreAllocSize* = AFX_DAO_BINARY_DEFAULT_SIZE, DWORD *dwBindOptions* = 0);

参数

pFX

指向 CDaoFieldExchange 类的对象的指针。这个对象包含函数调用的环境信息。有关 CDaoFieldExchange 对象能够指定的操作的附加信息参见 Visual C++程序员指南中的文章“DAO 记录字段交换：DFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，CByteArray 类型的值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

nPreAllocSize

应用框架预分配这么多内存。如果你的数据更多，框架会在必要时分配更多的内存。要获得更好的性能，将这个大小设为一个足够大的值以避

免重分配。在 AFXDAO.H 中，缺省的大小被定义为 AFX_DAO_BINARY_DEFAULT_SIZE。

dwBindOptions

使你能够享受 MFC 的双缓冲机制好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_DISABLE_FIELD_CACHE 并不使用双缓冲机制，你必须自己调用 SetFieldDirty 和 SetFieldNull。另外一个可能值 AFX_DAO_ENABLE_FIE- LD_CACHE 使用双缓冲机制，你不必进行额外的操作以将一个字段标记为脏的或 NULL。由于性能和内存方面的原因，不要使用这个值，除非你的二进制数据确实很小。

这些选项在 Visual C++程序员指南的“DAO 记录字段交换：双缓冲记录”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_Binary 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换字节数组。数据在 DAO 中的 DAO_BYTES 类型和记录集中的 CByteArray 类型之间进行映射。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Short, DFX_Single, DFX_Double, DFX_DateTime, DFX_Byte, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Bool

void AFXAPI DFX_Bool(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, BOOL& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange **类的对象的指针。这个对象包含函数调用的环境信息。有关** CDaoFieldExchange **对象能够指定的操作的附加信息参见“**Visual C++**程序员指南”中的文章“**DAO **记录字段交换：**DFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，BOOL 类型的值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

dwBindOptions

使你能够享受 MFC 的双缓冲机制好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_BOOL 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换布尔数据。数据在 DAO 中的 DAO_BOOL 类型和记录集中的 BOOL 类型之间进行映射。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Long, DFX_Currency, DFX_Short, DFX_Single, DFX_Double, DFX_DateTime, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Byte

void AFXAPI DFX_Byte(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, BYTE& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange **类的对象的指针。这个对象包含函数调用的环境信息。有关** CDaoFieldExchange **对象能够指定的操作的附加信息参见“**Visual C++**程序员指南”中的文章“**DAO **记录字段交换：**DFX **如何工作”。**

szName

数据列的名字。

value

指定数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，BYTE 类型的值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

dwBindOptions

使你能够享受 MFC 的双缓冲机制好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个值可能是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_BYTE 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换单字节数据。数据在 DAO 中的 DAO_BYTES 类型和记录集中的 BYTE 类型之间进行映射。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Short, DFX_Single, DFX_Double, DFX_DateTime, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Currency

void AFXAPI DFX_Currency(CDaoFieldExchange* pFX, LPCTSTR *szName*, COleCurrency& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange **类的对象的指针。这个对象包含函数调用的环境信息。有关** CDaoFieldExchange **对象能够指定的操作的附加信息参见“**Visual C++**程序员指南”中的文章“**DAO **记录字段交换：**DFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，该 COleCurrency 值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_Currency 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换货币数据。数据在 DAO 中的 DAO_CURRENCY 类型和记录集中的 COleCurrency 类型之间进行映射。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Bool, DFX_DateTime, DFX_Long, DFX_Short, DFX_Single, DFX_Double, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_DateTime

void AFXAPI DFX_DateTime(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, COleDateTime& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange **类的对象的指针。这个对象包含函数调用的环境信息。有关** CDaoFieldExchange **对象能够指定的操作的附加信息参见“**Visual C++**程序员指南”中的文章“**DAO **记录字段交换：**DFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。这个函数保存着对一个COleDateTime对象的引用。对于从记录集到数据源的数据传送，该值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

dwBindOptions

使你能够享受MFC的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值AFX_DAO_ENABLE_FIELD_CACHE使用双缓冲机制。另外一个可能值是AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC并不对这个字段进行检查。你必须自己调用SetFieldDirty和SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置CDaoRecordset::m_bCheckCacheForDirtyFields来控制是否对数据使用双缓冲机制。

说明

DFX_DateTime函数在CDaoRecordset对象的字段数据成员和数据源中记录的

列之间交换日期和时间数据。数据在 DAO 中的 DAO_DATE 类型和记录集中的 COleDateTime 类型之间进行映射。

注意 COleDateTime 取代了 DAO 中用于相同目的的 CTime 和 TIMESTAMP_STRUCT。但是 CTime 和 TIMESTAMP_STRUCT 仍被用于基于 ODBC 的数据访问类。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Short, DFX_Single, DFX_Double, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Double

void AFXAPI DFX_Double(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, double& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange 类的对象的指针。这个对象包含函数调用的环境信息。有关 CDaoFieldExchange 对象能够指定的操作的附加信息参见“Visual C++程序员指南”中的文章“DAO 记录字段交换：DFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，该 double 值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields

来控制是否对数据使用双缓冲机制。

说明

DFX_Double 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换 double 和 float 数据。数据在 DAO 中的 DAO_R8 类型和记录集中的 double 以及 float 类型之间进行映射。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Short, DFX_Single, DFX_DateTime, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Long

void AFXAPI DFX_Long(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, long& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange 类的对象的指针。这个对象包含函数调用的环境信息。有关 CDaoFieldExchange 对象能够指定的操作的附加信息参见“Visual C++程序员指南”中的文章“DAO 记录字段交换：DFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，该 long 值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”

一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

.DFX_Long 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换长整型数据。数据在 DAO 中的 DAO_I4 类型和记录集中的 long 类型之间进行映射。

示例

参见 DFX_Text.

请参阅　DFX_Text, DFX_Bool, DFX_Currency, DFX_Short, DFX_Single, DFX_Double, DFX_DateTime, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_LongBinary

void AFXAPI DFX_LongBinary(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, CLongBinary& *value*, DWORD *dwPreAllocLenth* = AFX_DAO_LONGBINARY_DEFAULT_SIZE, DWORD dwBindOptions = 0);

参数

pFX

指向 CDaoFieldExchange 类的对象的指针。这个对象包含函数调用的环境信息。有关 CDaoFieldExchange 对象能够指定的操作的附加信息参见"Visual C++程序员指南"中的文章"DAO 记录字段交换：DFX 如何工作"。

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，CLongBinary 类型的值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。

nPreAllocSize

应用框架预分配这么多内存。如果你的数据更多，框架会在必要时分配更多的内存。如果要获得更好的性能，将这个大小设为一个足够大的值以避免重分配。

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_FIELD_CACHE 并不使用双缓冲机

制，你必须自己调用 SetFieldDirty 和 SetFieldNull。另外一个可能值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制，不必进行额外的操作以将一个字段标记为脏的或 NULL。由于性能和内存方面的原因，不要使用这个值，除非你的二进制数据确实很小。

这些选项在“Visual C++程序员指南”的“DAO 记录字段交换：双缓冲记录”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

重点　我们推荐使用 DFX_Binary 来代替这个函数。DFX_LongBinary 是为了与 MFC 的 ODBC 类兼容而提供的。

DFX_LongBinary 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换大二进制对象（BLOB）数据。数据在 DAO 中的 DAO_BYTES 类型和记录集中的 CLongBinary 类型之间进行映射。

示例

参见 DFX_Text.

请参阅　DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Short,

DFX_Single, DFX_Double, DFX_DateTime, DFX_Byte, CDaoFieldExchange::SetFieldType, CLongBinary

DFX_Short

void AFXAPI DFX_Short(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, short& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange **类的对象的指针。这个对象包含函数调用的环境信息。有关** CDaoFieldExchange **对象能够指定的操作的附加信息参见“** Visual C++**程序员指南”中的文章“** DAO **记录字段交换：**DFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员中保存的值––––**要被传送的值。对于从记录集到数据源的数据传送，该** short **值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。**

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意 可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_Short 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换短整型数据。数据在 DAO 中的 DAO_I2 类型和记录集中的 short 类型之间进行映射。

示例

参见 DFX_Text.

请参阅 DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Single,

DFX_Double, DFX_DateTime, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Single

void AFXAPI DFX_Single(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, float& *value*, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

　　指向 CDaoFieldExchange **类的对象的指针。这个对象包含函数调用的环境信息。有关** CDaoFieldExchange **对象能够指定的操作的附加信息参见“**Visual C++**程序员指南”中的文章“**DAO **记录字段交换：**DFX **如何工作”。**

szName

　　数据列的名字。

value

　　指定的数据成员中保存的值----**要被传送的值。对于从记录集到数据源的数据传送，该** float **值是从指定的数据成员中获取的。对于从数据源到记录集的数据传送，该值是保存在指定的数据成员中的。**

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_Single 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换浮点数据。数据在 DAO 中的 DAO_R4 类型和记录集中的 float 类型之间进行映射。

示例

参见 DFX_Text.

请参阅　DFX_Text, DFX_Bool, DFX_Currency, DFX_Long, DFX_Short,

DFX_Double, DFX_DateTime, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DFX_Text

void AFXAPI DFX_Text(CDaoFieldExchange* *pFX*, LPCTSTR *szName*, CString& *value*, int *nPreAllocLength* = AFX_DAO_TEXT_DEFAULT_SIZE, DWORD *dwBindOptions* = AFX_DAO_ENABLE_FIELD_CACHE);

参数

pFX

指向 CDaoFieldExchange 类的对象的指针。这个对象包含函数调用的环境信息。有关 CDaoFieldExchange 对象能够指定的操作的附加信息参见 Visual C++程序员联机指南中的文章“DAO 记录字段交换：DFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员中保存的值----要被传送的值。对于从记录集到数据源的数据传送，该 CString 值是从指定的数据成员中获取的。对于从数据

源到记录集的数据传送，该值是保存在指定的数据成员中的。

nPreAllocSize

应用框架预分配这么多内存。如果你的数据更多，框架会在必要时分配更多的空间。如果要获得更好的性能，将这个大小设为一个足够大的值以避免重分配。

dwBindOptions

使你能够享受 MFC 的双缓冲机制的好处的选项，该机制能够检测发生了变化的记录集字段。缺省值 AFX_DAO_ENABLE_FIELD_CACHE 使用双缓冲机制。另外一个可能值是 AFX_DAO_DISABLE_FIELD_CACHE。如果你指定了这个值，MFC 并不对这个字段进行检查。你必须自己调用 SetFieldDirty 和 SetFieldNull。

这些选项在“Visual C++程序员指南”的“DAO 记录集：动态记录绑定”一文中有进一步的解释。

注意　可以通过设置 CDaoRecordset::m_bCheckCacheForDirtyFields 来控制是否对数据使用双缓冲机制。

说明

DFX_TEXT 函数在 CDaoRecordset 对象的字段数据成员和数据源中记录的列之间交换 CString 数据。数据在 DAO 中的 DAO_CHAR（如果定义了_UNICODE

则为 DAO_WCHAR）与记录集中的 CString 之间映射。

示例

这个例子演示了对 DFX_TEXT 的一些调用。同时注意对 CDaoFieldExchange::SetFieldType 的两个调用。通常 ClassWizard 把对 SetFieldType 和相关 DFX 的调用写入第二个调用。你必须编写第一个调用及其 DFX 调用。建议你将所有的参数放在“//{{AFX_FIELD_MAP”注释之前。必须将参数写在注释之外。

```
// DFX_Text 的例子
void CSections::DoFieldExchange(CDaoFieldExchange* pFX)
{
	pFX->SetFieldType(CDaoFieldExchange::param);
	DFX_Text(pFX, "Name", m_strNameParam);
	//{{AFX_FIELD_MAP(CSections)
	pFX->SetFieldType(CDaoFieldExchange::outputColumn);
	DFX_Text(pFX, "CourseID", m_strCourseID);
	DFX_Text(pFX, "InstructorID", m_strInstructorID);
	DFX_Short(pFX, "LabFee", m_nRoomNo);
	DFX_Text(pFX, "LabFee", m_strSchedule);
	DFX_Short(pFX, "SectionNo", m_nSectionNo);
	DFX_Currency(pFX, "LabFee", m_currLabFee);
```

```
    //}}AFX_FIELD_MAP
}
```

请参阅 DFX_Bool, DFX_Long, DFX_Currency, DFX_Short, DFX_Single, DFX_Double, DFX_DateTime, DFX_Byte, DFX_Binary, DFX_LongBinary, CDaoFieldExchange::SetFieldType

DISP_DEFVALUE

DISP_DEFVALUE(*theClass*, *pszName*)
#include <afxdisp.h>

参数

theClass
　　类的名字。

pszName
　　代表了对象的值的属性的外部名。

说明

这个宏使一个已有的属性成为对象的缺省值。使用缺省值可以简化 Visual Basic

应用程序中的自动化对象的设计。

对象的缺省值指在引用对象时，在没有指定属性或成员函数的情况下而获得或设置的值。

请参阅　调度映射, DECLARE_DISPATCH_MAP, DISP_PROPERTY_EX, DISP_FUNCTION, BEGIN_DISPATCH_MAP, END_DISPATCH_MAP

DISP_FUNCTION

DISP_FUNCTION(*theClass, pszName, pfnMember, vtRetVal, vtsParams*)
#include <afxdisp.h>

参数

theClass
　　类的名字。

pszName
　　函数的外部名字。

pfnMember
　　成员函数的名字。

vtRetVal

指定了函数返回类型的值。

vtsParams

指定了函数参数表的一个或多个常量的用空格分隔的列表。

说明

DISP_FUNCTION 宏被在调度映射中使用，用来定义一个 OLE 自动化函数。

vtRetVal 参数属于 VARTYPE 类型。这个参数的可能取值来自 VARENUM 枚举，如下：

符号	返回类型
VT_EMPTY	void
VT_I2	short
VT_I4	long
VT_R4	float
VT_R8	double
VT_CY	CY
VT_DATE	DATE
VT_BSTR	BSTR
VT_DISPATCH	LPDISPATCH
VT_ERROR	SCODE

续表

VT_BOOL	BOOL
VT_VARIANT	VARIANT
VT_UNKNOWN	LPUNKNOWN

vtsParams 参数是 VTS_ 常量中取值的用空格分隔的列表。有空格分隔的一个或多个取值的列表指定了函数的参数列表。例如：

VTS_I2 VTS_PI2

指定了包含一个短整数以及后面的短整数指针的列表。

The VTS_ 常量及其含义如下：

符号	参数类型
VTS_I2	short
VTS_I4	long
VTS_R4	float
VTS_R8	double
VTS_CY	Const CY or CY*
VTS_DATE	DATE
VTS_BSTR	LPCSTR
VTS_DISPATCH	LPDISPATCH
VTS_SCODE	SCODE
VTS_BOOL	BOOL

续表

VTS_VARIANT	Const VARIANT* or VARIANT&
VTS_UNKNOWN	LPUNKNOWN
VTS_PI2	short*
VTS_PI4	long*
VTS_PR4	float*
VTS_PR8	double*
VTS_PCY	CY*
VTS_PDATE	DATE*
VTS_PBSTR	BSTR*
VTS_PDISPATCH	LPDISPATCH*
VTS_PSCODE	SCODE*
VTS_PBOOL	BOOL*
VTS_PVARIANT	VARIANT*
VTS_PUNKNOWN	LPUNKNOWN*

请参阅 **调度映射**, DECLARE_DISPATCH_MAP, DISP_PROPERTY, DISP_PROPERTY_EX, BEGIN_DISPATCH_MAP, END_DISPATCH_MAP

DISP_PROPERTY

DISP_PROPERTY(*theClass, pszName, memberName, vtPropType*)

#include <afxdisp.h>

参数

theClass

类的名字。

pszName

属性的外部名字。

memberName

存储属性的成员变量的名字。

vtPropType

指定属性类型的值。

说明

DISP_PROPERTY **用在调度映射中，用于定义一个** OLE **自动化属性。**

vtPropType **参数属于** VARTYPE **类型。这个参数的取值可能来自** VARENUM

枚举：

符号	属性类型
VT_I2	Short
VT_I4	long
VT_R4	float
VT_R8	double
VT_CY	CY
VT_DATE	DATE
VT_BSTR	CString
VT_DISPATCH	LPDISPATCH
VT_ERROR	SCODE
VT_BOOL	BOOL
VT_VARIANT	VARIANT
VT_UNKNOWN	LPUNKNOWN

当一个外部客户改变了这个属性的时候，由 *memberName* **指定的成员变量的值就发生了改变。对于这个改变，没有任何通知。**

请参阅　调度映射, DECLARE_DISPATCH_MAP, DISP_PROPERTY_EX, DISP_FUNCTION, BEGIN_DISPATCH_MAP, END_DISPATCH_MAP

DISP_PROPERTY_EX

DISP_PROPERTY_EX(*theClass*, *pszName*, *memberGet*,*memberSet*, *vtPropType*)

#include <afxdisp.h>

参数

theClass

类的名字。

pszName

属性的外部名字。

memberGet

用于获取属性的成员变量的名字。

memberSet

用于设置属性的成员变量的名字。

vtPropType

指定属性类型的值。

说明

DISP_PROPERTY_EX 用在调度映射中，用于定义一个 OLE 自动化属性并命名获取、设置该属性的函数。

memberGet 和 *memberSet* 函数具有 *vtPropType* 参数所决定的签名。*memberGet* 函数没有参数，它返回 *vtPropType* 指定的类型的值。*memberSet* 函数具有一个 *vtPropType* 所指定的类型的参数，它没有返回值。

vtPropType 参数属于 VARTYPE 类型。这个参数的可能取值来自 VARENUM 枚举。有关这些值的列表参见对 DISP_FUNCTION 中 *vtRetVal* 参数的说明。注意在 DISP_FUNCTION 中列出的 VT_EMPTY 不能作为属性数据类型。

请参阅 调度映射，DECLARE_DISPATCH_MAP, DISP_PROPERTY,DISP_FUNCTION, BEGIN_DISPATCH_MAP, END_DISPATCH_MAP

DISP_PROPERTY_NOTIFY

DISP_PROPERTY_NOTIFY(*theClass,szExternalName, memberName,pfnAfterSet,vtPropType*)

#include <afxdisp.h>

参数

theClass

类的名字。

szExternalName

属性的外部名字。

memberName

保存属性的成员变量的名字。

pfnAfterSet

szExternalName 的通知函数的名字。

vtPropType

指定了属性类型的值。

说明

DISP_PROPERTY_NOTIFY 宏用在调度映射中，用来定义一个带通知的 OLE 自动化属性。与用 DISP_PROPERTY 定义的属性不同，当属性改变时，用 DISP_PROPERTY_NOTIFY 定义的属性自动地调用 *pfnAfterSet* 指定的函数。

vtPropType 参数属于 VARTYPE 类型。这个参数的可能取值来自 VARENUM 枚举：

符号	属性类型
VT_I2	short
VT_I4	long
VT_R4	float
VT_R8	double
VT_CY	CY
VT_DATE	DATE
VT_BSTR	CString
VT_DISPATCH	LPDISPATCH
VT_ERROR	SCODE
VT_BOOL	BOOL
VT_VARIANT	VARIANT
VT_UNKNOWN	LPUNKNOWN

请参阅 **调度映射**, DISP_PROPERTY, DISP_FUNCTION

DISP_PROPERTY_PARAM

DISP_PROPERTY_NOTIFY(*theClass,pszExternalName,pfnGet,pfnSet, vtPropType, vtsParams*)

#include <afxdisp.h>

参数

theClass

类的名字。

pszExternalName

属性的外部名字。

pfnGet

用于获取属性的成员函数名。

pfnSet

用于设置属性的成员函数名。

vtPropType

指定属性类型的值。

vtsParams

用空格分隔的 VTS_变量参数类型字符串，每一个代表一个参数。

说明

这个宏定义了分别用 Get 和 Set 成员函数访问的属性。与 DISP_PROPERTY_EX 宏不同，这个宏允许你为属性指定一个参数列表。这在实现具有索引或参数的属性时非常有用。

例如，考虑下面对 get 和 set 成员函数的声明，它允许用户在访问属性的时候指定行和列：

```
afx_msg short GetArray(short row, short column);
afx_msg short SetArray(short row, short column, short nNewValue);
```

这些与控制的调度映射中的 DISP_PROPERTY_PARAM 宏相对应：

```
DISP_PROPERTY_PARAM(CMyCtrl, "Array", GetArray, SetArray, VT-I2,
VTS_I2 VTS_I2)
```

另一个例子，考虑下面的 get 和 set 成员函数：

```
LPDISPATCH CMyObject::GetItem(short index1, short index2, short index3);
void CMyObject::SetItem(short index1, short index2, short index3, LPDISPATCH
newValue);
```

这些与控制的调度映射中的 DISP_PROPERTY_PARAM 宏相对应：

```
DISP_PROPERTY_PARAM(CMyObject, "item", GetItem, SetItem,
VT_DISPATCH,VTS_I2
↪ VTS_I2 VTS_I2)
```

请参阅　调度映射, DISP_PROPERTY _EX

DumpElements

template< class *TYPE* >
void AFXAPI DumpElements(CDumpContext& *dc*, const *TYPE** *pElements*, int *nCount*);

参数

dc

用于转储元素的环境。

TYPE

指定元素类型的模板参数。

pElements

指向要转储的元素的指针。

nCount

要转储的元素的数目。

说明

重载这个函数以为你的集合中的元素提供面向流的文本形式的诊断输出。如果转储的深度大于 0**，则** CArray::Dump**，**CList::Dump **和** CMap::Dump **都会调用**

这个函数。

缺省的实现不做任何操作。如果你的集合的元素是从 CObject 继承的，通常你的重载函数应当在集合的元素之间重复，按次序为每个元素调用 Dump。

有关诊断和 Dump 函数的信息参见“Visual C++程序员指南”中的“MFC 调试支持”。

请参阅 CDumpContext::SetDepth, CObject::Dump, CArray, CList, CMap

DYNAMIC_DOWNCAST

DYNAMIC_DOWNCAST(*class, pointer*)

参数

class

类的名字。

pointer

将要被强制转换为 class 类对象指针的指针。

说明

DYNAMIC_DOWNCAST 宏提供了将一个指针强制转换为指向一个类的对象的

指针的简单方式，同时检查这种强制转换是否合法。这个宏将会把 *pointer* 参数强制转换为 *class* 参数类型的对象指针。

如果 pointer 参数所指向的对象就是 class 类的对象，这个宏将返回一个适当的指针。如果这种转换不合法，这个宏将返回 NULL。

请参阅 STATIC_DOWNCAST

END_CATCH

END_CATCH

说明

标记了上一个 CATCH 或 AND_CATCH 块的结束。

关于 END_CATCH 宏的更多信息参见“Visual C++程序员指南”中的章节“异常”。

请参阅 TRY, CATCH, AND_CATCH, THROW, THROW_LAST

END_CATCH_ALL

END_CATCH_ALL

说明

标记了上一个 CATCH_ALL 或 AND_CATCH_ALL 块的结束。

请参阅 TRY, CATCH_ALL, AND_CATCH_ALL, THROW, THROW_LAST

END_CONNECTION_MAP

END_CONNECTION_MAP( )

说明

使用 END_CONNECTION_MAP 宏以结束你的连接映射的定义。

请参阅 BEGIN_CONNECTION_MAP, DECLARE_CONNECTION_MAP

END_CONNECTION_PART

END_CONNECTION_PART(*localClass*)

参数

localClass
　　指定了实现连接点的本地类的名字。

说明

使用 END_CONNECTION_PART 宏以结束你的连接点的定义。

请参阅 BEGIN_CONNECTION_PART, DECLARE_CONNECTION_MAP

END_DISPATCH_MAP

END_DISPATCH_MAP( )

#include <afxdisp.h>

说明

使用 END_DISPATCH_MAP 宏以结束你的调度映射的定义。它必须与 BEGIN_DISPATCH_ MAP 结合使用。

请参阅 调度映射, DECLARE_DISPATCH_MAP, BEGIN_DISPATCH_MAP, DISP_FUNCTION, DISP_PROPERTY, DISP_PROPERTY_EX, DISP_DEFVALUE

END_EVENT_MAP

END_EVENT_MAP( )

说明

使用 END_EVENT_MAP 宏以结束你的事件映射的定义。

请参阅 DECLARE_EVENT_MAP, BEGIN_EVENT_MAP

END_EVENTSINK_MAP

END_EVENTSINK_MAP( )

说明

使用 END_EVENTSINK_MAP 宏以结束你的事件接收映射的定义。

请参阅 DECLARE_EVENTSINK_MAP, BEGIN_EVENTSINK_MAP

END_MESSAGE_MAP

END_MESSAGE_MAP( )

说明

使用 END_MESSAGE_MAP 宏以结束你的消息映射的定义。

有关消息映射和 END_MESSAGE_MAP 宏的更多信息参见“Visual C++程序员指南”中的“消息处理和映射”主题。

请参阅 DECLARE_MESSAGE_MAP,BEGIN_MESSAGE_MAP, Message Map Function Categories

END_OLEFACTORY

END_OLEFACTORY(*class_name*)

参数

class_name

类工厂所属的控制类的名字。

说明

使用 END_OLEFACTORY 宏以结束你的控制类工厂的定义。

请参阅 BEGIN_OLEFACTORY, DECLARE_OLECREATE_EX

END_PARSE_MAP

END_PARSE_MAP(theClass)

参数

theClass

指定了拥有这个解析映射的类的名字。

说明

使用 END_PARSE_MAP **宏来结束你的解析映射的定义。它必须与** BEGIN_PARSE_MAP **结合使用。**
解析映射的例子参见 ON_PARSE_COMMAND。

请参阅 BEGIN_PARSE_MAP, ON_PARSE_COMMAND,
ON_PARSE_COMMAND_PARAMS,
DEFAULT_PARSE_COMMAND, CHttpServer

END_PROPPAGEIDS

END_PROPPAGEIDS(*class_name*)

参数

class_name

拥有这个属性页的控制类的名字。

说明

使用 END_PROPPAGEIDS 宏以结束你的属性页 ID 列表的定义。

请参阅 DECLARE_PROPPAGEIDS, BEGIN_PROPPAGEIDS

EVENT_CUSTOM

EVENT_CUSTOM(*pszName, pfnFire, vtsParams*)

参数

pszName

事件的名字。

pfnFire

事件引发的函数的名字。

vtsParams

用空格分隔的一个或多个常量的列表，指定了函数的参数列表。

说明

使用 EVENT_CUSTOM 宏来为一个自定义事件定义事件映射入口。
vtsParams 参数是来自 VTS_常量的值的列表，用空格分隔。其中的一个或多个值指定了函数的参数列表。例如：
VTS_COLOR VTS_FONT
指定了包含短整数以及后面的 BOOL 值的列表。
VTS_ 常量及其含义如下：

符号	参数类型
VTS_I2	short
VTS_I4	long
VTS_R4	float
VTS_R8	double
VTS_COLOR	OLE_COLOR
VTS_CY	CURRENCY
VTS_DATE	DATE
VTS_BSTR	constchar*
VTS_DISPATCH	LPDISPATCH
VTS_FONT	IfontDispatch*
VTS_HANDLE	HANDLE

续表

VTS_SCODE	SCODE
VTS_BOOL	BOOL
VTS_VARIANT	constVARIANT*
VTS_PVARIANT	VARIANT*
VTS_UNKNOWN	LPUNKNOWN
VTS_OPTEXCLUSIVE	OLE_OPTEXCLUSIVE
VTS_PICTURE	IpictureDisp*
VTS_TRISTATE	OLE_TRISTATE
VTS_XPOS_PIXELS	OLE_XPOS_PIXELS
VTS_YPOS_PIXELS	OLE_YPOS_PIXELS
VTS_XSIZE_PIXELS	OLE_XSIZE_PIXELS
VTS_YSIZE_PIXELS	OLE_YSIZE_PIXELS
VTS_XPOS_HIMETRIC	OLE_XPOS_HIMETRIC
VTS_YPOS_HIMETRIC	OLE_YPOS_HIMETRIC
VTS_XSIZE_HIMETRIC	OLE_XSIZE_HIMETRIC
VTS_YSIZE_HIMETRIC	OLE_YSIZE_HIMETRIC

注意　另外，除了VTS_FONT和VTS_PICTURE以外，还为所有的可变类型定义了可变常量，提供了指向可变数据常量的指针。这些常量按照VTS_PConstantname约定来命名。例如，VTS_PCOLOR值一个指向VTS_COLOR常量的指针。

请参阅 EVENT_CUSTOM_ID, DECLARE_EVENT_MAP

EVENT_CUSTOM_ID

EVENT_CUSTOM_ID(*pszName, dispid, pfnFire, vtsParams*)

参数

pszName

  事件的名字。

dispid

  当引发事件时由控制使用的调度 ID。

pfnFire

  事件引发的函数的名字。

vtsParams

  当事件被引发时，传递个控制容器的参数变量列表。

说明

使用 EVENT_CUSTOM_ID 宏来为属于 *dispid* 指定的调度 ID 的自定义事件定义一个事件引发函数。

vtsParams 参数是 VTS_常量中的值的列表。列表中用空格隔开的值指定了函数的参数列表。例如：

VTS_COLOR VTS_FONT

指定了包含短整数和后面的 BOOL 量的列表。
VTS_常量的列表参见 EVENT_CUSTOM。

请参阅 EVENT_CUSTOM

HashKey

```
template< class ARG_KEY > UINT AFXAPI HashKey( ARG_KEY key );
```

返回值

key 的散列值。

参数

ARG_KEY
指定了用于访问映射键的数据类型的模板参数。

key

要被计算散列值的键。

说明

计算给定键值的散列值。

这个函数被 CMap::RemoveKey 直接调用，CMap::Lookup 和 CMap::Operator[] 间接调用它。

缺省的实现将键值向右移动四个位置，生成一个散列值。可以重载这个函数，使它返回适用于你的应用程序的散列值。

请参阅 CMap

IMPLEMENT_DYNAMIC

IMPLEMENT_DYNAMIC(*class_name*, *base_class_name*)

参数

class_name

类的实际名字（不用引号括起来）。

base_class_name

基类的名字（不要引号括起来）。

说明

这个宏为动态的 CObject 派生类生成了必要的 C++代码，使其能够在运行时访问类的名字及其在继承结构中的位置。在.CPP 模块中使用 IMPLEMENT_DYNAMIC 宏，然后一次性地连接生成的目标代码。

有关的更多信息参见“Visual C++程序员指南”中的“CObject 类”主题。

请参阅 DECLARE_DYNAMIC, RUNTIME_CLASS, CObject::IsKindOf

IMPLEMENT_DYNCREATE

IMPLEMENT_DYNCREATE(*class_name*, *base_class_name*)

参数

class_name

类的实际名字（不用引号括起来）。

base_class_name

基类的实际名字（不用引号括起来）。

说明

与 DECLARE_DYNCREATE 宏一起使用 IMPLEMENT_DYNCREATE 宏，使 CObject 的派生类的对象能够在运行时被动态创建。框架利用这种能力来动态地创建新对象，例如，当它在串行化的过程中从磁盘读取对象的时候。在类的实现文件中加入 IMPLEMENT_DYNCREATE 宏。更多的信息参见“Visual C++ 程序员指南”中的“CObject 对象”主题。

如果你使用了 DECLARE_DYNCREATE 和 IMPLEMENT_DYNCREATE 宏，就可以使用 RUNTIME_CLASS 宏以及 CObject::IsKindOf 成员函数在运行时确定对象所属的类。

如果在类的声明中包含了 DECLARE_DYNCREATE，那么就必须在类的实现中包含 IMPLEMENT_DYNCREATE。

请参阅 DECLARE_DYNCREATE, RUNTIME_CLASS, CObject::IsKindOf

IMPLEMENT_OLECREATE

IMPLEMENT_OLECREATE(*class_name, external_name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8*)

#include <afxdisp.h>

参数

class_name

类的实际名字（不用引号括起来）。

external_name

对其它应用程序公开的对象名（不用引号括起来）。

l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8

类的 CLSID 的组成部分。

说明

这个宏必须出现在使用 DECLARE_OLECREATE 的类的实现文件中。

外部名是对其它应用程序公开的标识符。客户应用程序利用外部名来向自动化服务器的类对象发出请求。

OLE 的类 ID 是对象的唯一的 128 位标识符。它组成了一个 long 值，两个 WORD 值以及八个 BYTE 值，在句法描述中用 *l*，*w1*，*w2* 以及 b1 到 b8 来表示。ClassWizard 和 AppWizard 根据你的请求生成唯一的 OLE 类 ID。

请参阅 DECLARE_OLECREATE, CLSID Key

IMPLEMENT_OLECREATE_EX

IMPLEMENT_OLECREATE_EX(*class_name, external_name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8*)

参数

class_name

属性页类控制的名字。

external_name

对应用程序公开的对象名。

l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8

类的 CLSID **的组成部分。有关这些参数的详细信息参见** IMPLEMENT_OLECREATE **的说明。**

说明

这个宏实现你的控制的类工厂以及控制类的 GetClassID **成员函数。这个宏必须出现在使用了** DECLARE_OLECREATE_EX **宏或** BEGIN_OLEFACTORY **以及** END_OLEFACTORY **的类的控制的实现文件中。外部名是对其它应用程序公开的** OLE **控制的标识符。容器程序通过这个名字向控制类发出请求。**

请参阅 DECLARE_OLECREATE_EX, BEGIN_OLEFACTORY, END_OLEFACTORY, IMPLEMENT_OLECREATE

IMPLEMENT_OLETYPELIB

IMPLEMENT_OLETYPELIB(*class_name, tlid, wVerMajor, wVerMinor*)

参数

class_name
与类型库相关的控制类的名字。

tlid
类型库的 ID 数字。

wVerMajor
类型库的主版本号。

wVerMinor
类型库的次版本号。

说明

这个宏实现了控制的 GetTypeLib 成员函数。它必须出现在使用

DECLARE_OLETYPELIB 宏的控制的实现文件中。

请参阅 DECLARE_OLETYPELIB

IMPLEMENT_SERIAL

IMPLEMENT_SERIAL(*class_name*, *base_class_name*, *wSchema*)

参数

class_name

类的实际名字（不用引号括起来）。

base_class_name

基类的名字（不用引号括起来）。

wSchema

一个 UINT 类型的版本号，将被用在存档中，使得解串行程序能够识别并处理早期版本的程序所生成的数据。它的值不能是－1。

说明

这个宏为动态的 CObject 派生类对象生成必要的 C++代码，使它能够在运行时访问类名及其在继承关系中的位置。在.CPP 模块中使用 IMPLEMENT_SERIAL

宏，然后一次性地连接生成的目标代码。

你可以使用 AFX_API 来为使用了 DECLARE_SERIAL 和 IMPLEMENT_SERIAL 宏的类自动引出 CArchive 提取操作符。用下面的代码把类声明（在.H 文件中）括起来：

```
#undef AFX_API
#define AFX_API AFX_EXT_CLASS
<这里是你的类声明>
#undef AFX_API
#define AFX_API
```

有关的更多信息参见“Visual C++程序员指南”中的“CObject 类”主题。

请参阅　DECLARE_SERIAL, RUNTIME_CLASS, CObject::IsKindOf

ISAPIASSERT

ISAPIASSERT(*booleanExpression*)

参数

booleanExpression

　　指定了一个表达式（包括指针变量），其计算结果为非零值或 0。

说明

这个宏的作用与 MFC 宏 ASSERT 完全类似。它计算其参数。如果结果为 0，这个宏就打印出一条诊断信息并退出程序。如果结果为非零值，它就不做任何操作。

诊断信息具有如下形式：

assertion failed in file <name> in line <num>

这里的 *name* 是源文件的名字，而 *num* 是源文件中产生断言失败的位置的行号。

在应用程序的发行版本中，ISAPIASSERT 并不计算表达式，因此也不会中断程序。如果不管环境如何，都必须计算这个表达式，可以用 ISAPIVERIFY 宏来代替 ISAPIASSERT。ISAPIASSERT 仅在应用程序的调试版本中起作用。

ISAPI 应用程序不一定要使用 MFC。如果你的应用程序没有与 MFC 连接，ISAPIASSERT 提供了与 ASSERT 相同的功能。如果你的应用程序与 MFC 连接，那么 ISAPIASSERT 仅简单地调用 MFC 的 ASSERT。

请参阅 ISAPITRACE, ISAPITRACE0, ISAPITRACE1, ISAPITRACE2, ISAPITRACE3, ISAPIVERIFY

ISAPITRACE

ISAPITRACE(*exp*)

参数

exp

指定了可变数目的参数，对它们的使用方式与运行时函数 printf 中的可变数目参数相同。

说明

这个宏的作用与 MFC 宏 TRACE 完全相似，TRACE 提供了与 printf 函数相似的功能，向诸如调试终端之类的转储环境发送一个格式化字符串。与 MS-DOS 下的 C 程序中的 printf 相似，ISAPITRACE 宏是在你的程序运行时跟踪变量值的简便途径。在调试环境中，ISAPITRACE 宏的输出被送到 Visual C++的调试窗口。在发行版本中，它什么也不做。

ISAPI 应用程序并没有必要使用 MFC。如果你的应用程序没有与 MFC 连接，ISPAITRACE 提供了与 TRACE 相同的功能。如果你的应用程序与 MFC 进行连接，那么 ISAPITRACE 简单地调用 MFC 的 TRACE。

请参阅 ISAPIASSERT, ISAPITRACE0, ISAPITRACE1, ISAPITRACE2, ISAPITRACE3, ISAPIVERIFY

ISAPITRACE0

ISAPITRACE0(*exp*)

参数

exp

格式化字符串，与运行时函数 printf 中所用的类似。

说明

ISAPITRACE0 是可以用来进行调试输出的一组跟踪宏中的一种形式。这组宏包括 ISAPITRACE0，ISAPITRACE1，ISAPITRACE2 和 ISAPITRACE3。这些宏之间的差别是它们所带参数的个数。ISAPITRACE0 只有一个格式化字符串，可以被用来输出简单的文本信息。ISAPITRACE1 具有格式化字符串和一个参数––––要被转储的变量。ISAPITRACE2 和 ISAPITRACE3 在格式化字符串后面分别带有两个和三个参数。

在你生成的应用程序的发行版本中,ISAPITRACE0 什么也不做。与 ISAPITRACE 一样，它仅在应用程序的调试版本中才把数据转储到调试输出设备。

ISAPITRACE0 的作用与 MFC 宏 TRACE0 完全类似。ISAPI 应用程序没有必要使用 MFC。如果你的应用程序没有与 MFC 连接,ISAPITRACE0 提供与 TRACE0 相同的功能。如果你的应用程序与 MFC 连接，ISAPITRACE0 简单地调用 MFC

的 TRACE0。

请参阅 ISAPIASSERT, ISAPITRACE, ISAPITRACE1, ISAPITRACE2, ISAPITRACE3, ISAPIVERIFY

ISAPITRACE1

ISAPITRACE1(*exp, param1*)

参数

exp

格式化字符串，与运行时函数 printf 中所用的类似。

param1

将要被转储的变量的名字。

说明

这个宏的作用与 MFC 宏 TRACE1 完全相似。有关 ISAPITRACE1 的描述参见 ISAPITRAC- E0。

ISAPI 应用程序没有必要使用 MFC。如果你的应用程序没有与 MFC 连接，ISAPITRACE1 提供与 TRACE1 相同的功能。如果你的应用程序与 MFC 连接，

ISAPITRACE1 简单地调用 MFC 的 TRACE1。

请参阅 ISAPIASSERT, ISAPITRACE, ISAPITRACE0, ISAPITRACE2, ISAPITRACE3, ISAPIVERIFY

ISAPITRACE2

ISAPITRACE2(*exp, param1, param2*)

参数

exp

格式化字符串，与运行时函数 printf 中所使用的类似。

param1, param2

要被转储的变量的名字。

说明

这个宏的作用与 MFC 宏 TRACE2 完全相似。有关 ISAPITRACE2 的描述参见 ISAPITRAC- E0。

ISAPI 应用程序没有必要使用 MFC。如果你的应用程序没有与 MFC 连接，ISAPITRACE2 提供与 TRACE2 相同的功能。如果你的应用程序与 MFC 连接，

ISAPITRACE2 简单地调用 MFC 的 TRACE2。

请参阅 ISAPIASSERT, ISAPITRACE, ISAPITRACE0, ISAPITRACE1, ISAPITRACE3, ISAPIVERIFY

ISAPITRACE3

ISAPITRACE3(*exp, param1, param2, param3*)

参数

exp

格式化字符串，与运行时函数 printf 中所使用的类似。

param1, param2, param3

要被转储的变量的名字。

说明

这个宏的作用与 MFC 宏 TRACE3 完全相似。有关 ISAPITRACE3 的描述参见 ISAPITRAC- E0。

ISAPI 应用程序没有必要使用 MFC。如果你的应用程序没有与 MFC 连接，ISAPITRACE3 提供与 TRACE3 相同的功能。如果你的应用程序与 MFC 连接，

ISAPITRACE3 简单地调用 MFC 的 TRACE3。

请参阅 ISAPIASSERT, ISAPITRACE, ISAPITRACE0, ISAPITRACE1, ISAPITRACE2, ISAPIVERIFY

ISAPIVERIFY

ISAPIVERIFY(*booleanExpression*)

参数

booleanExpression

指定了一个表达式（包括指针变量），将被计算，结果为非零值或 0。

说明

这个宏的作用与 MFC 宏 VERIFY 完全相似。在应用程序的调试版本中，ISAPIVERIFY 宏计算它的参数。如果结果为 0，就打印出一条诊断信息并终止程序。如果结果为非零值，它什么也不做。

诊断信息具有如下形式：

assertion failed in file <name> in line <num>

这里的 *name* 是源文件的名字，*num* 是源文件中发生断言失败的位置的行号。

在应用程序的发行版本中，ISAPIVERIFY 计算表达式，但是并不打印或中断程序。例如，如果这个表达式是一个函数调用，那么将完成这个调用。

ISAPI 应用程序没有必要使用 MFC。如果你的应用程序没有与 MFC 连接，ISAPIVERIFY 提供与 VERIFY 相同的功能。如果你的应用程序与 MFC 连接，ISAPIVERIFY 简单地调用 MFC 的 VERIFY。

请参阅 ISAPIASSERT, ISAPITRACE, ISAPITRACE0, ISAPITRACE1, ISAPITRACE2, ISAPITRACE3

METHOD_PROLOGUE

METHOD_PROLOGUE(*theClass*, *localClass*)

参数

theClass

指定了要实现接口映射的类的名字。

localClass

指定了实现接口映射的本地类的名字。

说明

当调用一个引出接口的方法时，使用 METHOD_PROLOGUE 宏来维护正确的全局状态。

典型的情况是，CCmdTarget 派生类的对象所实现的接口的成员函数已经使用了这个宏以对 *pThis* 指针实现自动初始化。例如：

```
class CInnerUnknown : public IUnknown
    ...
    CInnerUnknown InnerUnknown;
    ...
// 内部 IUnknown 实现
    STDMETHODIMP_(ULONG) CInnerUnknown::AddRef()
    {
      METHOD_PROLOGUE(CCmdTarget, InnerUnknown)
      return pThis->InternalAddRef();
    }
```

附加的信息参见 Visual C++联机文档中的“技术注释 38”以及“Visual C++程序员指南”中的“创建新文档，窗口和视”一文中的“管理 MFC 全局模块的状态数据”一节。

ON_COMMAND

ON_COMMAND(*id, memberFxn*)

参数

id

命令ID。

memberFxn

命令被映射到的消息处理函数的名字。

说明

这个宏通常由ClassWizard自动插入消息映射，也可以手动插入。它指明了将由哪个函数来处理用户界面对象，如菜单项或工具条按钮所发出的命令消息。

当一个命令目标对象接收到了带有指定ID的Windows的WM_COMMAND消息时，ON_COMMAND将调用成员函数*memberFxn*来处理消息。

用ON_COMMAND把单个消息映射到成员函数。用ON_COMMAND_RANGE把一个范围的命令映射到一个成员函数。对于一个给定的命令ID，只能有一个消息映射入口。这就是说，不能将一个命令映射到多于一个的处理函数。相关的更多信息和例子参见“Visual C++程序员指南”中的“消息处理和映射”主

题。

示例

```
// ON_COMMAND 的例子
BEGIN_MESSAGE_MAP( CMyDoc, CDocument )
    //{{AFX_MSG_MAP( CMyDoc )
    ON_COMMAND( ID_MYCMD, OnMyCommand )
    // ... 其它入口，用于处理别的命令
    //}}AFX_MSG_MAP
END_MESSAGE_MAP( )
```

请参阅 ON_UPDATE_COMMAND_UI

ON_COMMAND_RANGE

ON_COMMAND_RANGE(*id1, id2, memberFxn*)

参数

id1

一个连续范围的命令 ID 的起始值。

id2

一个连续范围的命令ID的结束值。

memberFxn

该命令被映射到的消息处理函数的名字。

说明

使用这个宏把一个连续范围的命令ID映射到单个命令处理函数。ID的范围从*id1*开始，到*id2*结束。

用ON_COMMAND_RANGE把一个范围的命令ID映射到一个成员函数。用ON_COMMAND把单个命令ID映射到成员函数。每个给定的命令ID只能有一个消息映射入口。这就是说，不能把命令映射到多于一个的处理函数。关于映射消息范围的更多信息参见“Visual C++程序员指南”中的“消息映射范围的处理函数”。

ClassWizard不支持消息映射范围，所以你必须自己写入这个宏。确保你把它写在了消息映射的//{{AFX_MSG_MAP分界符外面。

请参阅 ON_UPDATE_COMMAND_UI_RANGE, ON_CONTROL_RANGE, ON_COMMAND

ON_CONTROL

ON_CONTROL(*wNotifyCode, id, memberFxn*)

参数

wNotifyCode

控件的通知代码。

id

命令 ID。

memberFxn

命令被映射到的消息处理函数的名字。

说明

这个宏指明哪个函数将处理自定义控件的通知消息。控件的通知消息是指控件发送给它的父窗口的消息。

对于每个要被映射到消息处理函数的控件通知消息，在消息映射中都必须有一个 ON_CONTROL 宏语句。

相关的更多信息及例子参见“Visual C++程序员指南”中的“消息处理和映射”主题。

请参阅 ON_MESSAGE, ON_REGISTERED_MESSAGE

ON_CONTROL_RANGE

ON_CONTROL_RANGE(*wNotifyCode, id1, id2, memberFxn*)

参数

wNotifyCode

你的处理函数所响应的通知代码。

id1

一个连续范围的控制 ID 的起始值。

id2

一个连续范围的控制 ID 的结束值。

memberFxn

控件被映射到的消息处理函数的名字。

说明

用这个宏把一个连续范围的控制 ID 映射到单个的 Windows 通知消息，如 BN_CLICKED，的处理函数。ID 的范围从 *id1* 开始，到 *id2* 结束。这个处理函

数是为从映射控件发出的特定通知而调用的。

ClassWizard 不支持消息映射范围，因此你必须自己写入这个宏。确保你把它写在了消息映射分界符//{{AFX_MSG_MAP 的外面。

请参阅 ON_UPDATE_COMMAND_UI_RANGE, ON_COMMAND_RANGE

ON_EVENT

ON_EVENT(*theClass, id, dispid, pfnHandler, vtsParams*)

参数

theClass

事件接收映射所属的类。

id

OLE 控件的控制 ID。

dispid

控件引发分事件的调度 ID。

pfnHandler

处理事件的成员函数的指针。这个函数具有 BOOL 类型的返回值，其参数的类型与事件的参数（参见 *vtsParams*）类型相匹配。函数应当返回

TRUE 以表明事件已被处理，否则返回 FALSE。

vtsParams

VTS_常量的序列，指定了事件参数的类型。这些值与诸如 DISP_FUNCTION 之类的调度映射入口中使用的常量相同。

说明

用 ON_EVENT 宏来为 OLE 控件引发的事件定义事件处理函数。

vtsParams 参数是用空格分隔的 VTS_常量的列表。用空格隔开的值指定了函数的参数列表。例如：

VTS_I2 VTS_BOOL

指定了一个包含短整数和后面的 BOOL 量的列表。

VTS_常量的列表参见 EVENT_CUSTOM。

请参阅 ON_EVENT_RANGE, ON_PROPNOTIFY, ON_PROPNOTIFY_RANGE

ON_EVENT_RANGE

ON_EVENT_RANGE(*theClass, idFirst, idLast, dispid, pfnHandler, vtsParams*)

参数

theClass

事件接收映射所属的类。

idFirst

范围中第一个 OLE 控件的控制 ID。

idLast

范围中最后一个 OLE 控件的控制 ID。

dispid

控件引发的事件的调度 ID。

pfnHandler

指向处理事件的成员函数指针。这个函数必须具有 BOOL 类型的返回值，第一个参数的类型为 UINT（用于控制 ID），另外的参数类型与事件的参数（参见 *vtsParams*）匹配。这个函数应当返回 TRUE 以表明该事件已被处理，否则返回 FALSE。

vtsParams

VTS_常量序列，指定了事件参数的类型。第一个常量应当是属于 VTS_I4 类型的，用于控制 ID。这些常量与诸如 DISP_FUNCTION 之类的调度映射入口中所使用的相同。

说明

使用 ON_EVENT_RANGE 宏来为一些 OLE 控件所引发的事件定义一个事件处理函数。这些控件的 ID 属于一个连续的范围。

vtsParams 参数是一个用空格分隔的 VTS_常量的列表。这些用空格隔开的值指定了函数的参数列表。例如：

VTS_I2 VTS_BOOL

指定了一个包含短整数和后面的 BOOL 量的列表。

VTS_常量的列表参见 EVENT_CUSTOM。

请参阅 ON_EVENT, ON_PROPNOTIFY, ON_PROPNOTIFY_RANGE

ON_EVENT_REFLECT

ON_EVENT_REFLECT(*theClass, dispid, pfnHandler, vtsParams*)

参数

theClass

事件接收映射所属的类。

dispid

控件引发的事件的调度 ID。

pfnHandler

指向处理事件的成员函数的指针。这个函数必须具有 BOOL 类型的返回值，第一个参数的类型为 UINT（用于控制 ID），另外的参数类型与事件的参数（参见 *vtsParams*）匹配。这个函数应当返回 TRUE 以表明该事件已被处理，否则返回 FALSE。

vtsParams

VTS_常量序列，指定了事件参数的类型。这些常量与诸如 DISP_FUNCTION 之类的调度映射入口中所使用的相同。

说明

当被用于 OLE 控件的封装类的事件接收映射时，ON_EVENT_REFLECT 宏在控件的容器处理之前接收控件引发的事件。

vtsParams 参数是一个用空格分隔的 VTS_常量的列表。这些用空格隔开的值指定了函数的参数列表。例如：

VTS_I2 VTS_BOOL
指定了一个包含短整数和后面的 BOOL 量的列表。

VTS_常量的列表参见 EVENT_CUSTOM。

请参阅 ON_EVENT, ON_PROPNOTIFY, ON_PROPNOTIFY_REFLECT

ON_MESSAGE

ON_MESSAGE(*message*, *memberFxn*)

参数

message
　　消息的 ID。

memberFxn

消息被映射到消息处理函数的名字。

说明

这个宏指明哪个函数将处理用户定义的消息。用户定义的消息通常位于 WM_USER 和 0x7FFF 之间。用户定义的消息是指不属于标准的 Windows WM_MESSAGE 消息的任何消息。对于每个需要被映射到消息处理函数的用户自定义消息，在你的消息映射中都必须有且只能有一个 ON_MESSAGE 宏语句。

有关的更多信息和例子参见“Visual C++程序员指南”中的“消息处理和映射”主题。

示例

```
// ON_MESSAGE 的例子
#define WM_MYMESSAGE (WM_USER + 1)
BEGIN_MESSAGE_MAP( CMyWnd, CMyParentWndClass )
    //{{AFX_MSG_MAP( CMyWnd
    ON_MESSAGE( WM_MYMESSAGE, OnMyMessage )
    // ... 可能的其它入口，用于处理其它消息
    //}}AFX_MSG_MAP
END_MESSAGE_MAP( )
```

请参阅 ON_UPDATE_COMMAND_UI, ON_CONTROL,

ON_REGISTERED_MESSAGE, ON_COMMAND, User-Defined Handlers

ON_OLECMD

ON_OLECMD(*pguid, olecmdid, id*)

参数

pguid

命令岁数的命令组的表示符。对于标准组使用 NULL。

olecmdid

OLE 命令的标识符。

id

发出命令的资源或对象的菜单 ID，工具条 ID，按钮 ID 或其它 ID。

说明

这个宏通过命令调度接口 IOleCommandTarget 转发命令。IOleCommandTarget 允许容器接收 DocObject 的用户所产生的命令，同时允许容器将相同的命令（例如 File 菜单中的 New，Open，SaveAs 以及 Print）发送给 DocObject。

IOleCommandTarget 比 OLE 自动化的 IDispatch 要简单。IOleCommandTarget 完全依赖于一个标准命令集，它们很少带参数，也不涉及类型信息（因而命令参数的类型安全特性也减小了）。如果你不需要调度带参数的命令，使用 COleServerDoc::OnExecOleCmd。

IOleCommandTarget 的标准菜单命令已经由 MFC 用下列宏实现了：

ON_OLECMD_CLEARSELECTION( )
发出 Edit Clear 命令，实现如下：
ON_OLECMD(NULL, OLECMDID_CLEARSELECTION, ID_EDIT_CLEAR)

ON_OLECMD_COPY( )
发出 Edit Copy 命令，实现如下：
ON_OLECMD(NULL, OLECMDID_COPY, ID_EDIT_COPY)

ON_OLECMD_CUT( )
发出 Edit Cut 命令，实现如下：
ON_OLECMD(NULL, OLECMDID_CUT, ID_EDIT_CUT)

ON_OLECMD_NEW( )
发出 File New 命令，实现如下：
ON_OLECMD(NULL, OLECMDID_NEW, ID_FILE_NEW)

ON_OLECMD_OPEN( )
发出 File Open 命令，实现如下：
ON_OLECMD(NULL, OLECMDID_OPEN, ID_FILE_OPEN)

ON_OLECMD_PAGESETUP( )

发出 File Page Setup **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_PAGESETUP, ID_FILE_PAGE_SETUP)

ON_OLECMD_PASTE( )

发出 Edit Paste **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_PASTE, ID_EDIT_PASTE)

ON_OLECMD_PASTESPECIAL( )

发出 Edit Paste Special **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_PASTESPECIAL, ID_EDIT_PASTE_SPECIAL)

ON_OLECMD_PRINT( )

发出 File Print **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_PRINT, ID_FILE_PRINT)

ON_OLECMD_PRINTPREVIEW( )

发出 File Print Preview **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_PRINTPREVIEW, ID_FILE_PRINT_PREVIEW)

ON_OLECMD_REDO( )

发出 Edit Redo **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_REDO, ID_EDIT_REDO)

ON_OLECMD_SAVE( )

发出 File Save **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_SAVE, ID_FILE_SAVE)

ON_OLECMD_SAVE_AS( )

发出 File Save As **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_SAVEAS, ID_FILE_SAVE_AS)

ON_OLECMD_SAVE_COPY_AS( )

发出 File Save Copy As **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_SAVECOPYAS, ID_FILE_SAVE_COPY_AS)

ON_OLECMD_SELECTALL( )

发出 Edit Select All **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_SELECTALL, ID_EDIT_SELECT_ALL)

ON_OLECMD_UNDO( )

发出 Edit Undo **命令，实现如下：**

ON_OLECMD(NULL, OLECMDID_UNDO, ID_EDIT_UNDO)

请参阅 COleCmdUI, COleServerDoc::OnExecOleCmd

ON_OLEVERB

ON_OLEVERB(*idsVerbName*, *memberFxn*)

参数

idsVerbName
动词名字的字符串资源 ID。

memberFxn
当动词被激活时框架调用的函数。

说明

这个宏定义了一个消息映射入口，将一个自定义的动词映射到控件的指定成员函数。

可用资源编辑器来生成自定义动词名字，加入字符串表。

memberFxn 的函数原型如下：

BOOL memberFxn(LPMSG *lpMsg*, HWND *hWndParent*, LPCRECT *lpRect*);

lpMsg，*hWndParent* 和 *lpRect* 参数的值时从 IOleObject::DoVerb 成员函数的对应参数中取得的。

请参阅 ON_STDOLEVERB

ON_PARSE_COMMAND

ON_PARSE_COMMAND(*FnName, mapClass, Args*)

参数

FnName

　　成员函数的名字。也是命令的名字。

mapClass

　　要把函数映射到的类的名字。

Args

　　映射到 *FnName* 的参数。符号的列表参见说明部分。

说明

ON_PARSE_COMMAND 宏被用于解析映射，用来定义从客户发送到 CHttpServer 对象的命令。

FnName 表示的成员函数必须以一个指向 CHttpServerContext 对象的指针作为它的第一个参数。*FnName* 属于 LPSTR 类型，用解析映射中的符号 ITS_LPSTR 来标识，这意味着，*FnName* 指向一个包含了 *mapClass* 类的成员函数名的字符串。

参数 Args 可以取下列值之一：

符号	类型或注释
ITS_EMPTY	Args 不能为空白。如果你没有参数，则使用 ITS_EMPTY
ITS_PSTR	字符串指针
ITS_RAW	精确的行数据发送到 ISAPI 扩展，不要使用具有任何其它参数类型的 ITS_RAW；若使用了，则导致 ASSERT。参见"Remarks"中的示例
ITS_I2	一个 short 值
ITS_I4	一个 long 值
ITS_R4	一个 float 值
ITS_R8	一个 double 值

示例

```
//   The following example Illustrates extracting
//   a string and a short sent to the server:
BEGIN_PARSE_MAP(CDerivedClass, CHttpServer)
    DEFAULT_PARSE_COMMAND(Myfunc, CDerivedClass)
    ON_PARSE_COMMAND(Myfunc, CDerivedClass, ITS_PSTR      ITS_I2)
    ON_PARSE_COMMAND_PARAMS("string integer=42")
    ON_PARSE_COMMAND(Myfunc2, CDerivedClass, ITS_PSTR        ITS_I2
ITS_PSTR)
```

```
    ON_PARSE_COMMAND_PARAMS("string integer string2='Default value'")
END_PARSE_MAP(CDerivedClass)
```

注意　如果你在可选的 ITS_PSTR 的缺省值中加入空格，使用单引号。

```
void Myfunc(CHttpServerContext* pCtxt, LPTSTR pszName, int nNumber);
void Myfunc2(CHttpServerContext* pCtxt, LPTSTR pszName, int nNumber, LPTSTR pszTitle);
// 下面的例子演示了提取发送给服务器的原始数据
BEGIN_PARSE_MAP(CDerivedClass, CHttpServer)
    DEFAULT_PARSE_COMMAND(MyFunc, CDerivedClass)
    ON_PARSE_COMMAND(Myfunc, CDerivedClass, ITS_RAW)
END_PARSE_MAP(CDerivedClass)
```

函数原型如下：

```
void CDerivedClass::Myfunc(CHttpServerContext* pCtxt, void* pVoid,
DWORD dwBytes);
```

在第二个例子中，pVoid **指针指向发送给你的扩展部分的数据。**dwBytes **参数是** pVoid **所指向的字节数目。如果** dwBytes **为零，**pVoid **可能不指向任何内容。**

> **注意　解析映射命令的处理函数必须以指向 CHttpServerContext 对象的指针为第一个参数，参数必须按照与 ON_PARSE_COMMAND 中相同的顺序定义。**

请参阅 BEGIN_PARSE_MAP, END_PARSE_MAP, ON_PARSE_COMMAND_PARAMS, DEFAULT_PARSE_COMMAND, CHttpServer

ON_PARSE_COMMAND_PARAMS

ON_PARSE_COMMAND_PARAMS(Params)

参数

Params

参数，映射到 *Args* 参数，并与 *FnName* 所标识的函数相关，在 ON_PARSE_COMMAND 宏中，位于 ON_PARSE_COMMAND_PARAMS 之前。

说明

ON_PARSE_COMMAND_PARAMS 宏标识并指定了参数的缺省值，该参数与一个函数相关，这个函数被映射到一个由客户发往 CHttpSever 对象的命令。ON_PARSE_COMM- AND_PARAMS 宏必须紧跟在与之相关的 ON_PARSE_COMMAND 宏之后。

如果参数被命名，客户在请求的时候必须提供参数的名字。例如，如果你的参数如下：

ON_PARSE_COMMAND_PARAMS("string int=42")
那么客户必须提供参数字符串，否则查询就会失败。

如果参数是可选的，客户不必提供它，解析映射将会提供缺省值。例如，如果你的参数如下：

ON_PARSE_COMMAND_PARAMS("string=default int=42")

那么客户的查询中不必定义任何参数，缺省的参数字符串是一个空字符串。

解析映射的例子参见 ON_PARSE_COMMAND。

请参阅 BEGIN_PARSE_MAP, END_PARSE_MAP, ON_PARSE_COMMAND, DEFAULT_PARSE_COMMAND, CHttpServer

ON_PROPNOTIFY

ON_PROPNOTIFY(*theClass, id, dispid, pfnRequest, pfnChanged*)

参数

theClass

事件接收映射所属的类。

id

OLE 控件的控制 ID。

dispid

与通知有关的属性的调度 ID。

pfnRequest

指向一个成员函数的指针，该函数处理属性的 OnRequestEdit 通知。这个函数必须具有 BOOL 类型的返回值和 BOOL*类型的参数。它应当将参数设为 TRUE 以允许改变属性，或者设为 FALSE 以禁止改变属性。这个函数应当返回 TRUE 以表明它已经处理了通知，否则返回 FALSE。

pfnChanged

指向一个成员函数的指针，该函数处理属性的 OnChanged 通知。函数应当具有 BOOL 类型的返回值和一个 UINT 类型的参数。这个函数必须返回 TRUE 以表明它已经处理了通知，否则返回 FALSE。

说明

用 ON_PROPNOTIFY 宏来定义一个事件接收映射入口以处理 OLE 控件发出的属性通知。

vtsParams 参数是一个用空格分隔的 VTS_常量的列表。这些用空格隔开的值指

定了函数的参数列表。例如：

VTS_I2 VTS_BOOL

指定了包含短整数和后面的 BOOL 量的列表。

VTS_常量的列表参见 EVENT_CUSTOM。

请参阅 ON_EVENT_RANGE, ON_PROPNOTIFY_RANGE

ON_PROPNOTIFY_RANGE

ON_PROPNOTIFY_RANGE(*theClass*, *idFirst*, *idLast*, *dispid*, *pfnRequest*, *pfnChanged*)

参数

theClass

事件接收映射所属的类。

idFirst

范围中第一个 OLE 控件的控制 ID。

idLast

范围中最后一个 OLE 控件的控制 ID。

dispid

与通知有关的属性的调度 ID。

pfnRequest

指向一个成员函数的指针，该函数处理属性的 OnRequestEdit 通知。这个函数必须具有 BOOL 类型的返回值和 UINT 以及 BOOL*类型的参数。它应当将参数设为 TRUE 以允许改变属性，或者设为 FALSE 以禁止改变属性。这个函数应当返回 TRUE 以表明它已经处理了通知，否则返回 FALSE。

pfnChanged

指向一个成员函数的指针，该函数处理属性的 OnChanged 通知。这个函数必须具有 BOOL 型返回值以及一个 UINT 参数。该函数必须返回 TRUE 以表明通知已经被处理，否则返回 FALSE。

说明

使用 ON_PROPNOTIFY_RANGE 宏来定义一个事件接收映射，用以处理 OLE 控件发出的属性通知，这些控件的 ID 属于一个连续的范围。

请参阅 ON_EVENT_RANGE, ON_PROPNOTIFY, ON_EVENT

ON_PROPNOTIFY_REFLECT

ON_PROPNOTIFY_REFLECT(*theClass, dispid, pfnRequest, pfnChanged*)

参数

theClass

事件接收映射所属的类。

dispid

与通知有关的属性的调度 ID**。**

pfnRequest

指向一个成员函数的指针，该函数处理属性的 OnRequestEdit **通知。这个函数必须具有** BOOL **类型的返回值和** BOOL* **型参数。这个函数应当将参数设为** TRUE **以允许改变属性，或者设为** FALSE **以禁止改变属性。函数应该返回** TRUE **以表明通知已经被处理，否则就返回** FALSE**。**

pfnChanged

指向一个成员函数的指针，该函数处理属性的 OnChanged **通知。这个函数必须具有** BOOL **类型的返回值，不能有参数。函数应该返回** TRUE **以表明通知已经被处理，否则就返回** FALSE**。**

说明

当 ON_PROPNOTIFY_REFLECT 宏被用于 OLE 控件的封装类的事件接收映射时，它在控件发出的属性通知被控件容器处理之前接收它们。

请参阅 ON_EVENT_REFLECT, ON_PROPNOTIFY

ON_REGISTERED_MESSAGE

ON_REGISTERED_MESSAGE(nMessageVariable, memberFxn)

参数

nMessageVariable

注册的窗口消息 ID 变量。

memberFxn

消息被映射到的消息处理函数的名字。

说明

Windows 的 RegisterWindowMessage 函数可被用来定义一个新的窗口消息，这个消息在系统内应当是唯一的。这个消息指明由哪个函数来处理注册的消息。

有关的更多信息和例子参见“Visual C++程序员指南”中的“消息处理和映射”主题。

示例

```
// ON_REGISTERED_MESSAGE 的例子
const UINT    wm_Find = RegisterWindowMessage( FINDMSGSTRING )
BEGIN_MESSAGE_MAP( CMyWnd, CMyParentWndClass )
    //{{AFX_MSG_MAP( CMyWnd )
    ON_REGISTERED_MESSAGE( wm_Find, OnFind )
    // ... 可能的入口，用于处理其它消息。
    //}}AFX_MSG_MAP
END_MESSAGE_MAP( )
```

请参阅 ON_MESSAGE, ON_UPDATE_COMMAND_UI, ON_CONTROL, ON_COMMAND, ::RegisterWindowMessage, User-Defined Handlers

ON_REGISTERED_THREAD_MESSAGE

ON_REGISTERED_THREAD_MESSAGE(*nMessageVariable*, *memberFxn*)

参数

nMessageVariable

注册的窗口消息 ID 变量。

memberFxn

消息被映射到的 CWinThread 消息处理函数的名字。

说明

这个宏指明由哪个函数来处理 Windows 的 RegisterWindowMessage 函数注册的消息。RegisterWindowMessage 函数被用于定义一个新的窗口消息，在整个系统中，该消息是唯一的。当使用 CWinThread 类的时候，必须用 ON_REGISTERED_THREAD_MESSAGE 来代替 ON_REGISTERED_MESSAGE。

请参阅 ON_REGISTERED_MESSAGE, ON_THREAD_MESSAGE, ::RegisterWindowMessage, CWinThread

ON_STDOLEVERB

ON_STDOLEVERB(*iVerb, memberFxn*)

参数

iVerb

被覆盖的动词的标准动词索引。

memberFxn

当动词被激活时由框架调用的函数。

说明

利用这个宏来改变标准动词的缺省动作。

标准动词索引的形式如 OLEIVERB_，后面跟一个动作。例如，OLEIVERB_SHOW，OLEIVERB_HIDE 和 OLEIVERB_UIACTIVATE 等标准动词的例子。

对可以用作 *memberFxn* 参数的函数原型的描述参见 ON_OLEVERB。

请参阅 ON_OLEVERB

ON_THREAD_MESSAGE

ON_THREAD_MESSAGE(*message*, *memberFxn*)

参数

message

消息的 ID。

memberFxn

消息被映射到的 CWinThread 消息处理函数的名字。

说明

这个宏指明由哪个函数来处理用户定义的消息。当使用 CWinThread 类的时候，必须用 ON_THREAD_MESSAGE 来代替 ON_MESSAGE。用户定义的消息是指那些不属于标准的 Windows WM_MESSAGE 的消息。对于每一个要映射到消息处理函数的用户定义函数，在消息映射中必须有且只能有一个 ON_THREAD_MESSAGE 宏语句。

请参阅 ON_MESSAGE, ON_REGISTERED_THREAD_MESSAGE, CWinThread

ON_UPDATE_COMMAND_UI

ON_UPDATE_COMMAND_UI(*id*, *memberFxn*)

参数

id

消息的 ID。

memberFxn

消息被映射到的消息处理函数的名字。

说明

这个宏通常由 ClassWizard 插入消息映射，用于指明由哪个函数处理用户界面的更新消息。

对于每个要映射到消息处理函数的用户界面更新命令，在消息映射中必须有且只能有一个 ON_UPDATE_COMMAND_UI 宏语句。

相关的信息和例子参见“Visual C++程序员指南”中的“消息处理和映射”主题。

请参阅 ON_MESSAGE, ON_REGISTERED_MESSAGE, ON_CONTROL, ON_COMMAND, CCmdUI

ON_UPDATE_COMMAND_UI_RANGE

ON_UPDATE_COMMAND_UI_RANGE(*id1*, *id2*, *memberFxn*)

参数

id1

一个连续的命令ID范围的起始ID。

id2

一个连续的命令ID范围的结束ID。

memberFxn

命令被映射到的更新消息处理函数的名字。

说明

使用这个宏将一个连续范围的命令ID映射到一个更新消息处理函数。更新消息处理函数更新与命令相关的菜单项和工具条按钮的状态。ID的范围从*id1*开始，到*id2*结束。

ClassWizard不支持消息映射范围，因此你必须自己加入这个宏。确保你把它写在了消息映射的分界符//{{AFX_MSG_MAP的外面。

请参阅 ON_COMMAND_RANGE, ON_CONTROL_RANGE

PROPPAGEID

PROPPAGEID(*clsid*)

参数

clsid

属性页的唯一的类 ID。

说明

使用这个宏加入一个 OLE 控件使用的属性页。所有的 PROPPAGEID 宏都必须放在你的控件的实现文件中，位于 BEGIN_PROPPAGEIDS 和 END_PROPPAGEIDS 宏之间。

请参阅 BEGIN_PROPPAGEIDS, END_PROPPAGEIDS

PX_Blob

BOOL PX_Blob(CPropExchange* *pPX*, LPCTSTR *pszPropName*, HGLOBAL& *hBlob*, HGLOBAL *hBlobDefault* = NULL);

返回值

如果成功地交换了数据则为非零值，否则为 0。

参数

pPX

指向CPropExchange对象的指针（通常用作传递给DoPropExchange的参数）。

pszPropName

要交换的属性的名字。

hBlob

对存储属性的变量（通常是你的类的成员变量）的引用。

hBlobDefault

属性的缺省值。

说明

在你的控件的DoPropExchange成员函数内调用这个函数，用以串行化或初始化一个存储二进制大对象（BLOB）数据的属性。属性的值将从*hBlob*引用的变量中读写。这个变量在最初调用PX_Blob之前必须被初始化为NULL（通常这可以在控件的构造函数中完成）。如果指定了*hBlobDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的初始化或串行化过程失败了，就会使用这个缺省值。

句柄*hBlob*和*hBlobDefault*指向一个内存块，其中包含如下内容：

- 包含了后面的二进制数据的长度（以字节为单位）的 DWORD 值，后面是：
- 包含了实际的二进制数据的内存块。

注意当载入 BLOB 类型的属性时，PX_Blob 会使用 Windows 的 API 函数 GlobalAlloc 来分配内存。你应该负责释放内存。因此，你的控件的析构函数必须对 BLOB 类型的属性句柄调用 GlobalFree 以释放为控件分配的内存。

请参阅　COleControl::DoPropExchange

PX_Bool

BOOL PX_Bool(CPropExchange* *pPX*, LPCTSTR *pszPropName*, BOOL& *bValue*);
BOOL PX_Bool(CPropExchange* *pPX*, LPCTSTR *pszPropName*, BOOL& *bValue*, BOOL *bDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName

将要交换的属性的名字。

bValue

对保存属性的变量的引用（通常是你的类的成员变量）。

bDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 BOOL 类型的属性。属性的值将从 *bValue* 引用的变量中读写。如果指定了 *bDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange

PX_Color

BOOL PX_Color(CPropExchange* *pPX*, LPCTSTR *pszPropName*, OLE_COLOR&

clrValue);
BOOL PX_Color(CPropExchange* *pPX*, LPCTSTR *pszPropName*, OLE_COLOR& *clrValue*, OLE_COLOR *clrDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX
: **指向** CPropExchange **对象的指针（通常作为参数传递给** DoPropExchange**）。**

pszPropName
: **将要交换的属性的名字。**

clrValue
: **对保存属性的变量的引用（通常是你的类的成员变量）。**

clrDefault
: **属性的缺省值。**

说明

在你的控件的 DoPropExchange **成员函数内调用这个函数，用以串行化或初始化一个** OLE_COLOR **类型的属性。属性的值将从** *clrValue* **引用的变量中读写。如果指定了** *clrDefault*，**它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。**

请参阅 COleControl::DoPropExchange

PX_Currency

BOOL PX_Currency(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CY& *cyValue*);
BOOL PX_Currency(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CY& *cyValue*, CY *cyDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName

将要交换的属性的名字。

cyValue

对保存属性的变量的引用（通常是你的类的成员变量）。

cyDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 currency 类型的属性。属性的值将从 *cyValue* 引用的变量中读写。如果指定了 *cyDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange

PX_DataPath

BOOL PX_DataPath(CPropExchange* *pPX*, LPCTSTR *pszPropName*,

CDataPathProperty& *dataPathProperty*);
BOOL PX_DataPath(CPropExchange* *pPXe*, CDataPathProperty& *dataPathProperty*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX
　　指向 CPropExchange **对象的指针（通常作为参数传递给** DoPropExchange**）。**

pszPropName
　　将要交换的属性的名字。

dataPathProperty
　　对保存属性的变量的引用（通常是你的类的成员变量）。

说明

在你的控件的 DoPropExchange **成员函数内部调用这个函数以串行化或初始化一个** CDataPathProperty **类型的日期路径属性。日期路径属性实现了异步的控件**

属性。属性的值将从 *dataPathProperty* 所引用的变量中读写。

请参阅 COleControl::DoPropExchange, CDataPathProperty

PX_Double

BOOL PX_Double(CPropExchange* *pPX*, LPCTSTR *pszPropName*, double& *doubleValue*);
BOOL PX_Double(CPropExchange* *pPX*, LPCTSTR *pszPropName*, double& *doubleValue*, double *doubleDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX
指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName
将要交换的属性的名字。

doubleValue

对保存属性的变量的引用（通常是你的类的成员变量）。

doubleDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 double 类型的属性。属性的值将从 *doubleValue* 引用的变量中读写。如果指定了 *doubleDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange, PX_Float, PX_Short

PX_Float

BOOL PX_Float(CPropExchange* *pPX*, LPCTSTR *pszPropName*, float& *floatValue*);

BOOL PX_Float(CPropExchange* *pPX*, LPCTSTR *pszPropName*, float& *floatValue*, float *floatDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName

将要交换的属性的名字。

floatValue

对保存属性的变量的引用（通常是你的类的成员变量）。

floatDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 float 类型的属性。属性的值将从 *floatValue* 引用的变量中读写。如果指定了 *floatDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串

行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange, PX_Double, PX_String

PX_Font

BOOL PX_Font(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CFontHolder& *font*, const FONTDESC FAR* *pFontDesc* = NULL, LPFONTDISP *pFontDispAmbient* = NULL);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName

要交换的属性的名字。

font

对包含了字体属性的 CFontHolder 对象的引用。

pFontDesc

指向 FONTDESC 结构的指针，该结构包含了在 *pFontDispAmbient* 为 NULL 时用来初始化字体属性的缺省状态的值。

pFontDispAmbient

指向字体的 IFontDisp 接口的指针，用于初始化字体属性的缺省状态。

说明

在控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化字体类型的属性。属性的值将从字体（一个 CFontHolder 引用）读写。如果指定了 *pFontDesc* 和 *pFontDispAmbient*，它们将被用于在必要时初始化属性的缺省值。如果由于某种原因控件的串行化过程失败了，就会使用这个值。通常，你在 *pFontDesc* 中传递 NULL，在 *pFontDispAmbient* 中传递 COleControl::AmbientFont 返回的值。注意 COleControl:: AmbientFont 返回的字体对象必须用 IFontDisp::Release 来释放。

请参阅 COleControl::DoPropExchange, COleControl::AmbientFont

PX_IUnknown

BOOL PX_IUnknown(CPropExchange* *pPX*, LPCTSTR *pszPropName*,

LPUNKNOWN& *pUnk*, REFIID *iid*, LPUNKNOWN *pUnkDefault* = NULL);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange **对象的指针（通常作为参数传递给** DoPropExchange**）。**

pszPropName

将要交换的属性的名字。

pUnk

对包含了对象的接口变量的引用，该变量代表属性的值。

iid

接口 ID **指明控件将使用属性对象的哪个接口。**

pUnkDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange **成员函数内调用这个函数，用以串行化或初始化一个具有** IUnknown **接口的对象所代表的属性。属性的值将从** *pUnk* **引用的变量中读写。如果指定了** *pUnkDefault*，**它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。**

请参阅 COleControl::DoPropExchange

PX_Long

BOOL PX_Long(CPropExchange* *pPX*, LPCTSTR *pszPropName*, long& *lValue*);
BOOL PX_Long(CPropExchange* *pPX*, LPCTSTR *pszPropName*, long& *lValue*, long *lDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange **对象的指针（通常作为参数传递给**

DoPropExchange）。

pszPropName

将要交换的属性的名字。

lValue

对保存属性的变量的引用（通常是你的类的成员变量）。

lDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 long 类型的属性。属性的值将从 *lValue* 引用的变量中读写。如果指定了 *lDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange

PX_Picture

BOOL PX_Picture(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CPictureHolder& *pict*);

BOOL PX_Picture(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CPictureHolder& *pict*, CPictureHolder& *pictDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX
　　指向 CPropExchange **对象的指针（通常作为参数传递给** DoPropExchange**）。**

pszPropName
　　将要交换的属性的名字。

pict
　　对保存属性的 CPictureHolder **对象的引用（通常是你的类的成员变量）。**

pictDefault
　　属性的缺省值。

说明

在你的控件的 DoPropExchange **成员函数内调用这个函数，用以串行化或初始**

化一个图形属性。属性的值将从 *pict* 引用的变量中读写。如果指定了 *pictDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange

PX_Short

BOOL PX_Short(CPropExchange* *pPX*, LPCTSTR *pszPropName*, short& *sValue*);
BOOL PX_Short(CPropExchange* *pPX*, LPCTSTR *pszPropName*, short& *sValue*, short *sDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName

将要交换的属性的名字。

sValue

对保存属性的变量的引用（通常是你的类的成员变量）。

sDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 short 类型的属性。属性的值将从 *sValue* 引用的变量中读写。如果指定了 *sDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange

PX_String

BOOL PX_String(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CString& *strValue*);

BOOL PX_String(CPropExchange* *pPX*, LPCTSTR *pszPropName*, CString& *strValue*, CString *strDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX
　　指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName
　　将要交换的属性的名字。

strValue
　　对保存属性的变量的引用（通常是你的类的成员变量）。

strDefault
　　属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个字符串属性。属性的值将从 *strValue* 引用的变量中读写。如果指定了 *strDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过

程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange, CString

PX_ULong

BOOL PX_ULong(CPropExchange* *pPX*, LPCTSTR *pszPropName*, ULONG& *ulValue*);
BOOL PX_ULong(CPropExchange* *pPX*, LPCTSTR *pszPropName*, ULONG& *ulValue*, long *ulDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回0。

参数

pPX
: 指向CPropExchange对象的指针（通常作为参数传递给DoPropExchange）。

pszPropName
: 将要交换的属性的名字。

ulValue

对保存属性的变量的引用（通常是你的类的成员变量）。

ulDefault

属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 ULONG 类型的属性。属性的值将从 *ulValue* 引用的变量中读写。如果指定了 *ulDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件的串行化过程失败了，就会使用这个缺省值。

请参阅 COleControl::DoPropExchange

PX_UShort

BOOL PX_UShort(CPropExchange* *pPX*, LPCTSTR *pszPropName*, USHORT& *usValue*);
BOOL PX_UShort(CPropExchange* *pPX*, LPCTSTR *pszPropName*, USHORT& *usValue*, USHORT *usDefault*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX
　　指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

pszPropName
　　将要交换的属性的名字。

usValue
　　对保存属性的变量的引用（通常是你的类的成员变量）。

usDefault
　　属性的缺省值。

说明

在你的控件的 DoPropExchange 成员函数内调用这个函数，用以串行化或初始化一个 unsigned short 类型的属性。属性的值将从 *usValue* 引用的变量中读写。如果指定了 *usDefault*，它将被用作该属性的缺省值。如果由于某种原因，控件

的串行化过程失败了，就会使用这个缺省值。

请参阅　COleControl::DoPropExchange

PX_VBXFontConvert

BOOL PX_VBXFontConvert(CPropExchange* *pPX*, CFontHolder& *font*);

返回值

如果成功地交换了数据，则返回非零值；否则返回 0。

参数

pPX

　　指向 CPropExchange 对象的指针（通常作为参数传递给 DoPropExchange）。

font

　　OLE 控件的字体属性，包含了转换的 VBX 控件与字体有关的属性。

说明

在控件的 DoPropExchange 成员函数内部调用这个函数，转换 VBX 控件与字体

有关的属性，用以初始化字体属性。

这个函数只能被那些设计来直接代替VBX控件的OLE控件使用。当Visual Basic开发环境转换一个包含VBX控件的窗体，使用相应的OLE控件时，它将会调用控件的IDataObject::SetData函数，传递一个包含了VBX控件的属性数据的属性集。这个操作随后使控件的DoPropExchange函数被激活。DoPropExchange能够调用PX_VBXFontConvert以把VBX控件与字体有关的属性（例如，“字体名”、“字体大小”等）转换为OLE控件中字体属性的相应内容。

只有当控件是从VBX窗体应用程序中转换而来时才能调用PX_VBXFontConvert。例如：

```
void CSampleCtrl::DoPropExchange(CPropExchange* pPX)
{
	ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
	COleControl::DoPropExchange(pPX);
	if (IsConvertingVBX())
		PX_VBXFontConvert(pPX, InternalGetFont());
}
```

请参阅 COleControl::DoPropExchange, COleControl::AmbientFont, PX_Font

RFX_Binary

void RFX_Binary(CFieldExchange* *pFX*, const char* *szName*, CByteArray& *value*, int *nMaxLength* = 255);

参数

pFX

指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象所指定操作的更多信息参见“** Visual C++**程序员指南”中的文章“记录字段交换：**RFX **如何工作”。**

szName

数据列的名字。

value

指定数据成员所保存的值----**要被交换的值。对于从记录集到数据源的数据传送，**CByteArray **类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。**

nMaxLength

要被交换的字符串或数组对象的最大长度。*nMaxLength* **的缺省值为** 255**。合法的值为** 1 **到** INT_MAX**。框架将分配这么多内存。为了获得最好的**

性能，应该传递一个足够大的值，使之能够容纳你要求的最大数据项。

说明

RFX_Binary 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_BINARY，SQL_VARBINARY 或 SQL_LONGVARBINARY 型数据列之间交换字节数组数据。数据源中这些类型的数据被映射到记录集中的 CByteArray 类型。

示例

参见“RFX_Text”

请参阅 RFX_Text, RFX_Bool, RFX_Long, RFX_Int, RFX_Single, RFX_Double, RFX_Date, RFX_Byte, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Binary_Bulk

void RFX_Binary_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, BYTE** *prgByteVals*, long** *prgLengths*, int *nMaxLength*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

prgByteVals

指向 BYTE 值的数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgByteVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

nMaxLength

prgByteVals 指向的数组中保存的指定最大允许长度。为了确保数据不被截断，应该传递一个足够大的值，使之能够容纳你要求的最大数据项。

说明

RFX_Binary_Bulk 函数将 ODBC 数据源的数据列中的多行字节数据传送给 CRecordset 的派生对象中的相应数组。数据源的列可以属于以下类型：SQL_BINARY，SQL_VARBINARY 或 SQL_LONGVARBINARY。记录集必须定义一个 BYTE 指针型的字段数据成员。

如果你把 *prgByteVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

> **注意** 成组记录字段交换仅把数据从数据源传送到记录集对象。要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见 RFX_Text_Bulk.

请参阅 RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Bool

void RFX_Bool(CFieldExchange* *pFX*, const char* *szName*, BOOL& *value*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员所保存的值----要被交换的值。对于从记录集到数据源的数据传送，BOOL 类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_BOOL 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_BIT 型数据列之间交换布尔型数据。

示例

参见 RFX_Text.

请参阅 RFX_Text, RFX_Long, RFX_Int, RFX_Single, RFX_Double, RFX_Date, RFX_Byte, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Bool_Bulk

void RFX_Bool_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, BOOL** *prgBoolVals,* long** *prgLengths*);

参数

pFX

  指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象所指定的操作的更多信息参见“** Visual C++**程序员指南”中的文章“记录字段交换：**RFX **如何工作”。**

szName

  数据列的名字。

prgBoolVals

指向 BOOL 值数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgBoolVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

说明

RFX_Bool_Bulk 函数将 ODBC 数据源的数据列中的多行布尔数据传送给 CRecordset 派生对象中的相应数组。数据源的列必须是 SQL_BIT 类型的。记录集必须定义一个 BOOL 指针型的字段数据成员。

如果你把 *prgBoolVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

注意　成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见 RFX_Text_Bulk.

请参阅 RFX_Binary_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Byte

void RFX_Byte(CFieldExchange* *pFX*, const char* *szName*, BYTE& *value*);

参数

pFX

指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象所指定的操作的更多信息参见“**Visual C++**程序员指南”中的文章“记录字段交换：**RFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员所保存的值----要被交换的值。对于从记录集到数据源的数据传送，BYTE 类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_Byte 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_TINYINT 型数据列之间交换单字节数据。

示例

参见 RFX_Text.

请参阅 RFX_Text, RFX_Bool, RFX_Long,RFX_Int, RFX_Single, RFX_Double, RFX_Date, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Byte_Bulk

void RFX_Byte_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, BYTE** *prgByteVals*, long** *prgLengths*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象所指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

prgByteVals

指向 BYTE 值数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgByteVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

说明

RFX_Byte_Bulk 函数将 ODBC 数据源的数据列中的多行布尔数据传送给

CRecordset 的派生对象中的相应数组。数据源的列必须是 SQL_TINYINT 类型的。记录集必须定义一个 BYTE 指针型的字段数据成员。

如果你把 *prgByteVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

> **注意** 成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见 RFX_Text_Bulk.

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Date

void RFX_Date(CFieldExchange* *pFX*, const char* *szName*, CTime& *value*);

void RFX_Date(CFieldExchange* *pFX*, const char* *szName*, TIMESTAMP_STRUCT& *value*);

参数

pFX

指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象能指定的操作的更多信息参见“**Visual C++**程序员指南”中的文章“记录字段交换：**RFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员所保存的值----**要被交换的值。在函数的两个版本中，**value **的数据类型是不一样的。函数的第一个版本使用对** CTime **对象的引用。对于从记录集到数据源的数据传送，该值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。函数的第二个版本使用对** TIMESTAMP_ STRUCT **结构的引用。你必须在调用前自己设置这个结构。对话框数据交换（**DDX**）和** ClassWizard **都不支持这个版本。在你的字段映射中，将你对** RFX_Date **的第二个版本的调用写在** ClassWizard **的注释分界符之外。**

说明

RFX_Date 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_DATE，SQL_TIME 或 SQL_TIMESTAMP 型数据列之间交换 CTime 或 TIMESTAMP_STRUCT 数据。

这个函数对应 CTime 的版本会带来一些中间处理的负担，范围也有限制。如果你认为这些因素很重要，那就使用第二个版本的函数。不过要注意它缺乏 ClassWizard 和 DDX 支持，并且需要你自己设置结构体。

示例

参见“RFX_Text”

请参阅 RFX_Text, RFX_Bool, RFX_Long, RFX_Int, RFX_Single, RFX_Double, RFX_Byte, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Date_Bulk

void RFX_Date_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, TIMESTAMP_STRUCT** *prgTSVals*, long** *prgLengths*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

prgTSVals

指向 TIMESTAMP_STRUCT 值数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。有关 TIMESTAMP_STRUCT 数据类型的更多信息参见“ODBC SDK 程序员参考”中的文章“记录字段交换：RFX 如何工作”。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgBoolVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

说明

RFX_Date_Bulk 函数将 ODBC 数据源的数据列中的多行 TIMESTAMP_STRUCT 数据传送给 CRecordset 的派生对象中的相应数组。数据源的列可以是以下类型：SQL_DATE，SQL_TIME 或 SQL_TIMESTAMP。记录集必须定义一个 TIMESTAMP_STRUCT 指针型的字段数据成员。

如果你把 *prgTSVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

> **注意** 成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见 RFX_Text_Bulk.

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Double

void RFX_Double(CFieldExchange* *pFX*, const char* *szName*, double& *value*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员所保存的值——要被交换的值。对于从记录集到数据源的数据传送，double 类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_Double 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_DOUBLE 型数据列之间交换 double float 数据。

示例

参见 RFX_Text.

请参阅 RFX_Text，RFX_Bool，RFX_Long，RFX_Int，RFX_Single，RFX_Date，RFX_Byte,RFX_Binary，RFX_LongBinary，CFieldExchange::SetFieldType

RFX_Double_Bulk

void RFX_Double_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, double** *prgDblVals*, long** *prgLengths*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX如何工作”。

szName

数据列的名字。

prgDblVals

指向 double 值数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgDblVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

说明

RFX_Double_Bulk 函数将 ODBC 数据源的数据列中的多行双精度浮点数据传送给 CRecordset 的派生对象中的相应数组。数据源的列必须是 SQL_DOUBLE 类型的。记录集必须定义一个 double 指针型的字段数据成员。

如果你把 *prgDblVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

注意　成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见“RFX_Text_Bulk”

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Int

void RFX_Int(CFieldExchange* *pFX*, const char* *szName*, int& *value*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员所保存的值----要被交换的值。对于从记录集到数据源的数据传送，int 类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_Int 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_SMALLINT 型数据列之间交换整数数据。

示例

参见“RFX_Text.”

请参阅 RFX_Text, RFX_Bool, RFX_Long, RFX_Single, RFX_Double, RFX_Date, RFX_Byte, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Int_Bulk

void RFX_Int_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, int** *prgIntVals*, long** *prgLengths*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

prgIntVals

指向整数数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgIntVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

说明

RFX_Int_Bulk 函数将 ODBC 数据源的数据列中的多行整数数据传送给

CRecordset 的派生对象中的相应数组。数据源的列必须是 SQL_SMALLINT 类型的。记录集必须定义一个 int 指针型的字段数据成员。

如果你把 *prgIntVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

> **注意** 成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见“RFX_Text_Bulk.”

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Long

void RFX_Long(CFieldExchange* *pFX*, const char* *szName*, LONG& *value*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员所保存的值––––要被交换的值。对于从记录集到数据源的数据传送，long 类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_Long 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_INTEGER 型数据列之间交换整数数据。

示例

参见“RFX_Text.”

请参阅 RFX_Text, RFX_Bool, RFX_Int, RFX_Single, RFX_Double, RFX_Date, RFX_Byte, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Long_Bulk

void RFX_Long_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, long** *prgLongVals*, long** *prgLengths*);

参数

pFX

指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象能指定的操作的更多信息参见“**Visual C++**程序员指南”中的文章“记录字段交换：**RFX **如何工作”。**

szName

数据列的名字。

prgLongVals

指向长整数数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 prgLongVals 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见 ODBC SDK 程序员参考中的 ODBC API 函数 SQLBindCol。

说明

RFX_Long_Bulk 函数将 ODBC 数据源的数据列中的多行长整数数据传送给 CRecordset 的派生对象中的相应数组。数据源的列必须是 SQL_INTEGER 类型的。记录集必须定义一个 long 指针型的字段数据成员。

如果你把 *prgLongVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

注意 成组记录字段交换仅把数据从数据源传送到记录集对象。要使你的记录集可更新，必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见“RFX_Text_Bulk.”

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Single_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_LongBinary

void RFX_LongBinary(CFieldExchange* *pFX*, const char* *szName*, CLongBinary& *value*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX如何工作”。

szName

数据列的名字。

value

指定的数据成员所保存的值——要被交换的值。对于从记录集到数据源的数据传送，CLongBinary 类型的值是从指定的数据成员中获得的。对

于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_LongBinary 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_LONGV- ARBINARY 或 SQL_LONGVARCHAR 类型的数据列之间传递使用 CLongBinary 类的二进制大对象（BLOB）数据。

示例

参见“RFX_Text.”

请参阅 RFX_Text, RFX_Bool, RFX_Long, RFX_Int, RFX_Single, RFX_Double, RFX_Date, RFX_Byte, RFX_Binary, CFieldExchange::SetFieldType, CLongBinary

RFX_Single

void RFX_Single(CFieldExchange* *pFX*, const char* *szName*, float& *value*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

value

指定的数据成员所保存的值----要被交换的值。对于从记录集到数据源的数据传送，float 类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。

说明

RFX_Single 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_REAL 型数据列之间交换浮点数据。

示例

参见“RFX_Text.”

请参阅 RFX_Text, RFX_Bool, RFX_Long, RFX_Int, RFX_Double, RFX_Date, RFX_Byte, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Single_Bulk

void RFX_Single_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, float** *prgFltVals*, long** *prgLengths*);

参数

pFX

指向 CFieldExchange 类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关 CFieldExchange 对象能指定的操作的更多信息参见“Visual C++程序员指南”中的文章“记录字段交换：RFX 如何工作”。

szName

数据列的名字。

prgFltVals

指向 float 数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgFltVals* 指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个 NULL 值，那么这个数组中将保存一个 SQL_NULL_DATA 值。更多的细节参见

“ODBC SDK 程序员参考”中的 ODBC API 函数 SQLBindCol。

说明

RFX_Single_Bulk 函数将 ODBC 数据源的数据列中的多行浮点数据传送给 CRecordset 的派生对象中的相应数组。数据源的列必须是 SQL_REAL 类型的。记录集必须定义一个 float 指针型的字段数据成员。

如果你把 *prgFltVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

> **注意** 成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，你必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

参见“RFX_Text_Bulk.”

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Text_Bulk, CFieldExchange::SetFieldType

RFX_Text

void RFX_Text(CFieldExchange* *pFX*, const char* *szName*, CString& *value*, int *nMaxLength* = 255, int *nColumnType* = SQL_VARCHAR, short *nScale* = 0);

参数

pFX

指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象能指定的操作的更多信息参见“** Visual C++**程序员指南** **”中的文章“记录字段交换：**RFX **如何工作”。**

szName

数据列的名字。

value

指定的数据成员所保存的值––––**要被交换的值。对于从记录集到数据源的数据传送，**CString **类型的值是从指定的数据成员中获得的。对于从数据源到记录集的数据传送，该值保存在指定的数据成员中。**

nMaxLength

将要被传送的字符串或数组的最大长度。nMaxLength **的缺省值是** 255**，合法的值为** 1 **到** INT_MAX**。框架会为数据分配这么多内存。为了获得**

最好的性能，应当传递一个足够大的值，使之能够容纳你要求的最大的数据项。

nColumnType

主要为参数使用。一个指定了参数类型的整数。参数的类型是 ODBC 数据类型 SQL_XXX 中的一个。

nScale

指定了 SQL_DECIMAL 或 SQL_NUMERIC 类型的值的范围。nScale 只在设置参数值的时候才有用。更多的信息参见“ODBC SDK 程序员参考”的附录 D 中的文章“精度，范围，长度和显示大小”。

说明

RFX_Text 函数在 CRecordset 对象的字段数据成员和数据源中记录的 SQL_LONGVA- RCHAR，SQL_CHAR，SQL_VARCHAR，SQL_DECIMAL 或 SQL_NUMERIC 型数据列之间交换 CString 数据。

示例

这个例子演示了几个对 RFX_Text 的调用。注意其中还包括了对 CFieldExchange::Set- FieldType 的两次调用。通常 ClassWizard 会写入对 SetFieldType 的第二次调用以及相关的 RFX 调用。你必须自己写入对 SetFieldType 的第一次调用以及相关的 RFX 调用。我们建议你把参数项放在

“//{{AFX_FIELD_MAP”注释之外。必须把参数放在注释之外。

```
// RFX_Text的例子
void CSections::DoFieldExchange(CFieldExchange* pFX)
{
    pFX->SetFieldType(CFieldExchange::inputParam);
    RFX_Text(pFX, "Name", m_strNameParam);
    //{{AFX_FIELD_MAP(CSections)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, "CourseID", m_strCourseID);
    RFX_Text(pFX, "InstructorID", m_strInstructorID);
    RFX_Int(pFX, "RoomNo", m_nRoomNo);
    RFX_Text(pFX, "Schedule", m_strSchedule);
    RFX_Int(pFX, "SectionNo", m_nSectionNo);
    RFX_Single(pFX, "LabFee", m_flLabFee);
    //}}AFX_FIELD_MAP
}
```

请参阅 RFX_Bool, RFX_Long, RFX_Int, RFX_Single, RFX_Double, RFX_Date, RFX_Byte, RFX_Binary, RFX_LongBinary, CFieldExchange::SetFieldType

RFX_Text_Bulk

void RFX_Text_Bulk(CFieldExchange* *pFX*, LPCTSTR *szName*, LPSTR* *prgStrVals*, long** *prgLengths*, int *nMaxLength*);

参数

pFX

指向 CFieldExchange **类对象的指针。这个对象包含了一些信息，用于定义函数调用的上下文环境。有关** CFieldExchange **对象能指定的操作的更多信息参见“** Visual C++**程序员指南”中的文章“记录字段交换：**RFX **如何工作”。**

szName

数据列的名字。

prgStrVals

指向 LPSTR **数组的指针。这个数组中保存了将要从数据源传送到记录集的数据。注意在** ODBC **的当前版本中，这些值不能是** Unicode **的。**

prgLengths

指向长整数数组的指针。这个数组中保存了 *prgFltVals* **指向的数组中每个值的以字节为单位的长度。注意如果对应的数据项中是个** NULL **值，那么这个数组中将保存一个** SQL_NULL_DATA **值。更多的细节参见**

“ODBC SDK 程序员参考”中的 ODBC API 函数 SQLBindCol。

nMaxLength

prgStrVals 所指向的数组中所能保存的值的最大长度，包括 null 结束字符。为了确保数据不被截断，应当传递一个足够大的值，使之能够容纳你要求的最大的数据项。

说明

RFX_Text_Bulk 函数将 ODBC 数据源的数据列中的多行字符数据传送给 CRecordset 的派生对象中的相应数组。数据源的列可以是 SQL_LONGVARCHAR，SQL_CHAR，SQL_VARCHAR，SQL_DECIMAL 或 SQL_NUMERIC 类型的。记录集必须定义一个 LPSTR 指针型的字段数据成员。

如果你把 *prgStrVals* 和 *prgLengths* 初始化为 NULL，那么它们指向的数组将自动分配，其大小等于行集的大小。

> **注意**　成组记录字段交换仅把数据从数据源传送到记录集对象。如果要使你的记录集可更新，你必须使用 ODBC 的 API 函数 SQLSetPos。这种做法的例子参见 DBFETCH。

更多的信息参见“Visual C++程序员指南”中的文章“记录集：成组获取记录（ODBC）”和“记录字段交换（RFX）”。

示例

ClassWizard **不支持** Bulk RFX **函数，因此你必须手动在** DoBulkFieldExchange **函数中写入调用。这个例子演示了对** RFX_Text_Bulk **的调用，同时还有对** RFX_Long_Bulk **的调用。这些调用的前面是** CFieldExchange::SetFieldType**。注意对于参数，你调用** RFX **函数，而不是** Bulk RFX **函数。**

```
void MultiRowSet::DoBulkFieldExchange( CFieldExchange* pFX )
{
    pFX->SetFieldType( CFieldExchange::outputColumn );
    RFX_Long_Bulk( pFX, _T( "[colRecID]" ),
                   &m_rgID, &m_rgIDLenghts );
    RFX_Text_Bulk( pFX, _T( "[colName]" ),
                   &m_rgName, &m_rgNameLengths, 30 );
    pFX->SetFieldType( CFieldExchange::inputParam );
    RFX_Text( pFX, "NameParam", m_strNameParam );
}
```

请参阅 RFX_Binary_Bulk, RFX_Bool_Bulk, RFX_Byte_Bulk, RFX_Date_Bulk, RFX_Double_Bulk, RFX_Int_Bulk, RFX_Long_Bulk, RFX_Single_Bulk, CFieldExchange::SetFieldType

RUNTIME_CLASS

RUNTIME_CLASS(*class_name*)

参数

class_name

类的实际名字（不用引号括起来）。

说明

利用这个宏通过 C++类的名字获得一个运行时类结构。

RUNTIME_CLASS 为 class_name 指定的类返回一个指向 CRuntimeClass 结构的指针。只有用 DECLARE_DYNAMIC，DECLARE_DYNCREATE 或 DECLARE_SERIAL 定义的 CObject 的派生类才能返回 CRuntimeClass 结构指针。

更多的信息参见“Visual C++程序员指南”中的“CObject 类”主题。

示例

```
// RUNTIME_CLASS 的例子
CRuntimeClass* prt = RUNTIME_CLASS( CAge );
```

ASSERT(lstrcmp(prt->m_lpszClassName, "CAge") == 0);

请参阅 DECLARE_DYNAMIC, DECLARE_DYNCREATE, DECLARE_SERIAL, CObject::GetRuntimeClass, CRuntimeClass

SerializeElements

template< class *TYPE* > void AFXAPI SerializeElements(CArchive& *ar*, *TYPE* pElements*, int *nCount*);

参数

TYPE

指定了元素类型的模板参数。

ar

用来写入存档或读出存档的档案对象。

pElements

指向要被存档的元素的指针。

nCount

要存档的元素的数目。

说明

CArray，CList **和** CMap **调用这个函数以串行化元素。缺省的实现进行按位读写。**

有关这个函数和其它帮助函数的实现的信息参见“Visual C++**程序员指南**”**中的文章**“**集合：如何生成类型安全的集合**”**。**

请参阅 **CArchive**

STATIC_DOWNCAST

STATIC_DOWNCAST(*class_name*, *pobject*)

参数

class_name
    类的名字。

pobject
    将要被强制转换为 *class_name* **类型的对象指针的指针。**

说明

如果在创建你的应用程序时定义了_DEBUG 预处理符号，这个宏将把一个对象指针从一种类型转换为一种相关的类型。如果指针不为 NULL，并且指向的对象不属于目标类型的话，这个宏将会引起 ASSERT。

在没有_DEBUG 的版本中，这个宏不作任何检查就进行转换。

目标类型由 *class_name* 参数指定，而 *pobject* 参数标识了指针。例如，你可能会通过以下表达式把一个名为 pYourDoc 的 CYourDocument 指针转换为 CDocument 指针：

```
CDocument* pDoc = STATIC_DOWNCAST(CDocument, pYourDoc);
```

如果 pYourDoc 并不指向一个 CDocument 对象，那么这个宏将会引起 ASSERT。

请参阅　DYNAMIC_DOWNCAST

THIS_FILE

说明

这个宏展开要被编译的文件的名字。这个信息被 ASSERT 和 VERIFY 宏所使用。AppWizard 和 ClassWizard 在它们生成的源代码文件中写入这个宏。

请参阅　ASSERT, VERIFY

THROW

THROW(*exception_object_pointer*)

参数

exception_object_pointer

指向从 CException **继承的异常对象的指针。**

说明

抛出指定的异常。THROW **中断程序的运行，将控制权转移到你的程序中相关的** CATCH **块。如果你没有提供** CATCH **块，那么控制权将会转移到微软基础类库的模块，它打印出一条错误信息，然后退出。**

更多的信息参见“Visual C++**程序员指南**”**中的文章**“**异常**”**。**

请参阅 THROW_LAST, TRY, CATCH, AND_CATCH, END_CATCH, CATCH_ALL, AND_CATCH_ALL, END_CATCH_ALL, AfxThrowArchiveException, AfxThrowFileException, AfxThrowMemoryException, AfxThrowNotSupportedExce-ption, AfxThrowResourceException, AfxThrowUserException

THROW_LAST

THROW_LAST( )

说明

将异常抛回下一个外部 CATCH **块。**

这个宏使你能够抛出一个本地产生的异常。如果你试图抛出一个刚捕捉到的异常，通常它会越过作用字段并被删除。通过 THROW_LAST**，这个异常就可以被正确地传递给下一个** CATCH **处理函数。**

更多的信息参见" Visual C++**程序员指南"中的文章"异常"。**

请参阅 THROW, TRY, CATCH, AND_CATCH, END_CATCH, CATCH_ALL, AND_CATCH_ALL, END_CATCH_ALL

TRACE

TRACE(*exp*)

参数

exp

指定了可变数目的参数，它们的使用方式与运行时函数 printf 中的可变数目的参数相同。

说明

提供了与 printf 函数类似的功能，可以向转储设备，例如文件或调试终端发送格式化字符串。与 MS-DOS 下的 C 程序中的 printf 类似，TRACE 宏是在程序指向时跟踪变量值的简便方式。在调试环境中，TRACE 宏的输出发送到 afxDump。在发行环境中，它不做任何操作。

TRACE 宏每次最多可以发送 512 个字符。如果你通过格式化命令调用 TRACE 宏，被展开后的格式化命令的字符串总长度不能超过 512，包括结尾的 NULL 字符。超出了这个限制就会引起 ASSERT。

注意 这个宏仅在 MFC 的调试版本中有效。

更多的信息参见“Visual C++程序员指南”中的文章“MFC 调试支持”。

示例

```
// TRACE 的例子
int i = 1;
char sz[] = "one";
TRACE( "Integer = %d, String = %s\n", i, sz );
```

// output： 'Integer = 1, String = one'

请参阅 TRACE0, TRACE1, TRACE2, TRACE3, AfxDump, afxTraceEnabled

TRACE0

TRACE0(*exp*)

参数

exp

与运行时函数 printf **中类似的格式字符串。**

说明

TRACE0 **与** TRACE **类似，是一组跟踪宏中的一个形式，可以用它们来产生调试输出。这组宏包括：**

- TRACE0 -- **接收一个格式字符串，可以用于将简单的文本消息转储到** afxDump。
- TRACE1 -- **接收一个格式字符串和一个参数（一个要被转储到** afxDump **的变量）。**
- TRACE2 -- **接收一个格式字符串和两个参数（两个要被转储到** afxDump

的变量）。

- TRACE3 -- 接收一个格式字符串和三个参数（三个要被转储到 afxDump 的变量）。

如果你生成了应用程序的发行版本，TRACE0 不做任何操作。与 TRACE 一样，它仅在应用程序的调试版本中才把数据转储到 afxDump 中。

注意 这个宏仅在 MFC 的调试版本中有效。

示例

```
// TRACE0 的例子
TRACE0( "Start Dump of MyClass members:" );
```

请参阅 TRACE, TRACE1, TRACE2, TRACE3

TRACE1

TRACE1(*exp, param1*)

参数

exp

与运行时函数 printf 中类似的格式字符串。

param1

将被转储的变量的名字。

说明

TRACE1 宏的描述参见 TRACE0。

示例

```
// TRACE1 的例子
int i = 1;
TRACE1( "Integer = %d\n", i );
// output：  'Integer = 1'
```

TRACE2

TRACE2(*exp, param1, param2*)

参数

exp

与运行时函数 printf 中类似的格式字符串。

param1

要被转储的变量的名字。

param2

要被转储的变量的名字。

说明

TRACE2 的描述参见 TRACE0。

示例

```
// TRACE2 的例子
int i = 1;
char sz[] = "one";
TRACE2( "Integer = %d, String = %s\n", i, sz );
// 输出： 'Integer = 1, String = one'
```

TRACE3

TRACE3(*exp, param1, param2, param3*)

参数

exp

　　与运行时函数 printf 中类似的格式字符串。

param1

　　要被转储的变量的名字。

param2

　　要被转储的变量的名字。

param3

　　要被转储的变量的名字。

说明

TRACE3 的描述参见 TRACE0。

TRY

TRY

说明

用这个宏来建立一个 TRY 块。TRY 块定义了一块代码，可以抛出异常。这些异常将被后面的 CATCH 和 AND_CATCH 块处理。允许递归：异常可以被传递给外部的 TRY 块,只要忽略它们或是使用 THROW_LAST 宏。用 END_CATCH 或 END_CATCH_ALL 宏来结束 TRY 块。

更多的信息参见“Visual C++程序员指南”中的文章“异常”。

请参阅 CATCH, AND_CATCH, END_CATCH, CATCH_ALL, AND_CATCH_ALL, END_CATCH_ALL, THROW, THROW_LAST

VERIFY

VERIFY(*booleanExpression*)

参数

booleanExpression

指定了一个表达式（包含指针变量），其计算结果为非零值或 0。

说明

在 MFC 的调试版本中，VERIFY 宏计算它的参数。如果结果是 0，这个宏就打印出一条消息并中止程序。如果结果为非零值，它就不做任何操作。

诊断消息的形式如下：

assertion failed in file <name> in line <num>

这里的 *name* 是源文件的名字，*num* 是源文件中发生断言失败的位置的行号。

在 MFC 的发行版本中，VERIFY 计算表达式的值，但是并不输出或中断程序。例如，如果这个表达式是一个函数调用，这个调用会被执行。

请参阅 ASSERT

ClassWizard 注释分界符

为了使 ClassWizard 能够识别用户输入的代码和 ClassWizard 输入的代码，使用了一些特别的分界符。这些分界符在代码中以注释的形式出现。因此它们不会被 ClassWizard 以外的任何东西编译和修改。

下面是 ClassWizard 的注释分界符的列表。更多的信息参见本章内容。

注释分界符

AFX_DATA

标记着头文件（.H）中成员变量定义的开始和结束，用于对话框数据交换（DDX）。

AFX_DATA_INIT

标记着对话框的构造函数中对话框数据交换（DDX）成员变量的初始化过程的开始和结束。

AFX_DATA_MAP

标记着对话框类的 DoDataExchange 成员函数中对话框数据交换（DDX）函数调用的开始和结束。

AFX_DISP

标记着类的头文件（.H）中 OLE 自动化声明的开始和结束。

AFX_DISP_MAP

标记着类的实现文件（.CPP）中 OLE 自动化映射的开始和结束。

AFX_EVENT

标记这类的头文件（.H）中 OLE 事件声明的开始和结束。

AFX_EVENT_MAP

标记着类的实现文件（.CPP）中 OLE 事件的开始和结束。

AFX_FIELD

标记着数据库的记录字段交换（RFX）的头文件（.H）中成员变量定义的开始和结束。

AFX_FIELD_INIT

标记着记录集类的构造函数中记录字段交换（RFX）成员变量初始化过程的开始和结束。

AFX_FIELD_MAP

标记着记录集类的 DoFieldExchange 成员函数中记录字段交换函数调用的开始和结束。

AFX_MSG

标记着头文件（.H）中与消息映射有关的 ClassWizard 入口的开始和结束。

AFX_MSG_MAP

标记着类的消息映射（在.CPP 文件中）中消息映射入口的开始和结束。

AFX_VIRTUAL

标记着类的头文件（.H）中虚拟函数重载声明的开始和结束。

AFX_DATA

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_DATA 被用来标记头文件（.H）中成员变量定义的开始和结束，用于对话框数据交换（DDX）：

```
//{{AFX_DATA(classname)
...
//}}AFX_DATA
```

更多的信息参见 AFX_DATA_MAP 和 AFX_DATA_INIT。

AFX_DATA_INIT

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_DATA_INIT 被用来标记对话框类的构造函数中对话框数据交换（DDX）成员变量初始化过程的开始和结束：

```
//{{AFX_DATA_INIT(classname)
```

...
//}}AFX_DATA_INIT

更多的信息参见 AFX_DATA_MAP 和 AFX_DATA_。

AFX_DATA_MAP

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_DATA_MAP 被用来标记对话框类的 DoDataExchange 成员函数中对话框数据交换（DDX）函数调用的开始和结束：

```
//{{AFX_DATA_MAP(classname)
...
//}}AFX_DATA_MAP
```

更多的信息参见 AFX_DATA_INIT 和 AFX_DATA_。

AFX_DISP

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_DISP 被用来标记类的头文件（.H）中 OLE 自动化声明的开始和结束：

```
//{{AFX_DISP(classname)
...
//}}AFX_DISP
```

更多的信息参见 AFX_DISP_MAP。

AFX_DISP_MAP

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_DISP_MAP 被用来标记类的实现文件（.CPP）中 OLE 自动化映射的开始和结束：

```
//{{AFX_DISP_MAP(classname)
```

...
//}}AFX_DISP_MAP

更多的信息参见 AFX_DISP。

AFX_EVENT

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_EVENT 被用来标记类的头文件（.H）中 OLE 事件声明的开始和结束：

```
//{{AFX_EVENT(classname)
...
//}}AFX_EVENT
```

更多的信息参见 AFX_EVENT_MAP。

AFX_EVENT_MAP

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_EVENT_MAP 被用来标记类的实现文件（.CPP）中 OLE 事件的开始和结束：

```
//{{AFX_EVENT_MAP(classname)
...
//}}AFX_EVENT_MAP
```

更多的信息参见 AFX_EVENT。

AFX_FIELD

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_FIELD 被用来标记类的头文件（.H）中成员变量声明的开始和结束，用于数据库的记录字段交换（RFX）：

```
//{{AFX_FIELD(classname)
```

```
...
//}}AFX_FIELD
```

更多的信息参见 AFX_FIELD_MAP 和 AFX_FIELD_INIT。

AFX_FIELD_INIT

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_FIELD_INIT 被用来标记记录集类的构造函数中记录字段交换（RFX）成员变量初始化过程的开始和结束：

```
//{{AFX_DATA_FIELD(classname)
...
//}}AFX_DATA_FIELD
```

更多的信息参见 AFX_FIELD_MAP 和 AFX_FIELD。

AFX_FIELD_MAP

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_FIELD_MAP 被用来标记记录集类的 DoFieldExchange 成员函数中记录字段交换（RFX）函数调用的开始和结束：

```
//{{AFX_FIELD_MAP(classname)
...
//}}AFX_FIELD_MAP
```

更多的信息参见 AFX_FIELD_INIT 和 AFX_FIELD。

AFX_MSG

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_MSG 被用来标记类的头文件（.H）中与消息映射有关的 ClassWizard 入口的开始和结束：

```
//{{AFX_MSG(classname)
```

```
...
//}}AFX_MSG
```

更多的信息参见 AFX_MSG_MAP。

AFX_MSG_MAP

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_MSG_MAP 被用来标记类的消息映射（在.CPP 文件中）中消息映射入口的开始和结束：

```
//{{AFX_MSG_MAP(classname)
...
//}}AFX_MSG_MAP
```

更多的信息参见 AFX_MSG。

AFX_VIRTUAL

说明

ClassWizard 和 AppWizard 在你的源代码文件中插入特别格式的注释分界符以标明 ClassWizard 可以写入的位置。AFX_VIRTUAL 被用来标记类的头文件（.H）中虚拟函数重载声明的开始和结束：

```
//{{AFX_VIRTUAL(classname)
...
//}}AFX_VIRTUAL
```

这些在.CPP 文件中没有对应的内容。

结构、风格、回调函数和消息映射

这个部分描述了微软基础类库使用的结构，风格和回调函数，还描述了 MFC 的消息映射。

MFC 使用的结构

这个主题后面是对不同成员函数调用的结构的描述。有关每个结构的用法的进一步信息参看每个结构的“请参阅”列表中列出的类和成员函数。

ABC 结构

ABC 结构具有如下形式：

```
typedef struct _ABC { /* abc */
    int       abcA;
    UINT      abcB;
```

```
    int      abcC;
} ABC;
```

ABC 结构包含了 TrueType 字体中字符的宽度。

成员

abcA

指定了字符的 A 宽度。A 宽度是指画出的字符图形离当前位置的距离。

abcB

指定了字符的 B 宽度。B 宽度是指画出的字符图形的宽度。

abcC

指定了字符的 C 宽度。C 宽度是指加在当前位置上的距离，用于提供字符图形右边的空白。

注释

字符的总宽度是 A，B 和 C 宽度的总和。A 或 C 宽度都可以是负的，指明上悬或下垂。

请参阅 CDC::GetCharABCWidths

ABCFLOAT 结构

ABCFLOAT 结构具有如下形式：

```
typedef struct _ABCFLOAT { /* abcf */
      FLOAT    abcfA;
      FLOAT    abcfB;
      FLOAT    abcfC;
} ABCFLOAT;
```

ABCFLOAT 结构包含了字符的 A，B 和 C 宽度。

成员

abcfA

指定了字符的 A 宽度。A 宽度是指画出的字符图形离当前位置的距离。

abcfB

指定了字符的 B 宽度。B 宽度是指画出的字符图形的宽度。

abcfC

指定了字符的 C 宽度。C 宽度是指加在当前位置上的距离，用于提供字符图形右边的空白。

注释

A，B 和 C 宽度是根据字体的基准线测量的。字符的增量（总宽度）是 A，B 和 C 宽度的总和。A 或 C 宽度都可以是负数，表明上悬或下垂。

请参阅 CDC::GetCharABCWidths

AFX_EXTENSION_MODULE 结构

AFX_EXTENSION_MODULE 结构具有如下形式：

```
struct AFX_EXTENSION_MODULE
{
    BOOL bInitialized;
    HMODULE hModule;
    HMODULE hResource;
    CRuntimeClass* pFirstSharedClass;
    COleObjectFactory* pFirstSharedFactory;
};
```

AFX_EXTENSION_MODULE 被用在 MFC 扩展 DLL 的初始化过程中，用于保存扩展 DLL 模块的状态。

成员

bInitialized

如果 DLL 模块已经用 AfxInitExtensionModule 初始化了，则为 TRUE。

hModule

指定了 DLL 模块的句柄。

hResource

指定了 DLL 的自定义资源模块的句柄。

pFirstSharedClass

指向有关 DLL 模块的第一个运行时类的信息（CRuntimeClass 结构）的指针。用于提供运行时类列表的开始部分。

pFirstSharedFactory

指向 DLL 模块的第一个对象工厂（COleObjectFactory 对象）的指针。用于提供类工厂列表的开始部分。

注释

MFC 扩展 DLL 需要在它们的 DllMain 函数中做两件事情：

- 调用 AfxInitExtensionModule 并检查返回值。
- 如果 DLL 要引出 CRuntimeClass 对象或者拥有它自己的资源，就创建一个

CDynLinkLibrary对象。

AFX_EXTENSION_MODULE结构被用于保存扩展DLL模块状态的一个备份，包括运行时对象的一个拷贝，它已经在进入DllMain函数之前被扩展DLL作为常规静态对象构造过程的一部分初始化过了。例如：

```
static AFX_EXTENSION_MODULE extensionDLL;
extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID)
{
	// 初始化这个DLL的扩展模块
	VERIFY(AfxInitExtensionModule(extensionDLL, hInstance));
```

AFX_EXTENSION_MODULE结构中保存的模块信息可以被拷贝到CDynLinkLibrary对象中。例如：

```
// CDynLinkLibrary 类
IMPLEMENT_DYNAMIC(CDynLinkLibrary, CCmdTarget)
// 构造函数
CDynLinkLibrary::CDynLinkLibrary(AFX_EXTENSION_MODULE& state,
BOOL bSystem)
{
#ifndef _AFX_NO_OLE_SUPPORT
    m_factoryList.Construct(offsetof(COleObjectFactory, m_pNextFactory));
#endif
```

```
    m_classList.Construct(offsetof(CRuntimeClass, m_pNextClass));
    // 从 AFX_EXTENSION_MODULE 结构中拷贝信息
    ASSERT(state.hModule != NULL);
    m_hModule = state.hModule;
    m_hResource = state.hResource;
    m_classList.m_pHead = state.pFirstSharedClass;
#ifndef _AFX_NO_OLE_SUPPORT
    m_factoryList.m_pHead = state.pFirstSharedFactory;
#endif
    m_bSystem = bSystem;
```

请参阅 AfxInitExtensionModule, AfxTermExtensionModule

BITMAP 结构

BITMAP 结构具有如下形式：

```
typedef struct tagBITMAP {    /* bm */
     int        bmType;
     int        bmWidth;
     int        bmHeight;
     int        bmWidthBytes;
     BYTE       bmPlanes;
```

```
    BYTE      bmBitsPixel;
    LPVOID    bmBits;
} BITMAP;
```

BITMAP 结构定义了逻辑位图的高，宽，颜色格式和位值。

成员

bmType

指定了位图的类型。对于逻辑位图，这个成员必须为 0。

bmWidth

指定了位图的宽度，以像素为单位。宽度必须大于 0。

bmHeight

指定了位图的高度，以扫描行为单位。高度必须大于 0。

bmWidthBytes

指定了每个扫描行中字节的数目。这个值必须是个偶数，因为图形设备接口（GDI）假定位图中的位值构成一个整数（2 字节）数组。换句话说，bmWidthBytes*8 必须是 16 的倍数，大于或等于 bmWidth 与 bmBitsPixel 相乘所得的值。

bmPlanes

指定了位图中颜色平面的数目。

bmBitsPixel

指定了每个位平面中用于定义一个像素所需的颜色位数。

bmBits

指向位图中位值的位置。bmBits 成员必须是一个指向单字节数组的长指针。

注释

现在使用的位图格式有单色的和彩色的。单色的位图使用每个位平面一位的格式。每个扫描线是 16 的倍数。

对于一个高度为 n 的单色位图，扫描线是按照如下方式组织的：

Scan 0
Scan 1
.
.
.
Scan n-2
Scan n-1

单色设备上的像素不是黑就是白。如果位图中对应的位是 1，则像素就被打开（白）。如果对应的位是 0，则像素被关闭（黑）。

所有具有 RC_BITBLT 位的设备都支持位图，该位是在 CDC::GetDeviceCaps 成员函数的 RASTERCAPS 索引中设置的。

每个设备都有它自己的颜色格式。为了在不同的设备间传递位图，使用 Windows 的 GetDIBits 和 SetDIBits 函数。

请参阅 CBitmap::CreateBitmapIndirect

BITMAPINFO 结构

BITMAPINFO 结构具有如下形式：

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER    bmiHeader;
    RGBQUAD             bmiColors[1];
} BITMAPINFO;
```

BITMAPINFO 结构定义了 Windows 设备无关位图（DIB）的度量和颜色信息。

成员

bmiHeader

指定了一个 BITMAPINFOHEADER 结构，包含了有关设备相关位图的度量和颜色格式的信息。

bmiColors

指定了一个 RGBQUAD 或 DWORD 数据类型的数组，定义了位图中的颜色。

注释

设备无关位图由两个部分组成：一个 BITMAPINFO 结构，描述了位图的度量和颜色信息；一个字节数组，定义了位图的像素。数组中的字节被组合在一起，但是每个扫描行必须用零填补，在一个 LONG 边界结束。如果高度为正的，位图的起始位置在左下角。如果高度为负，起始位置在左上角。

BITMAPINFOHEADER 结构中的 biBitCount 成员决定了定义像素的位数以及位图中的最大颜色数。这个成员可以是下列值之一：

- 位图是单色的，bmiColors 成员包含两个入口。位图数组中的每一位代表一个像素。如果该位被清除，则用 bmiColors 表中的第一种颜色显示该像素。如果该位被置位，则用表中的第二种颜色显示该像素。
- 位图最多有 16 种颜色，bmiColors 成员中包含了最多可达 16 个入口。位图中的每个像素用一个 4 位的值来表示，该值用作颜色表的索引。例如，如果位图中的第一个字节是 0x1F，这个字节代表两个像素。第一个像素包含了颜色表中第二种颜色，第二个像素包含了颜色表中第十六种颜色。
- 位图最多有 256 种颜色，bmiColors 成员包含了多达 256 个入口。在这种情况下，数组中的每个字节代表一个像素。

- 位图最多有 216 种颜色。BITMAPINFOHEADER 的 biCompression 成员必须是 BI_BITFIELDS。bmiColors 成员包含了 3 个 DWORD 型颜色掩码，分别代表了每个像素中的红，绿和蓝色成分。DWORD 型掩码中的位必须是连续的，不能与其它掩码重叠。并非像素中的所有位都必须被使用。数组中的每个 WORD 值代表一个像素。
- 位图最多具有 224 种颜色，bmiColors 成员为 NULL。位图数组中的每个三字节组合分别代表像素中蓝，绿红的深度。
- 位图中最多具有 232 种颜色。BITMAPINFOHEADER 中的 biCompression 成员必须是 BI_BITFIELDS。bmiColors 成员中包含了三个 DWORD 颜色掩码，分别指定了像素的红，绿和蓝成分。DWORD 掩码中的位必须是连续的，并且不能与其它掩码重叠。并非像素中的所有位都必须被使用。数组中的每个 DWORD 值代表一个像素。

BITMAPINFOHEADER 结构中的 biClrUsed 成员指定了颜色表中实际使用的索引的数目。如果 biClrUsed 成员被设为 0，位图将使用 biBitCount 成员中指定的最大颜色数。

bmiColors 表中的颜色应当按照其重要性的顺序出现。另一种情况是，对于使用 DIB 函数，bmiColors 成员可以是一个 16 位无符号整数的数组，指定了当前实现的逻辑调色板中的索引，而不是确切的 RGB 值。在这种情况下，使用位图的应用程序必须调用 Windows 的 DIB 函数（CreateDIBitmap，CreateDIBPatternBrush 和 CreateDIBSection），iUsage 参数应被设为 DIB_PAL_COLORS。

如果位图是一个压缩位图（这意味着，这种位图的数组直接跟在一个指针所引用的 BITMAPINFO 头的后面），在使用 DIB_PAL_COLORS 模式的时候，biClrUsed 成员必须被设为偶数，以便使 DIB 位图数组从 DWORD 边界开始。

> **注意** 如果位图被保存在文件中，或者要被传送到另一个应用程序，bmiColors 成员不能包含调色板索引。除非应用程序独占地使用和控制位图，位图的颜色表中应当包含准确的 RGB 值。

请参阅 CBrush::CreateDIBPatternBrush

CDaoDatabaseInfo 结构

CDaoDatabaseInfo 结构具有如下形式：

```
struct CDaoDatabaseInfo
{
      CString m_strName;             // Primary
      BOOL m_bUpdatable;             // Primary
      BOOL m_bTransactions;          // Primary
      CString m_strVersion;          // Secondary
      Long m_lCollatingOrder;        // Secondary
      short m_nQueryTimeout;         // Secondary
      CString m_strConnect;          // All
```

};

CDaoDatabaseInfo 结构中包含了与数据库对象有关的信息，该对象是为数据访问对象（DAO）定义的。这个数据库对象是 CDaoDatabase 类的 MFC 对象下面的一个 DAO 对象。对上面的 Primary，Secondary 和 All 的引用指明了 CDaoWorkspace::GetDatabaseInfo 成员函数是如何返回信息的。

成员

m_strName

数据库对象的唯一名字。要直接获得这个属性的值，调用 CDaoDatabase::GetName。有关的细节参见 DAO 帮助的“名字属性”主题。

m_bUpdatable

指明是否可以对数据库作出改变。要直接获得这个属性的值，调用 CDaoDatabase:: CanUpdate。有关的细节参见 DAO 帮助的“可更新的属性”主题。

m_bTransactions

指明一个数据源是否支持事务——一系列改变的记录，以后可以被滚动回去（取消）或执行（保存）。如果数据库是基于 Microsoft Jet 数据库引擎的，Transaction 属性是非零值，你可以使用事务。其它数据库引擎可能不支持事务。要直接获取这个属性，调用

CDaoDatabase::CanTransact。有关的细节参见 DAO 帮助的“事务属性”主题。

m_strVersion

指明了 Microsoft Jet 数据库引擎的版本。要直接获取这个属性的值，调用数据库对象的 GetVersion 成员函数。有关的细节参见 DAO 帮助的“版本属性”主题。

m_lCollatingOrder

指定了字符串比较或排序中使用的顺序。可能的取值包括：

- dbSortGeneral 使用一般（English，French，German，Portuguese，Italian 和 Modern Spanish）的排序顺序。
- dbSortArabic 使用 Arabic 排序顺序。
- dbSortCyrillic 使用 Russian 排序顺序。
- dbSortCzech 使用 Czech 排序顺序。
- dbSortDutch 使用 Dutch 排序顺序。
- dbSortGreek 使用 Greek 排序顺序。
- dbSortHebrew 使用 Hebrew 排序顺序。
- dbSortHungarian 使用 Hungarian 排序顺序。
- dbSortIcelandic 使用 Icelandic 排序顺序。
- dbSortNorwdan 使用 Norwegian 或 Danish 排序顺序。
- dbSortPDXIntl 使用 Paradox International 排序顺序。

- dbSortPDXNor 使用 Paradox Norwegian 或 Danish 排序顺序。
- dbSortPDXSwe 使用 Paradox Swedish 或 Finnish 排序顺序。
- dbSortPolish 使用 Polish 排序顺序。
- dbSortSpanish 使用 Spanish 排序顺序。
- dbSortSwedFin 使用 Swedish 或 Finnish 排序顺序。
- dbSortTurkish 使用 Turkish 排序顺序。
- dbSortUndefined 使用的排序顺序未定义或未知。

更多的信息参见 DAO 帮助中的“数据访问的自定义 Windows 注册表设置”主题。

m_nQueryTimeout

当在 ODBC 数据库上执行一个查询时，Microsoft Jet 数据库引擎在超时错误发生前等待的秒数。缺省的超时值为 60 秒。当 QueryTimeout 被设为 0 时，不产生超时。这可能会引起程序挂起。要直接获得这个属性的值，调用数据库对象的 GetQueryTimeout 成员函数。有关的细节参见 DAO 帮助的“QueryTimeout 属性”主题。

m_strConnect

提供了有关打开的数据库的源的信息。有关连接字符串以及直接获取这个属性值的信息参见 CDaoDatabase::GetConnect 成员函数。更多的信息参见 DAO 帮助的“连接属性”主题。

注释

CDaoWorkspace::GetDatabaseInfo 成员函数所获得的信息被保存在 CDaoDatabaseInfo 结构中。为 CDaoWorkspace 对象调用 GetDatabaseInfo 函数，数据库对象被保存在该对象的数据库集合中。在调试版本中，CDaoDatabaseInfo 还定义了一个 Dump 成员函数。你可以使用 Dump 来转储 CDaoDatabaseInfo 对象的内容。

有关这个结构以及其它 MFC DAO Info 结构的信息参见“Visual C++程序员指南”中的文章“DAO 集合：获得有关 DAO 对象的信息”。

请参阅 CDaoWorkspace, CDaoDatabase, CDaoWorkspace::GetDatabaseCount

CDaoErrorInfo 结构

CDaoErrorInfo 结构具有以下形式：

```
struct CDaoErrorInfo
{
    long m_lErrorCode;
    CString m_strSource;
    CString m_strDescription;
    CString m_strHelpFile;
    long m_lHelpContext;
```

};

CDaoErrorInfo 结构包含了为数据访问对象（DAO）定义的错误对象的信息。MFC 没有将 DAO 错误对象封装为一个类。CDaoException 类提供了访问 DAO DBEngine 对象中包含的错误集合的接口，该对象也包括了所有的工作空间。当 MFC 的 DAO 操作抛出一个你捕捉到的 CDaoException 对象时，MFC 会填充 CDaoErrorInfo 结构并将它保存在异常对象的 m_pErrorInfo 成员中。（如果你选择了直接调用 DAO，你必须自己调用异常对象的 GetErrorInfo 函数以填充 m_pErrorInfo。）

有关处理 DAO 错误的更多信息参见“Visual C++程序员指南”中的文章“异常：数据库异常”。相关的信息参见 DAO 帮助的“错误对象”主题。

成员

m_lErrorCode

一个数字的 DAO 错误代码。参见 DAO 帮助的“可捕捉的数据访问错误”主题。

m_strSource

最初产生错误的对象或应用程序的名字。源属性指定了一个字符串表达式，代表最初产生错误的对象；这个表达式通常是对象的类名。有关的细节参见 DAO 帮助的“源属性”主题。

m_strDescription

与错误相关的描述性字符串。有关的细节参见DAO帮助的“描述属性”主题。

m_strHelpFile

Microsoft Windows帮助文件的经完全验证的路径。有关的细节参见DAO帮助的“帮助上下文，帮助文件属性”主题。

m_lHelpContext

Microsoft Windows的帮助文件中主题的上下文ID。有关的细节参见DAO帮助的“帮助上下文，帮助文件属性”主题。

说明

CDaoException::GetErrorInfo成员函数获得的信息被保存在一个CDaoErrorInfo结构中。在异常处理函数中检查你捕捉到的CDaoException对象的m_pErrorInfo数据成员，或者在你生成的CDaoException对象中调用GetErrorInfo函数，目的是对可能会在DAO接口的直接调用中产生的错误进行检查。在调试版本中，CDaoErrorInfo还定义了一个Dump成员函数，可以使用Dump来转储CDaoErrorInfo对象的内容。

有关这个结构和其它MFC的DAO Info结构的信息参见“Visual C++程序员指南”中的文章“DAO集合：获取有关DAO对象的信息”。

请参阅 CDaoException

CDaoFieldInfo 结构

CDaoFieldInfo **结构具有如下形式：**

```
struct CDaoFieldInfo
{
    CString m_strName;               // Primary
    short m_nType;                   // Primary
    long m_lSize;                    // Primary
    long m_lAttributes;              // Primary
    short m_nOrdinalPosition;        // Secondary
    BOOL m_bRequired;                // Secondary
    BOOL m_bAllowZeroLength;         // Secondary
    long m_lCollatingOrder;          // Secondary
    CString m_strForeignName;        // Secondary
    CString m_strSourceField;        // Secondary
    CString m_strSourceTable;        // Secondary
    CString m_strValidationRule;     // All
    CString m_strValidationText;     // All
    CString m_strDefaultValue;       // All
```

};

CDaoFieldInfo 结构中包含了为数据访问对象（DAO）定义的字段对象的信息。上面对 Primary，Secondary 和 All 的引用指明了 CDaoTableDef，CDaoQueryDef 和 CDaoRecordset 类中的 GetFieldInfo 成员函数是如何返回信息的。

MFC 类不代表任何字段对象。相反，DAO 对象构成了下面包含字段对象集合的 MFC 类的对象：CDaoTableDef，CDaoRecordset 和 CDaoQueryDef。这些类提供了用于访问字段信息中单个项的成员函数，也可以利用 CDaoFieldInfo 对象以及它的 GetFieldInfo 成员函数来一次访问所有的信息。

CDaoFieldInfo 不仅可以用于检查对象的属性，还可以用于构造一个输入参数，用来创建表中的新字段。对于这个任务，可以使用更简单的选项。但是要想获得更好的控制，可以使用 CDaoTableDef::CreateField 的带 CDaoFieldInfo 参数的版本。

成员

m_strName
　　字段对象的唯一名字。有关的细节参见 DAO 帮助中的“名字属性”主题。

m_nType
　　一个指明字段数据类型的值。有关的细节参见 DAO 帮助中的“类型属

性”主题。这个属性的取值可能是下列值之一：

- dbBoolean 是/否，与 TRUE/FALSE 相同
- dbByte 字节
- dbInteger 短整数
- dbLong 长整数
- dbCurrency Currency，参见 MFC 类 COleCurrency
- dbSingle 单精度数
- dbDouble 双精度数
- dbDate 日期/时间，参见 MFC 类 COleDateTime
- dbText 文本，参见 MFC 类 CString
- dbLongBinary 长二进制（OLE 对象）可能你希望使用 MFC 类 CByteArray 来代替 CLongBinary，因为 CByteArray 功能更强，也更易使用
- dbMemo 备注，参见 MFC 类 CString。
- dbGUID 用于远程过程调用的全局唯一的标识符。更多的信息参见 DAO 帮助中的“类型属性”主题。

注意 不要对二进制数据使用字符串数据类型。这会使你的数据被传递给 Unicode/ANSI 转换层，导致负担增加，并且可能会导致不希望出现的转换。

m_lSize

指明了包含文本的 DAO 字段对象的最大大小，以字节为单位，或者是包含了文本或数字值的字段对象的固定大小。有关的细节参见 DAO 帮助中的“大小属性”主题。大小可能是下列值之一：

类型	大小（Bytes）	描述
dbBoolean	1 字节	是/否（与 TRUE/FALSE 相同）
dbByte	1	字节
dbInteger	2	整数
dbLong	4	长整数
dbCurrency	8	Currency（COleCurrency）
dbSingle	4	单精度数
dbDouble	8	双精度数
dbDate	8	日期/时间（COleDateTime）
dbText	1～255	文本（CString）
dbLongBinary	0	长二进制数据（OLE 对象，CbyteArray，用于代替 CLongBinary）
dbMemo	0	备注（CString）
dbGUID	16	用于远程过程调用的全局唯一的标识符

m_lAttributes

指定了表定义，记录集，查询定义或索引对象中包含的字段对象的特征。返回的值可能是这些值的和，用 C++的位或操作符（|）生成：

- dbFixedField 字段大小是固定的（数字字段的缺省值）。

- dbVariableField 字段大小可变（仅对文本字段成立）。
- dbAutoIncrField 新记录的字段值自动增加到一个唯一的长整数，该数不能改变。仅支持 Microsoft Jet 数据库表。
- dbUpdatableField 字段的值可以改变。
- dbDescending 字段按照降序（Z～A 或 100～0）保存（仅适用于索引对象的字段集合中的字段对象。在 MFC 中，索引对象自己就包含在表定义对象中）。如果你省略了这个常量，字段就按照升序（A～Z 或 0～100）保存（缺省值）。

在检查这个属性的设置的时候，可以使用 C++的位与操作符（&）来测试某个指定的值。在设置多重属性的时候，可以用位或操作符（|）把适当的常量组合起来。有关的细节参见 DAO 帮助中的“Attribute 属性”主题。

m_nOrdinalPosition

一个指定了数字顺序的值，将按照这种顺序显示由 DAO 字段对象所代表的字段。你可以用 CDaoTableDef::CreateField 来设置这个属性。有关的细节参见 DAO 帮助中的“OrdinalPosition 属性”主题。

m_bRequired

指明一个 DAO 字段对象是否需要非 Null 值。如果这个属性值为 TRUE，那么这个字段不允许有 Null 值。如果 Required 被设为 FALSE，这个字段就可以有 Null 值，也可以有符合 AllowZeroLength 和 ValidationRule 属性所指定的条件的值。有关的细节参见 DAO 帮助中的“Required 属

性”主题。你可以使用 CDaoTableDef::CreateField 来设置这个属性。

m_bAllowZeroLength

指明对于一个文本类型或 Memo 数据类型的 DAO 字段对象，空字符串（“”）是否是有效的值。如果这个属性为 TRUE，空字符串就是有效值。你可以将这个属性设为 FALSE 以确保不能将这个字段的值设为空字符串。有关的细节参见 DAO 帮助中的“AllowZeroLength 属性”主题。你可以用 CDaoTableDef::CreateField 来设置这个属性。

m_lCollatingOrder

为字符串比较和排序指定了文本顺序。有关的细节参见 DAO 帮助中的“为数据访问自定义 Windows 注册表设置”主题。可能的返回值列表参见 CDaoDatabaseInfo 结构中的 m_lCollatingOrder 成员。你可以用 CDaoTableDef::CreateField 来设置这个属性。

m_strForeignName

一个值，在关系中指定了与原始表中的字段对应的外部表中的 DAO 字段对象的名字。有关的细节参见 DAO 帮助中的“ForeignName 属性”主题。

m_strSourceField

指定了一个字段名，是表定义，记录集或查询集对象所包含的 DAO 字段对象的数据源。例如，你可以使用这个属性来确定一个查询字段的数据源，这个字段的名字与基础表中的字段名无关。有关的细节参见 DAO

帮助中的“SourceField，SourceTable 属性”主题。可以用 CDaoTableDef::CreateField 来设置这个属性。

m_strSourceTable

指定了一个表名，是表定义，记录集或查询集对象的数据源。这个属性指定了与字段对象相关的原始表明。例如，你可以使用这个属性来确定一个查询字段的数据源，这个字段的名字与基础表中的字段名无关。有关的细节参见 DAO 帮助中的“SourceField，SourceTable 属性”主题。你可以用 CDaoTableDef::CreateField 来设置这个属性。

m_strValidationRule

当改变或增加字段的数据时用于检验数据是否有效的值。有关的细节参见 DAO 帮助中的“ValidationRule 属性”主题。你可以用 CDaoTableDef::CreateField 来设置这个属性。

有关表定义的信息参见 CDaoTableDefInfo 结构的 m_strValidationRule 成员。

m_strValidationText

一个值，指定了当 DAO 字段对象的值不符合 ValidationRule 属性所指定的有效规则时应用程序显示的消息文本。有关的细节参见 DAO 帮助中的“ValidationText 属性”主题。你可以用 CDaoTableDef::CreateField 来设置这个属性。

m_strDefaultValue

一个 DAO 字段对象的缺省值。当创建了一个新记录时，DefaultValue 属性会被作为字段的缺省值自动输入。有关的细节参见 DAO 帮助中的“DefaultValue 属性”主题。你可以用 CDaoTableDef::CreateField 来设置这个属性。

注释

通过 GetFieldInfo 成员函数（属于包含该字段的类）获取的信息保存在 CDaoFieldInfo 结构中。调用保存了字段对象的包含对象的 GetFieldInfo 成员函数。在调试版本中，CDaoFieldInfo 还定义了一个 Dump 成员函数。可以使用 Dump 来转储 CDaoFieldInfo 对象的内容。

有关这个结构以及其他 MFC DAO Info 结构的使用，参见“Visual C++程序员指南”中的文章“DAO 集合：获取 DAO 对象的信息”。

请参阅 CDaoTableDef::GetFieldInfo, CDaoRecordset::GetFieldInfo, CDaoQueryDef::GetFieldInfo

CDaoIndexInfo 结构

CDaoIndexInfo 结构具有如下形式：

struct CDaoIndexInfo {

```
    CDaoIndexInfo( );                          // Constructor
    CString m_strName;                         // Primary
    CDaoIndexFieldInfo* m_pFieldInfos;         // Primary
    short m_nFields;                           // Primary
    BOOL m_bPrimary;                           // Secondary
    BOOL m_bUnique;                            // Secondary
    BOOL m_bClustered;                         // Secondary
    BOOL m_bIgnoreNulls;                       // Secondary
    BOOL m_bRequired;                          // Secondary
    BOOL m_bForeign;                           // Secondary
    long m_lDistinctCount;                     // All
    // Below the // Implementation comment:
    // Destructor, not otherwise documented
};
```

CDaoIndexInfo 结构中包含了有关为数据访问对象（DAO）定义的索引对象的信息。上面对 Primary，Secondary 和 All 的引用指明了 CDaoTableDef 和 CDaoRecordset 类的成员函数 GetIndexInfo 是符合返回信息的。

MFC 类不代表索引对象。相反，DAO 对象构成了属于 CDaoTableDef 或 CDaoRecordset 类的 MFC 对象，它们包含了索引对象的集合，称为索引集合。这些类提供了用于访问索引信息的单个项的成员函数，还可以调用包含对象的 GetIndexInfo 成员函数，利用 CDaoIndexInfo 对象同时访问所有项。

CDaoIndexInfo 具有一个构造函数和一个析构函数，用于适当地分配和释放 m_pFieldInfos 中的索引字段信息。

成员

m_strName

字段对象的唯一名字。有关的细节参见 DAO 帮助中的“Name 属性”主题。

m_pFieldInfos

指向一个 CDaoIndexFieldInfo 对象数组的指针，该数组指明了表定义或记录集中哪个字段是索引的关键字段。每个对象标识了索引中的一个字段。缺省的索引顺序是升序。一个索引对象可以具有一个或多个索引关键字段。它们可以是升序，降序或其组合。

m_nFields

m_pFieldInfos 中保存的字段的数目。

m_bPrimary

如果 Primary 属性为 TRUE，则索引对象代表一个主索引。主索引由一个或多个字段组成，这些字段可以按照给定的顺序唯一地标识表中的所有记录。由于索引字段必须是唯一的，DAO 中，索引对象的 Unique 属性也被设为 TRUE。如果主索引由多于一个字段组成，每个字段都可以有重复的值，但是所有索引字段取值的组合必须是唯一的。主索引由表

的关键字组成，通常包含了与主键中相同的字段。

当为一个表设置主键的时候，主键被自动设为表的主索引。更多的信息参见 DAO 帮助中的“Primary 属性”和“Unique 属性”主题。

注意 每个表最多只能有一个主索引。

m_bUnique

指明一个索引对象是否是表的唯一索引。如果这个属性为 TRUE，那么这个索引对象代表了唯一的索引。唯一的索引由表中的一个或多个字段组成，它们在逻辑上将表中的所有记录按照给定的唯一顺序排列起来。如果这个索引由一个字段组成，这个字段中的值必须是独立的。如果索引由多个字段组成，那么每个字段都可以有重复的值，但是它们的组合必须是唯一的。

如果索引的 Unique 和 Primary 属性都被设为 TRUE，那么这个索引是主索引，并且是唯一的。他按照给定的逻辑顺序唯一地标识了表中的所有记录。如果 Primary 属性被设为 FALSE，这个索引是次索引。次索引（关键索引或非关键索引）按照给定的顺序在逻辑上排列表中的记录，但是并不能作为表中记录的标识使用。

更多的信息参见 DAO 帮助中的“Primary 属性”和“Unique 属性”主题。

m_bClustered

指明一个索引对象是否代表一个表的成簇索引。如果这个顺序被设为

TRUE，那么索引对象代表成簇索引，否则就不是。成簇索引由一个或多个非关键字段组成，它们一起按照给定的顺序排列表中的所有记录。有了成簇索引，表中的数据将按照索引指定的顺序存储。成簇索引提供了访问数据的有效方式。更多的信息参见 DAO 帮助中的“Clustered 属性”主题。

注意 对于使用 Microsoft Jet 数据库引擎的数据库，Clustered 属性会被忽略，因为 Jet 数据库引擎不支持成簇索引。

m_bIgnoreNulls

指明是否有记录的索引项在它们的索引字段中包含 Null 值。如果这个属性为 TRUE，具有 Null 值的字段就没有索引项。如果要使对记录的查找更快速，可以为这个字段定义索引。如果你允许建立索引的字段中包含 Null 值，并且可能有许多项为 Null，可以把索引对象的 IgnoreNulls 属性设为 TRUE 以减小索引使用的存储空间。IgnoreNulls 属性和 Required 属性的设置共同决定了一个具有 Null 索引值的记录是否具有索引项，如下面的表格所示：

IgnoreNulls	Required	索引字段中的 Null
True	False	允许有 Null 值，但没有索引项
False	False	允许有 Null 值，有索引项
True 或 False	True	不允许有 Null 值，没有索引项

更多的信息参见 DAO 帮助中的“IgnoreNulls 属性”主题。

m_bRequired

指明一个 DAO 索引对象是否要求非 Null 值。如果这个属性为 TRUE，那么索引对象不允许有 Null 值。更多的信息参见 DAO 帮助中的“Required 属性”主题。

Tip

当你能够为 DAO 索引对象或字段对象（被表定义，记录集或查询定义对象所包含）设置这个属性的时候，就为字段对象设置这个属性。为字段对象设置的属性的有效性检查将在索引对象之前进行。

m_bForeign

指明索引对象是否代表了表中的一个外部关键字。如果这个属性为 TRUE，则索引对象代表了表中的外部关键字。外部关键字由外部表中的一个或多个字段组成，他唯一确定了主表中的行。当你创建一个要求引用完整性的关系时，Microsoft Jet 数据库引擎为外部表创建一个索引对象并设置 Foreign 属性。更多的信息参见 DAO 帮助中的“Foreign 属性”主题。

m_lDistinctCount

指定了相关表中包含的索引对象的唯一值的数目。检查 DistinctCount 属性以确定索引中唯一值或键的数目。每个键只被计数一次，即使在索引允许重复值时，它们多次出现。这个信息对于试图通过统计索引信息来优化数据访问的应用程序非常有用。唯一值的数目还被当作索引对象的

势。DistinctCount 属性并不总是反映某一时刻关键字的数目。例如，由于取消事务所引起的改变不会被 DistinctCount 属性立即反映。更多的信息参见 DAO 帮助中的“DistinctCount 属性”主题。

注释

用表定义对象的 GetIndexInfo 成员函数获得的信息被保存在 CDaoIndexInfo 结构中。调用包含了索引对象的表定义对象的 GetIndexInfo 成员函数。在调试版本中 CDaoIndexInfo 还定义了一个 Dump 成员函数。你可以使用 Dump 来转储 CDaoIndexInfo 对象的内容。

有关这个结构以及其他 MFC DAO Info 结构的使用，参见“Visual C++程序员指南”中的文章“DAO 集合：获取 DAO 对象的信息”。

请参阅 CDaoTableDef::GetIndexInfo

CDaoIndexFieldInfo 结构

CDaoIndexFieldInfo 结构具有如下形式：

```
struct CDaoIndexFieldInfo
{
    CString m_strName;              // Primary
    BOOL m_bDescending;             // Primary
```

};

CDaoIndexFieldInfo 结构中包含了为数据访问对象（DAO）定义的索引字段对象的信息。索引对象可以有多个字段，指明表定义（或基于表的记录集）的索引建立在哪些字段上。上面对 Primary 的引用指明了调用 CDaoTableDef 或 CDaoRecordset 类的 GetIndexInfo 成员函数时获得的信息是如何通过 CDaoIndexInfo 对象的 m_pFiledInfos 成员返回的。

MFC 类不代表索引对象和索引字段对象。相反，DAO 对象构成了 CDaoTableDef 或 CDaoRecordset 类的 MFC 对象，这些对象中包含了索引对象的集合，名为索引集。每个索引对象包含了一个字段对象的集合。这些类提供了用于访问索引信息中单个项的成员函数，你也可以通过调用包含对象的 GetIndexInfo 成员函数在 CDaoIndexInfo 对象中访问所有的项。CDaoIndexInfo 中具有一个数据成员 m_pFieldInfos，它指向一个 CDaoIndexFieldInfo 对象的数组。

成员

m_strName

　　索引字段对象的唯一的名字。有关的细节参见 DAO 帮助中的“Name 属性”主体。

m_bDescending

　　指明了索引对象定义的索引顺序。如果顺序是降序的则该成员的值为 TRUE。

注释

调用表定义或记录集对象的 GetIndexInfo 成员函数，你感兴趣的索引对象就保存在这些对象的索引集中。然后访问 CDaoIndexInfo 对象的 m_pFieldInfos 成员。m_pFieldInfos 数组的长度保存在 m_nFields 中。CDaoIndexFieldInfo 还在调试版本中定义了一个 Dump 函数。你可以使用 Dump 来转储 CDaoIndexFieldInfo 对象的内容。

有关这个结构以及其他 MFC DAO Info 结构的使用参加“Visual C++程序员指南”中的文章“DAO 集合：获取 DAO 对象的信息”。

请参阅 CDaoTableDef::GetIndexInfo, CDaoRecordset::GetIndexInfo

CDaoParameterInfo 结构

CDaoParameterInfo 结构具有如下形式：

```
struct CDaoParameterInfo
{
    CString m_strName;           // Primary
    short m_nType;               // Primary
    COleVariant m_varValue;      // Secondary
};
```

CDaoParameterInfo 结构中包含了为数据访问对象（DAO）定义的参数对象的信息。上面对 Primary 和 Secondary 的引用指明了 CDaoQueryDef 类的成员函数 GetParameterInfo 是如何返回信息的。

MFC 没有把 DAO 参数对象封装为一个类。DAO 查询对象构成了 MFC 的 CDaoQueryDef 对象的基础，在它们的参数集中保存了参数。要访问 CDaoQueryDef 对象中的参数对象，就调用查询定义对象的成员函数 GetParameterInfo，可以获得特点的参数名或是参数集的索引。你可以与 GetParameterInfo 一起使用 CDaoQueryDef::GetParameterCount 函数，以在参数集中循环。

成员

m_strName

参数对象的唯一的名字。更多的信息参见 DAO 帮助中的“Name 属性”主题。

m_nType

指明了参数对象的数据类型的值。可能取值的列表参见 CDaoFieldInfo 结构的 m_nType 成员。更多的信息参见 DAO 帮助中的“Type 属性”主题。

m_varValue

参数的值，保存在 COleVariant 对象中。

注释

通过成员函数 CDaoQueryDef::GetParameterInfo 获得的信息保存在 CDaoParameterInfo 结构中。为查询定义对象调用 GetParameterInfo 函数，这些对象的参数集中保存了参数对象。

> 注意　如果你希望得到或设置参数的值，使用 CDaoRecordset 类的 GetParamValue 和 SetParamValue 成员函数。

CDaoParameterInfo 还为调试模式定义了一个 Dump 成员函数。你可以使用 Dump 来转储 CDaoParameterInfo 对象的内容。关于这个结构以及其他 MFC DAO Info 结构的使用参见“Visual C++ 程序员指南”中的文章“DAO 集合：获得 DAO 对象的信息”。

请参阅　CDaoQueryDef

CDaoQueryDefInfo 结构

CDaoQueryDefInfo 结构具有如下形式：

```
struct CDaoQueryDefInfo
{
    CString m_strName;                    // Primary
    short m_nType;                        // Primary
```

```
    COleDateTime m_dateCreated;          // Secondary
    COleDateTime m_dateLastUpdated;      // Secondary
    BOOL m_bUpdatable;                   // Secondary
    BOOL m_bReturnsRecords;              // Secondary
    CString m_strSQL;                    // All
    CString m_strConnect;                // All
    short m_nODBCTimeout;                // All
};
```

CDaoQueryDefInfo **结构包含了为数据访问对象（**DAO**）定义的查询定义对象的信息。查询定义对象是** CDaoQueryDef **类的对象。上面对** Primary**，**Secondary **和** All **的引用指明了** CDaoDatabase **类的成员函数** GetQueryDefInfo **是如何返回信息的。**

成员

m_strName

查询定义对象的唯一的名字。更多的信息参见 DAO **帮助中的**“Name **属性”主题。调用** CDaoQueryDef::GetName **以直接访问这个属性。**

m_nType

指明了查询定义对象的操作类型的值。其取值可能是下列值之一：

- dbQSelect　Select — **该查询选择记录。**

- dbQAction Action — 该查询移动或改变数据，但是不返回记录。
- dbQCrosstab Crosstab — 该查询用与电子表格类似的格式返回数据。
- dbQDelete Delete — 该查询删除指定的行。
- dbQUpdate Update — 该查询更新一些记录。
- dbQAppend Append — 该查询在表或查询的尾部加入新记录。
- dbQMakeTable Make-table — 该查询根据记录集创建一个新表。
- dbQDDL Data-definition — 该查询影响表格及其部分的结构。
- dbQSQLPassThrough Pass-through — 直接把 SQL 语句传递给数据库，没有中间处理。
- dbQSetOperation Union — 该查询创建一个快照型记录集对象，其中包含了一个或多个表中指定记录的数据，所有重复的记录都被删除。如果要包含重复记录，在查询定义的 SQL 语句中加入 ALL 关键字。
- dbQSPTBulk 与 dbQSQLPassThrough 一起使用，指定了一种不返回记录的查询。

注意 在创建一个直接传递 SQL 的查询时，你不需要设置 dbQSQLPassThrough 常量。它是在创建查询定义对象并设置连接属性的时候由 Microsoft Jet 数据库引擎自动设置的。

更多的信息参见 DAO 帮助中的“Type 属性”主题。

m_dateCreated

创建查询定义的日期和时间。如果要直接获得创建查询定义的日期，调用与表相关的 CDaoTableDef 对象的成员函数 GetDateCreated。更多的信

息参见下面的注释。还可以参看 DAO 帮助中的“DateCreated，LastUpdated 属性”主题。

m_dateLastUpdated

最近一次对查询定义作出改变的日期和时间。如果要直接获得最近改变表的日期，调用查询定义的成员函数 GetDateLastUpdated。更多的信息参见下面的注释。同时参看 DAO 帮助中的“DateCreated，LastUpdated 属性”主题。

m_bUpdatable

指明是否可以修改查询定义对象。如果这个属性为 TRUE，查询定义可以被修改；否则不能。可以被修改意味着查询定义对象的查询定义可以被修改。如果查询定义可以被修改，则查询定义对象的 Updatable 属性被设为 TRUE，即使结果记录集不能被修改。如果要直接获得这个属性的值，调用查询定义对象的 CanUpdate 成员函数。更多的信息参见 DAO 帮助中的“Updatable 属性”主题。

m_bReturnsRecords

指明一个直接将 SQL 传递给外部数据库的查询是否返回记录。如果这个属性为 TRUE，该查询返回记录。如果要直接获得这个属性的值，调用 CdaoQuery- Def::GetReturnsRecords。并不是所有直接将 SQL 传递给外部数据库的查询都返回记录。例如，SQL UPDATE 语句更新记录，但不返回语句，而 SQL SELECT 语句却返回语句。更多的信息参见 DAO 帮助

中的“ReturnsRecords属性”主题。

m_strSQL

定义了查询定义对象所执行的查询的SQL语句。SQL属性中包含了SQL语句，决定了执行查询时记录如何被选择，分组或排序。你可以使用该查询把记录选择到一个dynaset类型或snapshot类型的记录集对象。你也可以定义一个批查询来修改数据而不返回记录。你可以调用查询定义的GetSQL成员函数来直接获得这个属性的值。更多的信息参见“Visual C++程序员指南”中的文章“DAO查询”以及DAO帮助中的“SQL属性”主题。

m_strConnect

提供了用于直接传递查询的数据库源的信息。这个信息采用连接字符串的形式。有关连接字符串以及直接获得这个属性值的更多信息参见CDaoDatabase::GetConnect成员函数。

m_nODBCTimeout

当在一个ODBC数据库上运行一个查询时，在产生一个超时错误之前，Microsoft Jet数据库引擎将等待的秒数。当你使用ODBC数据库的时候，例如Microsoft SQL Server，可能会因为网络阻塞或是ODBC服务器的重负而引起延迟。为了避免无限等待，你可以指定Microsoft Jet数据库引擎在产生错误前等待多长时间。缺省的超时值为60秒。你可以调用查询定义的GetODBCTimeout成员函数来直接获得这个属性的值。更多的信

息参见 DAO 帮助中的“ODBCTimeout 属性”主题。

注释

CDaoDatabase::GetQueryDefInfo 成员函数获得的信息被保存在 CDaoQueryDefInfo 结构中。调用数据库对象的 GetQueryDefInfo 函数，查询定义对象就保存在该对象的查询定义集合中。CDaoQueryDefInfo 还为调试版本定义了一个 Dump 成员函数。你可以利用 Dump 来转储 CDaoQueryDefInfo 对象的内容。CDaoDatabase 类也提供了用于直接访问 CDaoQueryDefInfo 对象所返回的所有属性的成员函数，因此你可能很少需要调用 GetQueryDefInfo。

当你在查询定义对象的字段集合或参数集合中加入新的字段或参数时，如果基础数据库不支持新对象所指定的数据类型，就会抛出一个异常。

日期和时间设置是从创建查询定义或最后修改查询定义的计算机上获得的。在多用户环境中，用户应当直接使用 net time 命令从文件服务器上获得这些设置，以避免 DateCreated 和 LastUpdated 属性中的不一致。有关这个结构和其他 MFC DAO Info 结构的信息参见“Visual C++ 程序员指南”中的文章“DAO 集合：获得 DAO 对象的信息”。

请参阅 CDaoQueryDef, CDaoDatabase

CDaoRelationInfo 结构

CDaoRelationInfo 结构具有如下形式：

```
struct CDaoRelationInfo
{
    CDaoRelationInfo( );                        // Constructor
    CString m_strName;                          // Primary
    CString m_strTable;                         // Primary
    CString m_strForeignTable;                  // Primary
    long m_lAttributes;                         // Secondary
    CDaoRelationFieldInfo* m_pFieldInfos;       // Secondary
    short m_nFields;                            // Secondary
    // Below the // Implementation comment:
    // Destructor, not otherwise documented
};
```

CDaoRelationInfo 结构中包含了在 CDaoDatabase 对象的两个表之间定义的关系的信息。上面对 Primary 和 Secondary 的引用指明了 CDaoDatabase 类的成员函数 GetRelationInfo 是如何返回信息的。

MFC 类不代表关系对象。相反，DAO 对象构成了 CDaoDatabase 类的 MFC 对象，它维护着一个关系对象的集合：CDaoDatabase 提供了用于访问关系信息中一些单独项的成员函数，也可以调用包容数据库对象的 GetRelationInfo 成员函

数，通过CDaoRelationInfo结构同时访问所有的信息。

成员

m_strName
: 关系对象的唯一名字。更多的信息参见DAO帮助中的“Name属性”主题。

m_strTable
: 命名关系中的主表。

m_strForeignTable
: 命名关系中的外部表。外部表是用于包含外部关键字的表。通常，可以利用外部表来建立或强化引用完整性。外部表通常位于一对多关系中多的一方。外部表的例子有包含以下内容的表：美国州代码或加拿大的省或顾客的订货。

m_lAttributes
: 包含了有关关系类型的信息。这个成员的值可以是下列值中的任何一个：

- dbRelationUnique　一对一关系。
- dbRelationDontEnforce　关系是非强迫的的。（没有引用完整性）
- dbRelationInherited　关系存在于非当前数据库中，其中包含了两个相连的表。

- dbRelationLeft 关系是左连接的。左外部连接包含了两个表中第一个（左边）表的所有记录，即使在第二个表（右边）中没有与之匹配的值。
- dbRelationRight 关系是右连接的。右的外部连接包含了两个表中第二个（右边）表的所有记录，即使在第一个表（左边）中没有与之匹配的值。
- dbRelationUpdateCascade 更新操作是重叠的。
- dbRelationDeleteCascade 删除操作是重叠的。

m_pFieldInfos

指向 CDaoRelationFieldInfo 结构数组的指针。对于关系中的每个字段，该数组中都包含一个对象。m_nFields 数据成员给出了数组元素的个数。

m_nFields

m_pFieldInfos 数据成员中 CDaoRelationFieldInfo 对象的个数。

注释

通过 CDaoDatabase::GetRelationInfo 成员函数获取的信息被保存在 CDaoRelationInfo 结构中。CDaoRelationInfo 还在调试版本中定义了一个 Dump 成员函数。你可以使用 Dump 来转储 CDaoRelationInfo 对象的内容。有关这个结构以及其他 MFC DAO Info 结构的信息参见“Visual C++ 程序员指南”中的文章“DAO 集合：获取 DAO 对象的信息”。

请参阅 CDaoRelationFieldInfo

CDaoRelationFieldInfo 结构

CDaoRelationFieldInfo 结构具有如下形式：

```
struct CDaoRelationFieldInfo
{
      CString m_strName;                 // Primary
      CString m_strForeignName;          // Primary
};
```

CDaoRelationFieldInfo 结构中包含了为数据访问对象（DAO）定义的关系中的字段的信息。DAO 关系对象指定了定义关系的主表中的字段和外部表中的字段。在上面的结构定义中对 Primary 的引用指明了调用 CDaoDatabase 类的成员函数 GetRelationInfo 获得的信息是如何在 CDaoRelationInfo 对象的 m_pFieldInfos 成员中返回的。

MFC 类不代表任何关系对象和关系字段对象。相反，DAO 对象构成了 CDaoDatabase 类的 MFC 对象，其中包含了关系对象的集合，称为关系集合。每个关系对象包含了一个关系对象的集合。每个关系对象使主表中的字段和外部表中的字段产生联系。合在一起，关系字段对象定义了每个表中的一组字段，它们一起定义了关系。CDaoDatabase 使你能够调用 GetRelationInfo 成员函数通

过 CDaoRelationInfo 对象来访问关系对象。CdaoRelation- Info 对象具有一个数据成员 m_pFieldInfos，它指向一个 CDaoRelationFieldInfo 对象的数组。

成员

m_strName

关系的主表中字段的名字。

m_strForeignName

关系的外部表中字段的名字。

注释

调用包容的 CDaoDatabase 对象的 GetRelationInfo 成员函数，你感兴趣的关系对象就保存在该对象的关系集合中。然后访问 CDaoRelationInfo 对象的 m_pFieldInfos 成员。CDaoRelationFieldInfo 还在调试版本中定义了一个 Dump 函数。你可以使用 Dump 来转储 CDaoRelationFieldInfo 对象的内容。

有关这个结构以及其他 MFC DAO Info 结构的信息参见“Visual C++ 程序员指南”中的文章“DAO 集合：获得 DAO 对象的信息”。

请参阅 CDaoRelationInfo

CDaoTableDefInfo 结构

CDaoTableDefInfo 结构具有如下形式：

```
struct CDaoTableDefInfo
{
    CString m_strName;                  // Primary
    BOOL m_bUpdatable;                  // Primary
    long m_lAttributes;                 // Primary
    COleDateTime m_dateCreated;         // Secondary
    COleDateTime m_dateLastUpdated;     // Secondary
    CString m_strSrcTableName;          // Secondary
    CString m_strConnect;               // Secondary
    CString m_strValidationRule;        // All
    CString m_strValidationText;        // All
    long m_lRecordCount;                // All
};
```

CDaoTableDefInfo 结构中包含了为数据访问对象（DAO）定义的表定义对象的信息。表定义是 CDaoTableDef 类的对象。上面对 Primary，Secondary 和 All 的引用指明了在 CDaoDatabase 类的成员函数 GetTableDefInfo 中信息是如何返回的。

成员

m_strName

表定义对象的唯一的名字。如果要直接获得这个属性的值，调用表定义对象的 GetName 成员函数。更多的信息参见 DAO 帮助中的“Name 属性”主题。

m_bUpdatable

指明是否可以修改表。确定一个表可以修改的最快的方式是否为这个表打开一个 CDaoTableDef 对象，然后调用这个对象的 CanUpdate 成员函数。对于一个新创建的表定义对象，CanUpdate 总是返回非零值（TRUE），对于连接的表定义对象则返回 0（FALSE）。一个新创建的表定义对象仅能被附加到一个用户具有写权限的数据库。如果该表仅包含不可更新的字段，CanUpdate 返回 0。当有一个或多个字段可以被更新时，CanUpdate 返回非零值。你只能编辑可更新的字段。更多的信息参见 DAO 帮助中的“Updatable 属性”主题。

m_lAttributes

指定了表定义对象所代表的表的特征。如果要获得表定义的当前属性，调用它的成员函数 GetAttributes。返回值可能是下面这些值的组合（使用位或操作符|）：

- dbAttachExclusive　对于使用 Microsoft Jet 数据库引擎的数据库，指明

这个表是一个独占使用的表。

- dbAttachSavePWD 对于使用 Microsoft Jet 数据库引擎的数据库，指明连接的表的用户 ID 和密码被保存在连接信息中。
- dbSystemObject 指明这个表是由 Microsoft Jet 数据库引擎提供的系统表（只读）。
- dbHiddenObject 指明这个表是由 Microsoft Jet 数据库引擎提供的隐含表（只读）。
- dbAttachedTable 指明这个表是非 ODBC 数据库，例如 Paradox 数据库的一个连接表。
- dbAttachedODBC 指明这个表是 ODBC 数据库，例如 Microsoft SQL Server 数据库的一个连接表。

m_dateCreated

创建表的日期和时间。如果要直接获得创建表的日期，调用与该表相关的 CDaoTableDef 对象的 GetDateCreated 成员函数。更多的信息参见下面的注释。相关信息参见 DAO 帮助中的“DateCreated，LastUpdated 属性”主题。

m_dateLastUpdated

最近对表的设计作出改变的日期和时间。如果要直接获取该表最近被修改的日期，调用与该表相关的 CDaoTableDef 对象的 GetDateLastUpdated 成员函数。更多的信息参见下面的注释。相关的信息参见 DAO 帮助中的“DateCreated，LastUpdated 属性”主题。

m_strSrcTableName

指定了连接表的名字，如果有的话。如果要直接获得源表的名字，调用与表相关的CDaoTableDef对象的GetSourceTableName成员函数。

m_strConnect

提供了与打开的数据库源有关的信息。你可以调用CDaoTableDef对象的成员函数GetConnect来检查这个属性。有关连接字符串的更多信息参见GetConnect。

m_strValidationRule

当表定义字段中的数据被改变或加入一个表中时，这个值被用来校验数据。校验仅支持使用Microsoft Jet数据库引擎的数据库。如果要直接获得校验规则，调用与该表相关的CDaoTableDef对象的GetValidationRule成员函数。相关的信息参见DAO帮助中的“Validation属性”主题。

m_strValidationText

指定了当ValidationRule属性所指定的校验规则不被满足时，应用程序要显示的消息文本。相关的消息参见DAO帮助中的“ValidationText属性”主题。

m_lRecordCount

在表定义对象中访问的记录数目。这个属性是只读的。如果要直接获取记录的计数，调用CDaoTableDef对象的GetRecordCount成员函数。GetRecordCount的文档进一步描述了记录计数。注意如果表中有很多记

录，要获得这个计数可能耗去很多时间。

注释

CDaoDatabase::GetTableDefInfo 成员函数获得的信息被保存在 CDaoTableDefInfo 结构中。调用 CDaoDatabase 结构的 GetTableDefInfo 成员函数，表定义对象就保存在的 TableDefs 集合中。CDaoTableDefInfo 还在调试版本中定义了一个 Dump 成员函数。你可以使用 Dump 来转储 CDaoTableDefInfo 对象的内容。

日期和时间设置是从基表被创建或修改的计算机上获得的。在多用户环境中，用户应该直接从文件服务器获得这些设置以避免 DateCreated 和 LastUpdated 属性产生不一致。

有关这个结构和其他 MFC DAO Info 结构的信息参见“Visual C++ 程序员指南”中的文章“DAO 集合：获得 DAO 对象的信息”。

请参阅 CDaoTableDef, CDaoDatabase, CDaoTableDef::CanUpdate, CDaoTableDef::GetAttributes, CDaoTableDef::GetDateCreated, CDaoTableDef::GetDateLastUpdated, CDaoTableDef::GetRecordCount, CDaoTableDef::GetSourceTableName, CDaoTableDef::GetValidationRule, CDaoTableDef::GetValidationText

CDaoWorkspaceInfo 结构

CDaoWorkspaceInfo 结构具有如下形式：

```
struct CDaoWorkspaceInfo
{
    CString m_strName;              // Primary
    CString m_strUserName;          // Secondary
    BOOL m_bIsolateODBCTrans;       // All
};
```

CDaoWorkspaceInfo 结构中包含了为使用数据访问对象（DAO）的数据库访问而定义的工作空间的信息。工作空间是 CDaoWorkspace 类的对象。上面对 Primary，Secondary 和 All 的引用指明了 CDaoWorkspace 类的 GetWorkspaceInfo 成员函数是如何返回信息的。

成员

m_strName

工作空间的唯一名字。如果要直接获得这个属性值，调用查询定义对象的 GetName 成员函数。更多的信息参见 DAO 帮助中的“Name 属性”主题。

m_strUserName

代表了工作空间对象的拥有者的值。相关的信息参见 DAO 帮助中的“UserName 属性”主题。

m_bIsolateODBCTrans
指明了涉及同一个 ODBC 数据库的多用户事务是否被隔离。更多的信息参见 CDaoWorkspace::SetIsolateODBCTrans。相关的信息参见 DAO 帮助中的“IsolateODBCTrans 属性”主题。

注释

CDaoWorkspace::GetWorkspaceInfo 成员函数获取的信息被保存在 CDaoWorkspaceInfo 结构中。CDaoWorkspaceInfo 还在调试版本中定义了一个 Dump 成员函数。你可以使用 Dump 来转储 CdaoWorkspaceInfo 对象的内容。有关这个对象和其他 MFC DAO Info 结构的信息参见“Visual C++ 程序员指南”中的文章“DAO 集合：获得 DAO 对象的信息”。

请参阅 CDaoWorkspace

CODBCFieldInfo 结构

CODBCFieldInfo 结构具有如下形式：

```
struct CODBCFieldInfo
{
```

```
    CString m_strName;
    SWORD m_nSQLType;
    UDWORD m_nPrecision;
    SWORD m_nScale;
    SWORD m_nNullability;
};
```

CODBCFieldInfo 结构中包含了 ODBC 数据源中字段的有关信息。如果要获得这个信息，调用 CRecordset::GetODBCFieldInfo。

成员

m_strName

字段的名字。

m_nSQLType

字段的 SQL 数据类型。它可以是 ODBC SQL 数据类型或驱动程序特定的 SQL 数据类型。有效的 ODBC SQL 数据类型的列表参见“ODBC SDK 程序员参考”的附录 D 中的“SQL 数据类型”。有关驱动程序特定的 SQL 数据类型的信息参看驱动程序的文档。

m_nPrecision

字段的最大精度。有关的细节参见“ODBC SDK 程序员参考”的附录 D 中的“精度，范围，长度和显示大小”。

m_nScale

字段的范围。有关的细节参见 ODBC SDK 程序员参考的附录 D 中的“精度，范围，长度和显示大小”。

m_nNullability

一个字段是否允许 Null 值。它可以是两个值之一：如果该字段允许 Null 值，则为 SQL_NULLABLE；如果该字段不允许 Null 值，则为 SQL_NO_NULLS。

请参阅 CRecordset::GetODBCFieldInfo, CRecordset::GetFieldValue

COLORADJUSTMENT 结构

COLORADJUSTMENT **结构具有如下形式：**

```
typedef struct  tagCOLORADJUSTMENT {      /* ca */
    WORD  caSize;
    WORD  caFlags;
    WORD  caIlluminantIndex;
    WORD  caRedGamma;
    WORD  caGreenGamma;
    WORD  caBlueGamma;
    WORD  caReferenceBlack;
```

```
    WORD   caReferenceWhite;
    SHORT caContrast;
    SHORT caBrightness;
    SHORT caColorfulness;
    SHORT caRedGreenTint;
} COLORADJUSTMENT;
```

COLORADJUSTMENT结构定义了当StretchBlt模式为HALFTONE时，Windows的StretchBlt和StretchDIBits函数使用的颜色调整值。

成员

caSize

指定了该结构以字节为单位的大小。

caFlags

指定了如何准备输出图象。这个成员可以被设为NULL或是下列值的组合：

- CA_NEGATIVE　指定了要显示原图的反相图。
- CA_LOG_FILTER　指定了一个对数函数，将对输出颜色的最后深度起作用。当亮度较低时，这将增加颜色的对比度。

caIlluminantIndex

指定了光源的亮度，图象将在该光源下显示。这个成员可以被设为下列值中的一个：

- ILLUMINANT_EQUAL_ENERGY
- ILLUMINANT_A
- ILLUMINANT_B
- ILLUMINANT_C
- ILLUMINANT_D50
- ILLUMINANT_D55
- ILLUMINANT_D65
- ILLUMINANT_D75
- ILLUMINANT_F2
- ILLUMINANT_TURNGSTEN
- ILLUMINANT_DAYLIGHT
- ILLUMINANT_FLUORESCENT
- ILLUMINANT_NTSC

caRedGamma

指定了原色中红原色的 n **次** gamma **修正值。这个值必须介于** 2500～65000 **之间。如果值等于** 10000**，则意味着没有** gamma **修正。**

caGreenGamma

指定了原色中绿原色的 n **次** gamma **修正值。这个值必须介于** 2500～65000

之间。如果值等于10000，则意味着没有gamma修正。

caBlueGamma

指定了原色中蓝原色的n次gamma修正值。这个值必须介于2500～65000之间。如果值等于10000，则意味着没有gamma修正。

caReferenceBlack

指定了源色的黑色参考值。任何比这个值暗的颜色被当作黑色。这个值必须介于0～4000之间。

caReferenceWhite

指定了源色的白色参考值。任何比这个值亮的颜色被当作白色。这个值必须介于6000～10000之间。

caContrast

指定了对源对象应用的对比度值。这个值必须介于－100～100 之间。如果该值为0则意味着没有对比度调整。

caBrightness

指定了对源对象应用的亮度值。这个值必须介于－100～100 之间。如果该值为0，则意味着没有亮度调整。

caColorfulness

指定了对源对象应用的饱和度值。这个值必须介于－100～100 之间。如果该值为0，则意味着没有饱和度调整。

caRedGreenTint

指定了应用于源对象的红或绿校正值。这个值必须介于－100～100之间。正数将会向红色调整，而负数将会向绿色调整。如果这个值为 0，则意味着没有调整。

请参阅 CDC::GetColorAdjustment

COMPAREITEMSTRUCT结构

COMPAREITEMSTRUCT数据结构具有这种形式：

```
typedef struct tagCOMPAREITEMSTRUCT {
    UINT      CtlType;
    UINT      CtlID;
    HWND      hwndItem;
    UINT      itemID1;
    DWORD     itemData1;
    UINT      itemID2;
    DWORD     itemData2;
} COMPAREITEMSTRUCT;
```

COMPAREITEMSTRUCT结构为有序的自画列表框或组合框中的两项提供了标识符和来自应用程序的数据。当应用程序在用 CBS_SORT 或 LBS_SORT 风格

创建的自画列表框或组合框中加入一个新项时，Windows 将给拥有者发送一个 WM_COMPAREITEM 消息。这个消息的 lParam 参数中包含了指向 COMPAREITEMSTRUCT 结构的长指针。通过接收这个消息，拥有者比较这个两个项并返回一个值，指明哪一项在前。

成员

CtlType

ODT_LISTBOX（这指定了一个自画列表框）或 ODT_COMBOBOX（这指定了一个自画组合框）。

CtlID

列表框或组合框的控制 ID。

hwndItem

控件的窗口句柄。

itemID1

将要比较的列表框或组合框中第一项的索引。

itemData1

应用程序为要比较的第一项提供的数据。这个值在把这个项加入组合框或列表框时给定。

itemID2

将要比较的列表框或组合框中第二项的索引。

itemData2

应用程序为要比较的第二项提供的数据。这个值在把这个项加入组合框或列表框时给定。

请参阅 CWnd::OnCompareItem

CREATESTRUCT 结构

CREATESTRUCT 结构具有如下形式：

```
typedef struct tagCREATESTRUCT {
    LPVOID      lpCreateParams;
    HANDLE      hInstance;
    HMENU       hMenu;
    HWND        hwndParent;
    int         cy;
    int         cx;
    int         y;
    int         x;
    LONG        style;
    LPCSTR      lpszName;
```

```
    LPCSTR      lpszClass;
    DWORD       dwExStyle;
} CREATESTRUCT;
```

CREATESTRUCT结构定义了传递给应用程序的窗口过程的初始化参数。

成员

lpCreateParams

指向将被用于创建窗口的数据的指针。

hInstance

标识了拥有新窗口的模块的模块实例的句柄。

hMenu

标识了要被用于新窗口的菜单。如果是子窗口，则包含整数ID。

hwndParent

标识了拥有新窗口的窗口。如果新窗口是一个顶层窗口，这个参数可以为NULL。

cy

指定了新窗口的高。

cx

指定了新窗口的宽。

y

指定了新窗口的左上角的 y 轴坐标。如果新窗口是一个子窗口，则坐标是相对于父窗口的；否则坐标是相对于屏幕原点的。

x

指定了新窗口的左上角的 x 轴坐标。如果新窗口是一个子窗口，则坐标是相对于父窗口的；否则坐标是相对于屏幕原点的。

style

指定了新窗口的风格。

lpszName

指向一个以 null 结尾的字符串，指定了新窗口的名字。

lpszClass

指向一个以 null 结尾的字符串，指定了新窗口的 Windows 类名（一个 WNDCLASS 结构；更多的信息参见 Win32 SDK 文档）。

dwExStyle

指定了新窗口的扩展风格。

请参阅 CWnd::OnCreate

DELETEITEMSTRUCT结构

DELETEITEMSTRUCT结构具有如下形式：

```
typedef struct tagDELETEITEMSTRUCT { /* ditms */
    UINT CtlType;
    UINT CtlID;
    UINT itemID;
    HWND hwndItem;
    UINT itemData;
} DELETEITEMSTRUCT;
```

DELETEITEMSTRUCT结构指定一个被删除的自画列表框或组合框项。当从列表框或组合框中删除一项或当列表框或组合框被销毁时，对于被删除的每一项，Windows向它的拥有者发送WM_DELETEITEM消息。消息的lParam参数包含指向这个结构的指针。

成员

CtlType

是ODT_LISTBOX（对于自画列表框）或ODT_COMBOBOX（对于自画组合框）。

CtlID

指定列表框或组合框的标识符。

itemID

指定将被删除的项在列表框或组合框中的索引。

hwndItem

控件的标识符。

itemData

指定程序定义的与该项相关的数据。此值通过消息的 lParam 参数传递给控件，而这一消息向列表框或组合框中增加项。

请参阅 CWnd::OnDeleteItem

DEVMODE 结构

DEVMODE 结构具有如下形式：

```
typedef struct _devicemode {      /* dvmd */
    TCHAR   dmDeviceName[32];
    WORD    dmSpecVersion;
    WORD    dmDriverVersion;
    WORD    dmSize;
    WORD    dmDriverExtra;
    DWORD   dmFields;
```

```
    short   dmOrientation;
    short   dmPaperSize;
    short   dmPaperLength;
    short   dmPaperWidth;
    short   dmScale;
    short   dmCopies;
    short   dmDefaultSource;
    short   dmPrintQuality;
    short   dmColor;
    short   dmDuplex;
    short   dmYResolution;
    short   dmTTOption;
    short   dmCollate;
    TCHAR   dmFormName[32];
    WORD    dmUnusedPadding;
    USHORT dmBitsPerPel;
    DWORD   dmPelsWidth;
    DWORD   dmPelsHeight;
    DWORD   dmDisplayFlags;
    DWORD   dmDisplayFrequency;
} DEVMODE;
```

DEVMODE数据结构中包含了有关设备初始化和打印机环境的信息。

成员

dmDeviceName

　　指定了驱动程序支持的设备名称；例如，对于 PCL/HP LaserJet®，会是 PCL/HP 激光打印机。这个字符串在设备驱动程序之间是相互不同的。

dmSpecVersion

　　指定了初始化数据的版本数字，这个结构就基于这些数据。

dmDriverVersion

　　指定了打印机驱动程序开发商分配的打印机驱动程序版本号。

dmSize

　　指定了 DEVMODE 结构的大小，以字节为单位，不包括 dmDriverData（与设备有关）成员。如果应用程序仅操作数据中与驱动程序无关的部分，它就可以使用这个成员以确定该结构的长度，而并不需要有不同版本的记录。

dmDriverExtra

　　包含了这个结构后面的私有驱动程序数据的数目，以字节为单位。如果设备驱动程序不使用该设备独有的信息，就把这个成员设为零。

dmFields

指定了 DEVMODE 结构的其余成员中哪些已被初始化。第 0 位（定义为 DM）ORIENTATION）代表 dmOrientation，第 1 位（定义为 DM_PAPERSIZE）代表 dmPaperSize 等等。打印机驱动出现仅支持那些适合打印技术的成员。

dmOrientation

选择纸的方向。这个成员可以为 DMORIENT_PORTRAIT（1）或 DMORIENT_ LANDSCAPE（2）。

dmPaperSize

选择将用于打印的纸张大小。如果纸张的长度和宽度都用 dmPaperLength 和 dmPaperWidth 成员来设置的话，这个成员可以被设为 0。否则，dwPaperSize 成员可以被设为以下值之一：

DMPAPER_LETTER letter，8 1/2 ×11 英寸
MPAPER_LEGAL Legal，8 1/2 × 14 英寸
DMPAPER_A4 A4 letter，210 × 297 毫米
DMPAPER_CSHEET C letter，17 × 22 英寸
DMPAPER_DSHEET D letter，22 × 34 英寸
DMPAPER_ESHEET E letter，34 × 44 英寸
DMPAPER_LETTERSMALL letter small，8 1/2 × 11 英寸
DMPAPER_TABLOID Tabloid，11 × 17 英寸
DMPAPER_LEDGER Ledger，17 × 11 英寸

DMPAPER_STATEMENT Statement, 5 1/2 × 8 1/2 英寸
DMPAPER_EXECUTIVE Executive, 7 1/4 × 10 1/2 英寸
DMPAPER_A3 A3 Sheet, 297 × 420 毫米
DMPAPER_A4SMALL A4 Small Sheet, 210 × 297 毫米
DMPAPER_A5 A5 Sheet, 148 × 210 毫米
DMPAPER_B4 B4 Sheet, 250 × 354 毫米
DMPAPER_B5 B5 Sheet, 182 × 257 毫米
DMPAPER_FOLIO Folio, 8 1/2 × 13 英寸
DMPAPER_QUARTO Quarto, 215 × 275 毫米
DMPAPER_10X14 10 × 14 英寸
DMPAPER_11X17 11 × 17 英寸
DMPAPER_NOTE Note, 8 1/2 × 11 英寸
DMPAPER_ENV_9 #9 EnVelope, 3 7/8 × 8 7/8 英寸
DMPAPER_ENV_10 #10 EnVelope, 4 1/8 × 9 1/2 英寸
DMPAPER_ENV_11 #11 EnVelope, 4 1/2 × 10 3/8 英寸
DMPAPER_ENV_12 #12 EnVelope, 4 3/4 × 11 英寸
DMPAPER_ENV_14 #14 EnVelope, 5 × 11 1/2 英寸
DMPAPER_ENV_DL DL EnVelope, 110 × 220 毫米
DMPAPER_ENV_C5 C5 EnVelope, 162 × 229 毫米
DMPAPER_ENV_C3 C3 EnVelope, 324 × 458 毫米
DMPAPER_ENV_C4 C4 EnVelope, 229 × 324 毫米

DMPAPER_ENV_C6 C6 EnVelope， 114 × 162 毫米
DMPAPER_ENV_C65 C65 EnVelope， 114 × 229 毫米
DMPAPER_ENV_B4 B4 EnVelope， 250 × 353 毫米
DMPAPER_ENV_B5 B5 EnVelope， 176 × 250 毫米
DMPAPER_ENV_B6 B6 EnVelope， 176 × 125 毫米
DMPAPER_ENV_ITALY Italy EnVelope， 110 × 230 毫米
DMPAPER_ENV_MONARCH Monarch EnVelope， 3 7/8 × 7 1/2 英寸
DMPAPER_ENV_PERSONAL 6 3/4 EnVelope， 3 5/8 × 6 1/2 英寸
DMPAPER_FANFOLD_US US Std Fanfold, 14 7/8 × 11 英寸
DMPAPER_FANFOLD_STD_GERMAN German Std Fanfold, 8 1/2 × 12 英寸
DMPA PER_FANFOLD_LGL_GERMAN German Legal Fanfold, 8 1/2 × 13 英寸

dmPaperLength

重定义由 dmPaperSize 成员指定的纸张长度，可用于自定义纸张大小，也可以用于点阵打印机，这种打印机能打出任意长度的纸张。这些值与这个结构中其他指定物理长度的值都是以 0.1 毫米为单位的。

dmPaperWidth

重载由 dmPaperSize 成员指定的纸张宽度。

dmScale

指定了打印输出的缩放因子。实际的页面大小为物理纸张的大小乘以

dmScale/100。例如，对于信纸大小的纸张，如果 dmScale 的值为 50，它将可以容纳相当于 17×22 英寸页面的内容，因为输出的文本和图形的宽、高都将是原始大小的一半。

dmCopies

如果设备支持多页拷贝，则选择了要打印的拷贝数目。

dmDefaultSource

保留，必须为 0。

dmPrintQuality

指定了打印机的分辨率。有四种预定义的与设备无关的值：

DMRES_HIGH
DMRES_MEDIUM
DMRES_LOW
DMRES_DRAFT

如果给定了一个正值，它就指定了每英寸打印的点数（DPI），因此是与设备有关的。

dmColor

对于彩色打印机，在彩色和单色之间切换。下面是可能的取值：

- DMCOLOR_COLOR
- DMCOLOR_MONOCHROME

dmDuplex

为支持双面打印的打印机选择双面打印方式。可能的取值如下：

- DMDUP_SIMPLEX
- DMDUP_HORIZONTAL
- DMDUP_VERTICAL

dmYResolution

指定了打印机在 y 方向的分辨率，以每英寸的点数为单位。如果打印机对该成员进行了初始化，dmPrintQuality 成员指定了打印机在 x 方向的分辨率，以每英寸点数为单位。

dmTTOption

指明如何打印 TrueType 字体。这个成员可以取如下值之一：

- DMTT_BITMAP 把 TrueType 字体作为图形打印。这是点阵打印机的缺省动作。
- DMTT_DOWNLOAD 将 TrueType 字体作为软字体下载。这是使用打印机控制语言（PCL）的惠普打印机的缺省动作。
- DMTT_SUBDEV 用 TrueType 字体替换设备字体。这是 PostScript® 打印机的缺省动作。

dmCollate

指定在打印多份拷贝的时候是否使用校对。使用 DMCOLLATE_FALSE

后能够得到更快更有效的输出，因为不管要打印多少份拷贝，只向打印机传送一次数据。打印机仅被通知再打印一页。这个成员可以是下列值之一：

- DMCOLLATE_TRUE 当打印多份拷贝时进行校对。
- DMCOLLATE_FALSE 当打印多份拷贝时不进行校对。

dmFormName

指定了要使用的格式名字。例如，Letter 或 Legal。这些名字的完整集合可以通过 Windows 的 EnumForms 函数获得。

dmUnusedPadding

用于将结构对齐到 DWORD 边界。不能使用或引用这个成员。它的名字和用法是保留的，在以后的版本中可能会变化。

dmBitsPerPel

指定了显示设备的颜色分辨率，以像素的位数为单位。例如，16 色使用 4 位，256 色使用 8 位，而 65536 色使用 16 位。

dmPelsWidth

指定了可见设备表面的以像素为单位的宽度。

dmPelsHeight

指定了可见设备表面的以像素为单位的高度。

dmDisplayFlags

指定了设备的显示模式。下面是有效的标志：

- DM_GRAYSCALE 指定显示使用了无颜色设备。如果没有设置这个标志，就假定彩色模式。
- DM_INTERLACED 指定了隔行显示模式。如果没有设置这个标志，就假定非隔行模式。

dmDisplayFrequency

指定了显示设备的特定模式所使用的以赫兹为单位的频率（每秒的周期数）。

注释

在 dmDisplayMode 成员后面将是一些设备驱动程序的私有数据。以字节为单位的私有数据数目是由 dmDriverExtra 成员指定的。

请参阅 CDC::ResetDC, CPrintDialog::GetDevMode

DEVNAMES 结构

DEVNAMES 结构具有如下形式：

```
typedef struct tagDEVNAMES { /* dvnm */
    WORD wDriverOffset;
    WORD wDeviceOffset;
```

```
    WORD wOutputOffset;
    WORD wDefault;
    /* wDefault 后面是驱动程序，设备和端口名字符串 */
} DEVNAMES;
```

DEVNAMES 结构中包含的字符串指定了打印机的驱动程序，设备和输出端口的名字。PrintDlg 函数使用这些字符串来初始化系统定义的 Print 对话框中的成员。当用户关闭对话框时，将在这个结构中返回有关选中的打印机的信息。

成员

wDriverOffset

（输入/输出）指定了一个以 null 结尾的字符串的偏移，其中包含了设备驱动程序的文件名（没有扩展名）。对于输入，这个字符串将被用于确定要显示在对话框中的打印机。

wDeviceOffset

（输入/输出）指定了以 null 结尾的字符串（包括 null 在内，最多有 32 个字节），其中包含了设备的名字。这个字符串必须与 DEVMODE 结构中的 dmDeviceName 成员相同。

wOutputOffset

（输入/输出）指定了以 null 结尾的字符串，其中包含了物理输出设备（输出端口）的 DOS 设备名。这个字符串必须与 DEVMODE 结构中的

dmDeviceName 成员相同。

wDefault

指定 DEVNAMES 结构中包含的字符串是否表示了缺省打印机。这个字符串被用来检验自最近一次打印操作以来，缺省打印机是否发生变化。对于输入操作，如果设置了 DN_DEFAULTPRN 标志，DEVNAMES 结构中的其它值将被检验，与当前的缺省打印机作比较。如果有字符串不匹配，就会显示一个警告信息，通知用户可能需要重新格式化文档。对于输出操作，只有当显示了 Print Setup 对话框并且用户选择了 OK 按钮时，wDefault 成员才会发生改变。如果选择了缺省的打印机，就会设置 DN_DEFAULTPRN 标志。如果选择了一个其它打印机，则不会设置这个标志。这个成员中其它所有的位都是保留的，被 Print 对话框过程内部使用。

请参阅 CPrintDialog::CreatePrinterDC

DRAWITEMSTRUCT 结构

DRAWITEMSTRUCT 结构具有如下形式：

```
typedef struct tagDRAWITEMSTRUCT {
    UINT      CtlType;
    UINT      CtlID;
```

```
    UINT      itemID;
    UINT      itemAction;
    UINT      itemState;
    HWND      hwndItem;
    HDC       hDC;
    RECT      rcItem;
    DWORD     itemData;
} DRAWITEMSTRUCT;
```

DRAWITEMSTRUCT 结构提供了拥有者窗口在决定如何画出自画控件或菜单项时必须得到的信息。自画控件或菜单项的拥有者窗口在 WM_DRAWITEM 消息的 lParam 参数中接收到一个指向这个结构的指针。

成员

CtlType

控件类型。控件类型的取值如下：

- ODT_BUTTON 自画按钮
- ODT_COMBOBOX 自画组合框
- ODT_LISTBOX 自画列表框
- ODT_MENU 自画菜单
- ODT_LISTVIEW 列表视控件

- ODT_STATIC 自画 Static 控件
- ODT_TAB Tab 控件

CtlID

组合框，列表框或按钮的控制 ID。对菜单不使用这个成员。

itemID

菜单的菜单项 ID 或是组合框或列表框中的项的索引。对于空的列表框或组合框，这个成员是一个负值，这允许应用程序只在 rcItem 成员指定的位置画出焦点矩形，既使控件中没有项。这样用户就可以知道列表框或组合框是否具有输入焦点。itemAction 成员中位的设置决定了该矩形是否应当画得就象列表框或组合框拥有输入焦点那样。

itemAction

定义了要求的绘图动作。它可以是下面位中的一个或多个：

- ODA_DRAWENTIRE 当需要画出整个控件时设置该位。
- ODA_FOCUS 当控件获得或失去输入焦点时设置该位。如果要确定控件是否拥有输入焦点，应该检查 itemState 成员。
- ODA_SELECT 当选择项发生变化时设置该位。如果要确定新的选择状态，应该检查 itemState 成员。

itemState

指定了完成当前绘图动作后项的可视状态。如果要使菜单项无效，则会设置 ODS_GRAYED 标志。状态标志如下：

- ODS_CHECKED 如果要标记菜单项则设置该位。仅对菜单使用。
- ODS_DISABLED 如果要把该项画成禁止状态则设置该位。
- ODS_FOCUS 如果该项拥有输入焦点则设置该位。
- ODS_GRAYED 如果要使该项变灰则设置该位。仅对菜单使用。
- ODS_SELECTED 如果该项被选中则设置该位。
- ODS_COMBOBOXEDIT 绘图发生在自画组合框控件的选择区域(编辑控件）。
- ODS_DEFAULT 该项为缺省项。

hwndItem

指定了组合框，列表框和按钮控件的窗口句柄。指定了包含菜单项的菜单的句柄（HMENU）。

hDC

标识了一个设备环境。在控件上进行绘图操作时必须使用这个设备环境。

rcItem

hDC 成员指定的设备环境中的矩形，定义了将要画出的控件的边界。Windows 自动将画出的任何东西裁剪在组合框，列表框和按钮的设备环境之内,但是它不裁剪菜单项。在画出菜单项的时候,拥有者不能在 rcItem 成员所定义的矩形之外绘图。

itemData

对于组合框或列表框，这个成员包含了下面的函数传递给列表框的值：

- CComboBox::AddString
- CComboBox::InsertString
- CListBox::AddString
- CListBox::InsertString

对于菜单，这个成员包含了下面的函数传递给菜单的值：

- CMenu::AppendMenu
- CMenu::InsertMenu
- CMenu::ModifyMenu

请参阅 CWnd::OnDrawItem

EXTENSION_CONTROL_BLOCK 结构

EXTENSION_CONTROL_BLOCK **结构具有如下形式：**

```
typedef struct _EXTENSION_CONTROL_BLOCK {
    DWORD     cbSize;                                    //IN
    DWORD     dwVersion                                  //IN
    HCONN     ConnID;                                    //IN
    DWORD     dwHttpStatusCode;                          //OUT
    CHAR      lpszLogData[HSE_LOG_BUFFER_LEN];           //OUT
    LPSTR     lpszMethod;                                //IN
```

```
LPSTR     lpszQueryString;                    //IN
LPSTR     lpszPathInfo;                       //IN
LPSTR     lpszPathTranslated;                 //IN
DWORD     cbTotalBytes;                       //IN
DWORD     cbAvailable;                        //IN
LPBYTE    lpbData;                            //IN
LPSTR     lpszContentType;                    //IN
BOOL ( WINAPI * GetServerVariable )
    ( HCONN       hConn,
      LPSTR       lpszVariableName,
      LPVOID      lpvBuffer,
      LPDWORD     lpdwSize );
BOOL ( WINAPI * WriteClient )
    ( HCONN       ConnID,
    LPVOID      Buffer,
    LPDWORD     lpdwBytes,
    DWORD       dwReserved );
BOOL ( WINAPI * ReadClient )
    ( HCONN       ConnID,
    LPVOID      lpvBuffer,
    LPDWORD     lpdwSize );
```

```
    BOOL ( WINAPI * ServerSupportFunction )
        ( HCONN          hConn,
        DWORD          dwHSERRequest,
        LPVOID         lpvBuffer,
        LPDWORD        lpdwSize,
        LPDWORD        lpdwDataType );
} EXTENSION_CONTROL_BLOCK, *LPEXTENSION_CONTROL_BLOCK;
```

服务器通过 EXTENSION_CONTROL_BLOCK.与 ISA 通讯。

上面对 IN 和 OUT 的引用指明该成员是适用于发送到扩展的消息（IN）还是来自扩展的消息（OUT）。

成员

EXTENSION_CONTROL_BLOCK 结构包括如下的域：

cbSize

结构的大小。

dwVersion

HTTP_FILTER_REVISION 的版本信息。HIWORD 中包含了主版本号，LOWORD 中包含了次版本号。

ConnID

由 HTTP 服务器分配的唯一数字。它不能被修改。

dwHttpStatusCode

当完成请求时当前事务的状态。可能是下列值之一：

- HTTP_STATUS_BAD_REQUEST
- HTTP_STATUS_AUTH_REQUIRED
- HTTP_STATUS_FORBIDDEN
- HTTP_STATUS_NOT_FOUND
- HTTP_STATUS_SERVER_ERROR
- HTTP_STATUS_NOT_IMPLEMENTED

lpszLogData

大小为 HSE_LOG_BUFFER_LEN 的缓冲区。包含了当前事务的与 ISA 相关的以 null 结尾的登记信息字符串，这个登记信息将被输入到 HTTP 服务器日志。出于管理的目的，为 HTTP 服务器和 ISA 事务维护一个日志文件会非常有用。

lpszMethod

生成请求的方法。与 CGI 变量 REQUEST_METHOD 等价。

lpszQueryString

包含查询信息的以 null 结尾的字符串。与 CGI 变量 QUERY_STRING 等价。

lpszPathInfo

以 null 结尾的字符串，包含了客户给出的附加路径信息。与 CGI 变量 PATH_INFO 等价。

lpszPathTranslated

以 null 结尾的字符串，包含了转换路径。与 CGI 变量 PATH_TRANSLATED 等价。

cbTotalBytes

要从客户端接收的字节总数。与 CGI 变量 CONTENT_LENGTH 等价。如果这个值为 0xffffffff，则有 4G 或更多的数据。在这种情况下，必须调用 C HttpServerContext:: ReadClient 直到没有更多的数据。

cbAvailable

lpbData 所指向的缓冲区的可能容量。如果 cbTotalBytes 与 cbAvailable 相同，变量 lpbData 指向的缓冲区将包含了客户发出的所有数据。否则 cbTotalBytes 将包含接收到的数据的总字节数。ISA 将需要适用回调函数 CHttpServerContext::ReadClient 来读出数据的剩余部分（从偏移为 cbAvailable 的位置开始）。

lpbData

指向一个大小为 cbAvailable 的缓冲区，其中包含了客户发出的数据。

lpszContentType

以null结尾的字符串，包含了客户发出数据的内容的类型。与CGI变量CONTENT_TYPE等价。

GetServerVariable

这个函数将与HTTP连接或服务器本身有关的信息（包括CGI变量）拷贝到一个缓冲区中。GetServerVariable接收如下的参数：

- *hConn* 连接的句柄。
- *lpszVariableName* 以null结尾的字符串，指明被要求的是哪个变量。变量的名字为：

变量名	描述
ALL_HTTP	所有还没有被解析为一个上面的变量之一的HTTP头。这些变量的形式为：HTTP_<起始域名>
AUTH_PASS	当客户提供密码时，这将获得与REMOTE_USER对应的密码。它是一个以null结尾的字符串

续表

AUTH_TYPE	包含了使用的鉴定的类型。例如，如果使用了 Basic 鉴定，该字符串为“Basic”。对于 WindowsNTChallenge-response，这个字符串为“NTLM”。其它鉴定模式将具有别的字符串。因为可以在 Internet 服务器中加入新的鉴定类型，所以要列出所以可能的字符串是不现实的。如果该字符串为空，则没有使用鉴定
CONTENT_LENGTH	脚本能够期望从客户接收的字节数
CONTENT_TYPE	在 POST 请求的主体部分提供的有关内容类型的信息
GATEWAY_INTERFACE	服务器支持的 CGI 规格的修订版。当前的版本为 CGI/1.1
HTTP_ACCEPT	特殊情形的 HTTP 头。Accept 的值：连结的域，通过“，”隔开。例如，如果下面的几行代码是 HTTP 头的一部分： accept:*/*;q=0.1 accept:text/html accept:image/jpeg 那么 HTTP_ACCEPT 变量将具有以下值： */*;q=0.1,text/html,image/jpeg

续表

PATH_INFO	附加的路径信息，与客户给定的相同。这个构成了 URL 中在脚本名之后，而在查询字符串（如果有）之前的部分
PATH_TRANSLATED	这是 PATH_INFO 的值，但是还把一些虚拟路径名扩展到了目录信息中。
QUERY_STRING	在引用该脚本的 URL 中？后面的信息
REMOTE_ADDR	客户的 IP 地址
REMOTE_HOST	客户的主机名字
REMOTE_USER	这里包括了客户提供，经服务器鉴定的用户名
REQUEST_METHOD	HTTP 的请求方法
SCRIPT_NAME	要指向的脚本程序的名字
SERVER_NAME	服务器的主机名（或 IP 地址），与在引用自己的 URL 中出现的一样
SERVER_PORT	接收到请求的 TCP/IP 端口
SERVER_PROTOCOL	与请求相关的信息获取协议的名字和版本。通常是 HTTP/1.0
SERVER_SOFTWARE	CGI 程序运行的 Web 服务器的名字和版本

lpvBuffer

指向缓冲区的指针，该缓冲区用于接收请求的信息。

LpdwSize

指向一个 DWORD 值的指针，该值指明了缓冲区可以容纳的字节数。如果成功地结束，DWORD 中包含了传送到缓冲区内的字节数目（包括结束符 null）。

WriteClient

从指定的缓冲区向客户发送信息。WriteClient 具有如下参数：

- *ConnID* HTTP 服务器分配的唯一的连接数。
- *Buffer* 指向要写入数据的缓冲区的指针。
- *lpdwBytes* 指向要写入的数据的指针。
- *dwReserved* 为将来的用途保留。

ReadClient

将 Web 客户的 HTTP 请求中的信息读入调用者提供的缓冲区。ReadClient 具有如下参数：

- *ConnID* HTTP 服务器分配的唯一的连接数。
- *lpvBuffer* 指向接收要求的信息的缓冲区的指针。
- *lpdwSize* 指向 DWORD 值的指针，该值指明缓冲区能够获得的字节数。**lpdwSize* 中包含了实际传送到缓冲区的字节数。

ServerSupportFunction

向 ISA 提供一些通用目的函数，例如与 HTTP 服务器实现相关的函数。

ServerSupportFunction 具有如下参数：

- *hConn* 连接句柄。
- *dwHSERRequest* HTTP 服务器扩展值。可能的取值和相关参数的列表参见 ChttpServerContext::ServerSupportFunction。
- *lpvBuffer* 与 HSE_REQ_SEND_RESPONSE_HEADER 一起使用的时候，它指向一个以 null 结尾的字符串（例如，“401 Acess Denied”）。如果这个缓冲区为空，这个函数将发出缺省的响应“200 OK”。与 HSE_REQ_DONE_WITH_SESSION 一起使用的时候，它指向一个 DWORD 值，指明了请求的状态代码。
- *lpdwSize* 与 HSE_REQ_SEND_RESPONSE_HEADER 一起使用的时候，它指向 *ldwDataType* 缓冲区的大小。
- *lpdwDataType* 以 nll 为结尾的字符串，指向将与头一起发送的可选的头或数据。如果为 NULL，这个头将以一组“\r\n”结束。

注释

服务器用后缀 .EXE 和 .BAT 将文件标识为 CGI（通用网关接口）可执行程序。另外，服务器能用 DLL 后缀将文件标识为要执行的脚本。

当服务器载入 DLL 时，它在入口点 ChttpServer::GetExtensionVersion 调用 DLL，以获得 HTTP_FILTER_REVISION 的版本号，以及服务器管理者的简短文本描述。对于每个客户请求，都要调用 ChttpServer::GetExtensionVersion 入口点。

扩展将接收共同需要的信息，如查询字符串，路径信息，方法名和转换路径。

请参阅

ChttpServerContext::ReadClient, ChttpServer::GetExtensionVersion, ChttpServer::HttpExtensionProc

FILETIME 结构

FILETIME 结构具有如下形式：

```
typedef struct _FILETIME {
    DWORD dwLowDateTime;    /* low 32 bits   */
    DWORD dwHighDateTime;   /* high 32 bits */
} FILETIME, *PFILETIME, *LPFILETIME;
```

FILETIME 结构是一个 64 位值，代表了自 1601 年以来经过的 100 纳秒的个数。

成员

dwLowDateTime

指定了文件时间的低 32 位值。

DwHighDateTime

指定了文件时间的高 32 位值。

请参阅 Ctime::Ctime

HSE_VERSION_INFO 结构

HSE_VERSION_INFO 结构具有如下形式：

```
typedef struct _HSE_VERSION_INFO {
    DWORD dwExtensionVersion;
    CHAR lpszExtensionDesc[HSE_MAX_EXT_DLL_NAME_LEN];
} HSE_VERSION_INFO, *LPHSE_VERSION_INFO;
```

这个结构是由 ChttpServer::GetExtensionVersion 成员函数的 pVer 参数所指向的。它提供了 ISA 的版本号以及 ISA 的文本描述。

成员

dwExtensionVersion

ISA 的版本号。

LpszExtensionDesc

ISA 的文本描述。缺省的实现提供了文本的位置，应当重定义 ChttpServer::GetExtensi- onVersion 函数以提供你自己的描述。

请参阅 ChttpServer::GetExtensionVersion

HTTP_FILTER_CONTEXT 结构

HTTP_FILTER_CONTEXT 结构具有如下形式：

```
typedef struct _HTTP_FILTER_CONTEXT
{
DWORD      cbSize;                          //IN
DWORD      Revision;                        //IN
PVOID      ServerContext;                   //IN
DWORD      ulReserved;                      //IN
BOOL       fIsSecurePort;                   //IN
PVOID      pFilterContext;                  //IN/OUT
BOOL     (WINAPI * GetServerVariable)
struct _HTTP_FILTER_CONTEXT *     pfc,
LPSTR            lpszVariableName,
LPVOID           lpvBuffer,
LPDWORD      lpdwSize
);
BOOL     (WINAPI * AddResponseHeaders)
struct _HTTP_FILTER_CONTEXT *     pfc,
LPSTR        lpszHeaders,
DWORD      dwReserved
```

```
);
BOOL        (WINAPI * WriteClient)
struct _HTTP_FILTER_CONTEXT *       pfc,
LPVOID        Buffer,
LPDWORD       lpdwBytes,
DWORD         dwReserved
);
VOID *        (WINAPI * AllocMem)
struct _HTTP_FILTER_CONTEXT *       pfc,
DWORD         cbSize,
DWORD         dwReserved
);
BOOL        (WINAPI * ServerSupportFunction)
struct _HTTP_FILTER_CONTEXT *       pfc,
enum SF_REQ_TYPE       sfReq,
PVOID           pData,
DWORD           ul1,
DWORD           ul2
);
} HTTP_FILTER_CONTEXT, *PHTTP_FILTER_CONTEXT;
```

上面的注释 IN **或** IN/OUT **指明成员是仅用于发送到过滤器的消息（**IN**）还是**

用于发送到过滤器和从过滤器发出的消息（OUT）。

成员

cbSize
　　以字节数计的结构大小。

Revision
　　结构的修正级别，小于或等于 HTTP_FILTER_REVISION 的版本。

ServerContext
　　为服务器的使用而保留。

UlReserved
　　为服务器的使用而保留。

FIsSecurePort
　　如果为 TRUE 表明这一事件通过一个安全端口发生。

PFilterContext
　　过滤器使用的指针，指向环境信息，过滤器将之与请求相关联。在 SF_NOTIFY_END _OF_NET_SESSION 通知中可以安全地释放与请求相关的内存。

GetServerVariable
　　函数的指针，此函数用于获取服务器和连接的信息。有关细节参见

ChttpServerCon- text::GetServerVariable。GetServerVariable 具有以下参数：

- *pfc* 指向传递到 ChttpFilter::HttpFilterProc 的过滤器环境的指针。
- *lpszVariableName* 要获取的服务器变量。
- *lpvBuffer* 用于存储变量值的缓冲区。
- *lpdwSize* 缓冲区 *lpvBuffer* 的大小。

AddResponseHeaders

函数的指针，此函数在 HTTP 响应上加上头。有关细节参见 ChttpServerContext::Server- SupportFunction 中对于 HSE_REQ_SEND_RESPONSE_HEADER 的描述。AddRespon- seHeaders 具有以下参数：

- *pfc* 指向传送给 ChttpFilter::HttpFilterProc 过滤器环境的指针。
- *lpszHeaders* 包含要加的头的字符串指针。
- *dwReserved* 留作将来使用。必须为 0。

WriteClient

函数的指针，此函数将原始数据发回客户。有关细节参见 ChttpFilterContext::Write- Client。WriteClient 具有以下参数：

- *pfc* 传送给 ChttpFilter::HttpFilterProc 的指针。
- *Buffer* 包含发送给客户的数据的缓冲区。
- *lpdwBytes* *Buffer* 指向的缓冲区大小。

- *dwReserved* 留作将来使用。

AllocMem

指向用来分配内存的函数的指针。当请求结束时，这个函数分配的内存将被自动释放。AllocMem 具有以下参数：

- *pfc* 传递给 ChttpFilter::HttpFilterProc 的指针。
- *cbSize* 要分配的缓冲区的大小。
- *dwReserved* 留作将来使用。

ServerSupportFunction

指向用来扩展 ISAPI 过滤器 API 的指针。下面列出的参数与使用的 ISA 有关：

- *pfc* 指向用来扩展 ISAPI 过滤器 API 的函数的指针。
- *sfReq* 服务器函数通知。可能值如下：

  SF_REQ_SEND_RESPONSE_HEADER 发送一个完整的 HTTP 服务器响应头，包括状态，服务器版本，消息时间和 MIME 版本。服务器扩展会在信息的结尾添加其它信息，比如内容类型，内容长度等等，最后是额外的'\r\n'。

  SF_REQ_ADD_HEADERS_ON_DENIAL 如果服务器忽略 HTTP 请求，在服务器错误响应加上特定的头。这允许一个授权的过滤器不过滤任何请求而广播它的服务。通常这个头是遵照自定义鉴定模式的

WWW 鉴定头，但是对头指定的内容并不作任何限制。

SF_REQ_SET_NEXT_READ_SIZE 仅被原始数据过滤器使用，返回 SF_STATUS_ READ_NEXT。

PData

字符串指针。与 ISA 有关。对 sfReg 的每个值，相应的 pData 值参见注释部分的表格。

Ul1, ul2

与 ISA 有关。对 sfReg 的每个值，相应的值参见注释部分的表格。

注释

下面是与 ServerSupportFunction 参数对应的可能值：

sfReq	pData	ul1, ul2
SF_REQ_SEND_RESPONSE_HEADER	以零结尾的字符串，指向可选的状态字符串（比如“401 AccessDenied”），对于缺省的响应“200 OK”为 NULL	以零结尾的字符串，指向将被附加到头和设置头的可选数据。如为 NULL，头将以空行结束

续表

SF_REQ_ADD_HEADERS_ON_DENIAL	以零结尾的字符串，指向一个或多个以'\r\n'结尾的头行	
SF_REQ_SET_NEXT_READ_SIZE		下一次要读的字节数

请参阅

ChttpFilter::HttpFilterProc, ChttpFilter::OnLog, ChttpServerContext, ChttpServerContext::GetServerVariable,ChttpServerContext::ServerSupportFunction, ChttpServerContext::WriteClient

HTTP_FILTER_LOG 结构

The HTTP_FILTER_LOG 结构具有如下形式：

```
typedef struct _HTTP_FILTER_LOG
{
const CHAR *     pszClientHostName;              //IN/OUT
const CHAR *     pszClientUserName;              //IN/OUT
const CHAR *     pszServerName;                  //IN/OUT
const CHAR *     pszOperation;                   //IN/OUT
```

```
const CHAR *       pszTarget;                          //IN/OUT
const CHAR *       pszParameters;                      //IN/OUT
DWORD              dwHttpStatus;                       //IN/OUT
DWORD              dwWin32Status;                      //IN/OUT
} HTTP_FILTER_LOG, *PHTTP_FILTER_LOG;
```

ChttpFilter::HttpFilterProc 中的 *pvNotification* 指向这个结构，此时 *NotificationType* 应该是 SF_NOTIFY_LOG，指明服务器将把信息记入日志文件。字符串不能改变，但是可以替换指针。如果改变了字符串指针，它们所指向的内存必须保持有效，直到产生下一次通知。

上面注释中的 IN/OUT 指明该成员应用于输入（IN）或输出（OUT）到过滤器的消息。

成员

pszClientHostName

客户的主机名。

PszClientUserName

客户的用户名。

PszServerName

客户所连接的服务器名。

PszOperation

HTTP 命令。

PszTarget

HTTP 命令的目标。

PszParameters

传递给 HTTP 命令的参数。

DwHttpStatus

HTTP 的返回状态。

DwWin32Status

Win32 错误代码。

请参阅 ChttpFilter::HttpFilterProc, ChttpFilter::OnLog

HTTP_FILTER_PREPROC_HEADERS 结构

The HTTP_FILTER_PREPROC_HEADERS 结构具有如下形式：

```
typedef struct _HTTP_FILTER_PREPROC_HEADERS
{
BOOL       (WINAPI * GetHeader)
    struct _HTTP_FILTER_CONTEXT *      pfc,
```

```
    LPSTR       lpszName,
    LPVOID      lpvBuffer,
    LPDWORD     lpdwSize
    );
BOOL      (WINAPI * SetHeader)
    struct _HTTP_FILTER_CONTEXT *      pfc,
    LPSTR      lpszName,
    LPSTR      lpszValue
    );
BOOL      (WINAPI * AddHeader)
    struct _HTTP_FILTER_CONTEXT *      pfc,
    LPSTR      lpszName,
    LPSTR      lpszValue
    );
DWORD      dwReserved;
}                                         HTTP_FILTER_PREPROC_HEADERS,
*PHTTP_FILTER_PREPROC_HEADERS;
```

ChttpFilter::HttpFilterProc 中的 *pvNotification* 指向这个结构，此时 *NotificationType* 应该是 SF_NOTIFY_PREPROC_HEADERS，指明服务器将要处理客户头。

成员

GetHeader

函数指针，该函数接收指定的头值。头名中必须包括冒号（“:”）。可以用特定的值，如“method”，“url”和“version”来接收请求行的单独部分。GetHeader 具有以下参数：

- *pfc* 从传递给 ChttpFilter::HttpFilterProc 的过滤器环境指针中获得的过滤器环境。
- *lpszName* 要获得的头的名字。
- *lpvBuffer* 指向大小为 *lpdwSize* 的缓冲区的指针，其中保存了头的值。
- *lpdwSize* *lpvBuffer* 指向的缓冲区的大小。

SetHeader

函数指针，该函数用于改变或删除头的值。SetHeader 具有以下参数：

- *pfc* 从传递给 ChttpFilter::HttpFilterProc 的过滤器环境指针中获得的过滤器环境。
- *lpszName* 要改变或删除的头的名字。
- *lpszValue* 字符串指针，要把头改变为该指针所指向的字符串；或者是指向“\0”的指针，表明要删除头。

AddHeader

函数指针，该函数用于加入头。AddHeader 具有如下参数：

- *pfc* 从传递给 ChttpFilter::HttpFilterProc 的过滤器环境指针中获得的过滤器环境。
- *lpszName* 要改变或删除的头的名字。
- *lpszValue* 字符串指针，要把头改变为该指针所指向的字符串；或者是指向“\0”的指针，表明要删除头。

请参阅 ChttpFilter::HttpFilterProc, ChttpFilter::OnPreprocHeaders

HTTP_FILTER_RAW_DATA 结构

The HTTP_FILTER_RAW_DATA 结构具有如下形式：

```
typedef struct _HTTP_FILTER_RAW_DATA
{
PVOID          pvInData;                    //IN
DWORD          cbInData;                    //IN
DWORD          cbInBuffer;                  //IN
DWORD          dwReserved;                  //IN
} HTTP_FILTER_RAW_DATA, *PHTTP_FILTER_RAW_DATA;
```

这个结构被传递给 ChttpFilter::HttpFilterProc 的 SF_NOTIFY_READ_RAW_DATA 和 SF_NOTIFY_SEND_RAW_DATA 通知类型。

上面注释中的 IN 指明要处理的消息将发往过滤器。

成员

pvInData

指向数据缓冲区的指针（输入或输出）。

CbInData

pvInData 所指向的缓冲区中数据的数量。

CbInBuffer

pvInData 所指向的缓冲区的大小。

DwReserved

为将来的使用保留。

请参阅 ChttpFilter::HttpFilterProc, ChttpFilter::OnReadRawData, ChttpFilter::OnSendRawData

HTTP_FILTER_URL_MAP 结构

The HTTP_FILTER_URL_MAP 结构具有如下形式：

```
typedef struct _HTTP_FILTER_URL_MAP
{
```

```
const CHAR *        pszURL;                  //IN
CHAR *              pszPhysicalPath;         //IN/OUT
DWORD               cbPathBuff;              //IN
} HTTP_FILTER_URL_MAP, *PHTTP_FILTER_URL_MAP;
```

ChttpFilter::HttpFilterProc 中的 *pvNotification* 指向这个结构，此时 *NotificationType* 应该是 SF_NOTIFY_URL_MAP，指明服务器将把指定了 URL 映射到实际路径。过滤器可以随时修改实际路径。

上面注释中的 IN 或 IN/OUT 指明该成员是仅适用于发往过滤器的消息（IN）还是对发往过滤器和过滤器发出的消息都适用（IN/OUT）。

成员

pszURL

将要被映射到实际路径的 URL。

PszPhysicalPath

保存实际路径的缓冲区指针。

CbPathBuff

pszPhysicalPath 所指向的缓冲区的大小。

请参阅 ChttpFilter::HttpFilterProc, ChttpFilter::OnUrlMap

HTTP_FILTER_VERSION 结构

HTTP_FILTER_VERSION 结构具有如下形式：

```
typedef struct _HTTP_FILTER_VERSION
{
DWORD   dwServerFilterVersion;                          //IN
DWORD   dwFilterVersion;                                //OUT
CHAR    lpszFilterDesc[SF_MAX_FILTER_DESC_LEN+1];       //OUT
DWORD   dwFlags;                                        //OUT
} HTTP_FILTER_VERSION, *PHTTP_FILTER_VERSION;
```

这个结构被服务器传递给应用程序的 ChttpFilter::HttpFilterProc 入口点，用来把任何环境信息与 HTTP 请求关联起来。

上面注释中的 IN 或 OUT 指明该成员是适用于发往过滤器的消息（IN）还是过滤器发出的消息（OUT）。

成员

dwServerFilterVersion

过滤使用的头的版本。当前头文件的版本为 HTTP_FILTER_REVISION。

DwFilterVersion

HTTP_FILTER_REVISION 的版本。

LpszFilterDesc

保存简短的字符串的位置，该字符串描述了 ISAPI 过滤器应用程序。

DwFlags

SF_NOTIFY_* 标志的组合，用来指明这个应用程序需要什么事件，过滤器以何种优先级载入。有效标志的列表参见 ChttpFilter::GetFilterVersion 和 ChttpFilter:: HttpFilterProc。

请参阅 ChttpFilter::HttpFilterProc, ChttpFilter::GetFilterVersion

LINGER 结构

LINGER 结构具有如下形式：

```
struct linger {
      u_short l_onoff;                    // option on/off
      u_short l_linger;                   // linger time
};
```

LINGER 结构被用来操作 CasyncSocket::GetSockOpt 的 SO_LINGER 和 SO_DONTLINGER 选项。

注释

设置了 SO_DONTLINGER 选项可以防止成员函数阻塞。在等待发送还未发送

出去的数据时可以关闭。设置这个选项等价于把 SO_LINGER 的 l_onoff 设为 0。

请参阅 CasyncSocket::GetSockOpt, CasyncSocket::SetSockOpt

LOGBRUSH 结构

LOGBRUSH 结构具有如下形式：

```
typedef struct tag LOGBRUSH { /* lb */
    UINT          lbStyle;
    COLORREF   lbColor;
    LONG          lbHatch;
} LOGBRUSH;
```

LOGBRUSH 结构定义了物理刷子的风格、颜色和模板。它被 Windows 的 CreateBrushIndirect 和 ExtCreatePen 函数使用。

成员

lbStyle

指定了刷子的风格。LbStyle 成员必须是以下风格之一：

- BS_DIBPATTERN 一个代模板的刷子，用设备无关位图（DIB）来定义。如果 lbStyle 为 BS_DIBPATTERN，那么 lbHatch 成员中包含了压缩 DIB 的句柄。

- BS_DIBPATTERNPT 一个代模板的刷子，用设备无关位图（DIB）来定义。如果 lbStyle 为 BS_DIBPATTERNPT，那么 lbHatch 成员中包含了指向压缩 DIB 的指针。
- BS_HATCHED 阴影刷子。
- BS_HOLLOW 空刷子。
- BS_NULL 与 BS_HOLLOW 相同。
- BS_PATTERN 用内存位图定义的模板刷子。
- BS_SOLID 实心刷子。

LbColor

指定了画出刷子的颜色。如果 lbStyle 为 BS_HOLLOW 或 BS_PATTERN 风格，将会忽略 lbColor。如果 lbStyle 为 BS_DIBPATTERN 或 BS_DIBPATTERNBT，则 lbColor 的低位字指定了 BITMAPINFO 结构的 bmiColors 成员是包含了准确的红、绿、蓝（RGB）值还是包含当前使用的逻辑调色板的索引。LbColor 成员可以取如下值之一：

- DIB_PAL_COLORS 颜色表中包含了当前使用的逻辑调色板中的 16 位索引数组。
- DIB_RGB_COLORS 颜色表中包含了准确的 RGB 值。

LbHatch

指定了阴影的风格。其含义依赖于 lbStyle 所定义的刷子风格。如果 lbStyle 为 BS_DIBPATTERN，则 lbHatch 成员中包含了包装的 DIB 的句柄。如果 lbStyle 为 BS_DIBPATTERNPT，则 lbHatch 成员中包含了包装的 DIB

的指针。如果 lbStyle 为 BS_HATCHED，则 lbHatch 成员指定了创建阴影时使用的线条的方向。它可以取下面列出的值：

- HS_BDIAGONAL　45 度向上，从左到右的阴影。
- HS_CROSS　纵横交叉的阴影。
- HS_DIAGCROSS　45 度交叉的阴影。
- HS_FDIAGONAL　45 度向下，从左到右的阴影。
- HS_HORIZONTAL　水平阴影。
- HS_VERTICAL　垂直阴影。

如果 lbStyle 是 BS_PATTERN，则 lbHatch 为定义了模板的位图句柄。如果 lbStyle 为 BS_SOLID 或 BS_HOLLOW，则 lbHatch 被忽略。

注释

lbColor 控制着阴影刷子的前景色，CDC::SetBkMode 和 CDC::SetBkColorbm 函数控制着背景色。

请参阅　CDC::GetCharABCWidths

LOGPEN 结构

LOGPEN 结构具有如下形式：

```
typedef struct tagLOGPEN {    /* lgpn */
```

```
    UINT        lopnStyle;
    POINT       lopnWidth;
    COLORREF    lopnColor;
} LOGPEN;
```

LOGPEN 结构定义了画笔的风格、宽度和颜色，画笔是用于画出线条和边界的绘图对象。Cpen::CreatePenIndirect 函数使用 LOGPEN 结构。

成员

lopnStyle

指定了画笔的风格。这个成员可以是以下值之一：

- PS_SOLID 创建实心的画笔。
- PS_DASH 创建短线画笔。（仅当画笔宽度为 1 时有效）
- PS_DOT 创建点线画笔。（仅当画笔宽度为 1 时有效）
- PS_DASHDOT 创建点划线画笔。（仅当画笔宽度为 1 时有效）
- PS_DASHDOTDOT 创建短线与两个点相间的画笔。（仅当画笔宽度为 1 时有效）
- PS_NULL 创建空画笔。
- PS_INSIDEFRAME 创建一种画笔，在封闭图形的框架内部画线，这个图形是由那些指定边界矩形的 GDI 输出函数（例如 Ellipse，Rectangle，RoundRect，Pie 以及 Chord 成员函数）生成的。当对那些

不指定边界矩形的 GDI 输出函数（例如 LineTo 成员函数）应用这个风格时，画笔的绘图区域并不被限制在框架内部。

如果画笔具有 PS_INSIDEFRAME 风格，并且其颜色与逻辑颜色表中的颜色不匹配，那么将用抖动色画出该画笔。不能用 PS_SOLID 风格来创建具有抖动色的画笔。当画笔宽度小于或等于 1 时，PS_INSIDEFRAME 风格与 PS_SOLID 风格相同。

当对 Ellipse，Rectangle 和 RoundRect 以外的函数产生的 GDI 对象使用 PS_INSIDEFRAME 风格时，画出的线条可能不会完全位于指定的框架内部。

LopnWidth

指定画笔的宽度，使用逻辑单位。如果 lopnWidth 成员为 0，则不论当前的映射模式是什么，画笔在光栅设备上的宽度都是一个象素。

LopnColor

指定画笔的颜色。

注释

lopnWidth 成员中 POINT 结构的 y 值没有被使用。

请参阅　Cpen::CreatePenIndirect

MEASUREITEMSTRUCT 结构

MEASUREITEMSTRUCT 数据结构具有如下形式：

```
typedef struct tagMEASUREITEMSTRUCT {
    UINT      CtlType;
    UINT      CtlID;
    UINT      itemID;
    UINT      itemWidth;
    UINT      itemHeight;
    DWORD   itemData
} MEASUREITEMSTRUCT;
```

MEASUREITEMSTURCT 结构通知 Windows 自画控件或菜单项的尺度。这使得 Windows 能够正确处理控件的用户交互。如果没有正确地填充 MEASUREITEMSTRUCT 结构中的成员，可能会导致控件的不正确的操作。

成员

CtlType

包含了控件的类型。控件类型的取值如下：

- ODT_COMBOBOX 自画组合框。
- ODT_LISTBOX 自画列表框。

- ODT_MENU 自画菜单。

CtlID

包含了组合框、列表框或按钮的控制 ID。菜单不使用这个成员。

ItemID

包括了菜单的菜单项 ID 或是可变高度的组合框或列表框中列表框项的 ID。这个成员不对固定高度的组合框、列表框和按钮使用。

ItemWidth

指定了菜单项的宽度。自画菜单项的所有者必须在它从消息返回之前填充这个成员。

ItemHeight

指定了列表框或菜单中一项的高度。在从消息返回之前，自画组合框、列表框或菜单项的所有者必须填充这个成员。列表框项的最大高度为 255。

ItemData

对于组合框或列表框，这个成员中包含了下列函数传递给列表框的值：

- CcomboBox::AddString
- CcomboBox::InsertString
- ClistBox::AddString
- ClistBox::InsertString

对于菜单，这个成员中包含了下列函数传递给菜单的值：

- Cmenu::AppendMenu
- Cmenu::InsertMenu
- Cmenu::ModifyMenu

请参阅 CWnd::OnMeasureItem

MINMAXINFO 结构

MINMAXINFO 结构具有如下形式：

```
typedef struct tagMINMAXINFO {
    POINT ptReserved;
    POINT ptMaxSize;
    POINT ptMaxPosition;
    POINT ptMinTrackSize;
    POINT ptMaxTrackSize;
} MINMAXINFO;
```

MINMAXINFO 结构中包含了有关窗口的最大化大小和位置以及最大、最小跟踪大小的信息。

成员

ptReserved

为内部使用保留。

PtMaxSize

指定了窗口的最大化宽度（point.x）和最大化高度（point.y）。

ptMaxPosition

指定了最大化窗口的左边位置（point.x）和顶部位置（point.y）。

ptMinTrackSize

指定了窗口的最小跟踪宽度（point.x）和最小跟踪高度（point.y）。

ptMaxTrackSize

指定了窗口的最大跟踪宽度（point.x）和最大跟踪高度（point.y）。

请参阅 CWnd::OnGetMinMaxInfo

MSG 结构

MSG 结构具有如下形式：

```
typedef struct tagMSG {        // msg
    HWND        hwnd;
    UINT        message;
```

```
    WPARAM   wParam;
    LPARAM   lParam;
    DWORD    time;
    POINT    pt;
} MSG;
```

MSG 结构中包含了线程的消息队列中的消息信息。

成员

hwnd

标识了接收的消息的窗口过程所属的窗口的句柄。

Message

指定了消息号。

WParam

指定了消息的附加信息。具体的含义与 message 成员的值有关。

LParam

指定了消息的附加信息。具体的含义与 message 成员的值有关。

Time

指定了发出消息的时间。

Pt

指定了发出消息时光标位置的屏幕坐标。

NCCALCSIZE_PARAMS结构

NCCALCSIZE_PARAMS结构具有如下形式：

```
typedef struct tagNCCALCSIZE_PARAMS {
    RECT            rgrc[3];
    PWINDOWPOS      lppos;
} NCCALCSIZE_PARAMS;
```

NCCALCSIZE_PARAMS结构中包含了一些信息，应用程序在处理WM_NCCALCSIZE消息的时候用它来计算窗口客户区的大小，位置和有效内容。

成员

rgrc

指定了一个矩形数组。第一个矩形包含了窗口被移动或改变大小之后的新坐标。第二个矩形包含了窗口被移动或改变大小之前的坐标。第三个矩形包含了在被移动或改变大小之前窗口客户区的坐标。如果该窗口是一个子窗口，那么这些坐标是相对于父窗口的客户区的。如果该窗口是

一个顶层窗口，那么这些坐标是相对于屏幕的。

Lppos

指向 WINDOWPOS 结构，其中包含了引起窗口移动或改变大小的操作所指定的大小或位置值。

请参阅 CWnd::OnNcCalcSize

PAINTSTRUCT 结构

PAINTSTRUCT 结构具有如下形式：

```
typedef struct tagPAINTSTRUCT {
    HDC  hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved[16];
} PAINTSTRUCT;
```

PAINTSTRUCT 结构中包含了被用于画出窗口客户区的信息。

成员

hdc

标识了用于绘图的显示环境。

FErase

指定是否要重画背景。如果应用程序需要重画背景，则其为非零值。如果创建 Windows 的窗口类时没有使用背景刷子（参见 Win32 SDK 文档中 WNDCLASS 结构的 hbrBackground 成员的描述），那么应由应用程序负责画出背景。

RcPaint

指定了要重画的矩形区域的左上角和右下角。

FRestore

保留成员。它被 Windows 内部使用。

FIncUpdate

保留成员。它被 Windows 内部使用。

RgbReserved[16]

保留成员。Windows 内部使用的保留内存块。

请参阅 CpaintDC::m_ps

POINT 结构

POINT 数据结构具有如下形式：

```
typedef struct tagPOINT {
    LONG x;
    LONG y;
} POINT;
```

POINT 结构定义一个点的 x 和 y 坐标。

成员

x

    指定了点的 x 坐标。

Y

    指定了点的 y 坐标。

请参阅　Cpoint

RECT 结构

RECT 数据结构具有如下形式：

```
typedef struct tagRECT {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT;
```

RECT **结构定义了矩形的左上角坐标和右下角坐标。**

成员

left

指定了矩形的左上角的 x **坐标。**

Top

指定了矩形的左上角的 y **坐标。**

Right

指定了矩形的右下角的 x **坐标。**

Bottom

指定了矩形的右下角的 y **坐标。**

请参阅 Crect

RGNDATA结构

RGNDATA结构具有如下形式：

```
typedef struct _RGNDATA { /* rgnd */
    RGNDATAHEADER rdh;
    char          Buffer[1];
} RGNDATA;
```

RGNDATA结构中包含了一个头和一个矩形数组，组成了一个区域。这些矩形从左到右、从上到下排列，没有重叠。

成员

rdh

　　指定了一个RGNDATAHEADER结构。（有关这个结构的更多信息参见Win32 SDK文档）这个结构的成员指定了区域的类型（是矩形还是不规则多边形），组成区域的矩形数目，包含矩形结构的缓冲区大小，等等。

Buffer

　　指定了任意大小的缓冲区，其中包含了组成区域的RECT结构。

请参阅　CRgn::CreateFromData, CRgn::GetRegionData

SIZE 结构

SIZE 结构具有如下形式：

```
typedef struct tagSIZE {
      int cx;
      int cy;
} SIZE;
```

SIZE 结构指定了矩形的宽度和高度。

成员

cx

指定了函数返回时的 x 范围。

Cy

指定了函数返回时的 y 范围。

注释

这个结构中保存的矩形尺度信息可以作为视口的范围，窗口范围，文本范围，位图尺度或者是一些扩展函数使用的 aspect-ratio 过滤器。

请参阅 Csize

SOCKADDR 结构

SOCKADDR 结构具有如下形式：

```
struct sockaddr {
      unsigned short    sa_family;
      char              sa_data[14];
};
```

SOCKADDR 结构被用来保存参与 Windows Sockets 通讯的机器的 Internet 协议（IP）地址。

成员

sa_family

Socket 地址家族。

Sa_data

所有不同的 socket 地址结构的最大大小。

注释

Microsoft TCP/IP Sockets 开发包只支持 Internet 地址域。为了填充地址的每个部分，应当使用 SOCKADDR_IN 结构，它是这个地址格式所特有的。SOCKADDR 和 SOCKADDR_IN 数据结构的大小相同。你可以在这两种数据结构之间转换。更多的信息参见 Win32 SDK 文档中的“Windows Sockets 编程考虑”。

请参阅 SOCKADDR_IN, CasyncSocket::Create, Csocket::Create

SOCKADDR_IN 结构

SOCKADDR_IN 结构具有如下形式：

```
struct sockaddr_in{
      short              sin_family;
      unsigned short     sin_port;
      struct   in_addr   sin_addr;
      char               sin_zero[8];
};
```

在 Internet 地址家族中，SOCKADDR_IN 结构被 Windows Sockets 用来指定要连接的本地或远程结束点地址。这是一种与 Internet 地址家族有关的 SOCKADDR 结构的形式，并且它可以被强制转换为 SOCKADDR。

成员

sin_family
    地址家族（必须是 AF_INET**）。**

Sin_port
    IP **端口。**

Sin_addr
    IP **地址。**

Sin_zero
    用于将该结构对齐到与 SOCKADDR **相同的大小。**

注释

这个结构中的 IP **地址部分属于** IN_ADDR **类型。在** Windows Sockets **头文件** WINSOCK.H **中，**IN_ADDR **是按照下面的方式定义的：**

```
struct   in_addr {
    union   {
        struct{
            unsigned   char     s_b1,
                                s_b2,
                                s_b3,
```

```
                        s_b4;
        }   S_un_b;
            struct  {
            unsigned  short     s_w1,
                                s_w2;
             }  S_un_w;
              unsigned long     S_addr;
      } S_un;
};
```

更多的信息参见 Win32 SDK 文档中的“Windows Sockets 编程考虑”。

请参阅 SOCKADDR

SYSTEMTIME 结构

SYSTEMTIME 结构具有如下形式：

```
typedef struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
```

```
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

SYSTEMTIME 结构代表了日期和时间，对于月、日、年、周、时、分、秒和毫秒分别使用了不同的成员。

成员

wYear

当前的年。

WMonth

当前的月，一月为 1。

WDayOfWeek

当前的周日，星期日为 0，星期一为 1，依次类推。

WDay

当前的日期。

WHour

当前的小时。

WMinute

当前的分钟数。

WSecond

当前的秒数。

WMilliseconds

当前的毫秒数。

请参阅 Ctime::Ctime

TEXTMETRIC 结构

TEXTMETRIC 结构具有如下形式：

```
typedef struct tagTEXTMETRIC {   /* tm */
    int   tmHeight;
    int   tmAscent;
    int   tmDescent;
    int   tmInternalLeading;
    int   tmExternalLeading;
    int   tmAveCharWidth;
    int   tmMaxCharWidth;
```

```
    int  tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int  tmOverhang;
    int  tmDigitizedAspectX;
    int  tmDigitizedAspectY;
} TEXTMETRIC;
```

TEXTMETRIC **结构中包含了有关物理字体的基本信息。所有的大小都是用逻辑单位给出的，这意味着，它们依赖于显示环境的当前映射模式。**

有关这个结构的更完整的信息参见 Win32 SDK **文档中的** TEXTMETRIC。

请参阅 CDC::GetTextMetrics, TEXTMETRIC

WINDOWPLACEMENT 结构

WINDOWPLACEMENT 数据结构具有如下形式：

```
typedef struct tagWINDOWPLACEMENT {        /* wndpl */
    UINT    length;
    UINT    flags;
    UINT    showCmd;
    POINT   ptMinPosition;
    POINT   ptMaxPosition;
    RECT    rcNormalPosition;
} WINDOWPLACEMENT;
```

WINDOWPLACEMENT 结构中包含了有关窗口在屏幕上位置的信息。

成员

length

指定了结构的长度，以字节为单位。

Flags

指定了控制最小化窗口的位置的标志以及复原窗口的方法。这个成员可以是下面列出的标志之一，或都是：

- WPF_SETMINPOSITION 表明可以指定最小化窗口的 x 和 y 坐标。如果是在 ptMinPosition 成员中设置坐标，则必须指定这个标志。
- WPF_RESTORETOMAXIMIZED 表明复原后的窗口将会被最大化，而不管它在最小化之前是否是最大化的。这个设置仅在下一次复原窗口时有效。它不改变缺省的复原操作。这个标志仅当 showCmd 成员中指定了 SW_SHOWMINIMIZED 时才有效。

ShowCmd

指定了窗口的当前显示状态。这个成员可以是下列值之一：

- SW_HIDE 隐藏窗口，使其它窗口变为激活的。
- SW_MINIMIZE 最小化指定的窗口，并激活系统列表中的顶层窗口。
- SW_RESTORE 激活并显示窗口。如果窗口是最小化或最大化的，Windows 将把它恢复到原来的大小和位置（与 SW_SHOWNORMAL 相同）。
- SW_SHOW 激活窗口并按照当前的位置和大小显示窗口。
- SW_SHOWMAXIMIZED 激活窗口并将其显示为最大化的。
- SW_SHOWMINIMIZED 激活窗口并将其显示为图标。
- SW_SHOWMINNOACTIVE 将窗口显示为图标。当前激活的窗口仍保持激活状态。
- SW_SHOWNA 按当前状态显示窗口。当前激活的窗口仍保持激活状态。

- SW_SHOWNOACTIVATE 按最近的位置和大小显示窗口。当前激活的窗口仍保持激活状态。
- SW_SHOWNORMAL 激活并显示窗口。如果窗口是最小化或最大化的，Windows 将它恢复到原来的大小和位置（与 SW_RESTORE 相同）。

PtMinPosition

指定了窗口被最小化时左上角的位置。

PtMaxPosition

指定了窗口被最大化时左上角的位置。

RcNormalPosition

指定了窗口处于正常状态（复原）时的坐标。

请参阅 CWnd::SetWindowPlacement

WINDOWPOS 结构

WINDOWPOS 数据结构具有如下形式：

```
typedef struct tagWINDOWPOS { /* wp */
    HWND    hwnd;
    HWND    hwndInsertAfter;
    int     x;
```

```
    int          y;
    int          cx;
    int          cy;
    UINT         flags;
} WINDOWPOS;
```

WINDOWPOS 结构包含了有关窗口的大小和位置的信息。

成员

hwnd

标识窗口。

HwndInsertAfter

标识了一个窗口，本窗口将被放在这个窗口的后面。

X

指定了窗口的左边界的位置。

Y

指定了窗口的右边界的位置。

Cx

指定了窗口的宽度，以象素为单位。

Cy

指定了窗口的高度，以象素为单位。

Flags

指定了窗口位置的选项。这个成员可以是下列值之一：

- SWP_DRAWFRAME 画出窗口的边框（在窗口类的描述中定义）。窗口接收到一个 WM_NCCALCSIZE 消息。
- SWP_FRAMECHANGED 向窗口发送一个 WM_NCCALCSIZE 消息，即使没有改变窗口的大小。如果没有指定这个标志，仅当窗口的大小发生变化时才发送 WM_NCCALCSIZE 消息。
- SWP_HIDEWINDOW 隐藏窗口。
- SWP_NOACTIVATE 不激活窗口。
- SWP_NOCOPYBITS 废弃客户区的全部内容。如果没有指定这个标志，将会保存客户区的有效内容并在窗口被改变大小或重定位以后回送到客户区。
- SWP_NOMOVE 保留当前位置（忽略 x 和 y 成员）。
- SWP_NOOWNERZORDER 不改变所有者窗口在 Z 轴上的顺序。
- SWP_NOSIZE 保留当前大小（忽略 cx 和 cy 成员）。
- SWP_NOREDRAW 不重画改变的内容。
- SWP_NOREPOSITION 与 SWP_NOOWNERZORDER 相同。
- SWP_NOSENDCHANGING 防止窗口接收 WM_WINDOWPOSCHANGING 消息。

- SWP_NOZORDER　保留当前顺序（忽略 hwndInsertAfter 成员）
- SWP_SHOWWINDOW　显示窗口。

请参阅　CWnd::OnWindowPosChanging

WSADATA 结构

WSADATA 结构具有如下形式：

```
struct WSAData {
     WORD                 wVersion;
     WORD                 wHighVersion;
     char                 szDescription[WSADESCRIPTION_LEN+1];
     char                 szSystemStatus[WSASYSSTATUS_LEN+1];
     unsigned short       iMaxSockets;
     unsigned short       iMaxUdpDg;
     char FAR *           lpVendorInfo;
};
```

WSADATA 结构被用来保存全局函数 AfxSocketInit 返回的 Windows Sockets 初始化信息。

成员

wVersion

Windows Sockets DLL 期望调用者使用的 Windows Sockets 规范的版本。

WHighVersion

这个 DLL 能够支持的 Windows Sockets 规范的最高版本。通常它与 wVersion 相同。

SzDescription

以 null 结尾的 ASCII 字符串，Windows Sockets DLL 将对 Windows Sockets 实现的描述拷贝到这个字符串中，包括制造商标识。文本（最多可以有 256 个字符）可以包含任何字符，但是要注意不能包含控制字符和格式字符，应用程序对其最可能的使用方式是把它（可能被截断）显示在在状态信息中。

SzSystemStatus

以 null 结尾的 ASCII 字符串，Windows Sockets DLL 把有关的状态或配置信息拷贝到该字符串中。Windows Sockets DLL 应当仅在这些信息对用户或支持人员有用时才使用它们，它不应被作为 szDescription 域的扩展。

IMaxSockets

单个进出能够打开的 socket 的最大数目。Windows Sockets 的实现能提供

一个全局的 socket 池，可以为任何进程分配；或者它也可以为 socket 分配属于进程的资源。这个数字能够很好地反映 Windows Sockets DLL 或网络软件的配置方式。应用程序的编写者可以通过这个数字来粗略地指明 Windows Sockets 的实现方式对应用程序是否有用。例如，X Windows 服务器在第一次启动的时候可能会检查 iMaxSockets 的值：如果这个值小于 8，应用程序将显示一条错误信息，指示用户重新配置网络软件（这是一种可能要使用 szSystemStatus 文本的场合）。显然无法保证某个应用程序能够真正分配 iMaxSockets 个 socket，因为可能有其它 Windows Sockets 应用程序正在使用。

IMaxUdpDg

Windows Sockets 应用程序能够发送或接收的最大的用户数据包协议（UDP）的数据包大小，以字节为单位。如果实现方式没有限制，那么 iMaxUdpDg 为零。在 Berkeley sockets 的许多实现中，对于 UDP 数据包有个固有的限制（在必要时被分解），大小为 8192 字节。Windows Sockets 的实现可以对碎片重组缓冲区的分配作出限制。对于适合的 Windows Sockets 实现，iMaxUdpDg 的最小值为 512。注意不管 iMaxUdpDg 的值是什么，都不推荐你发回一个比网络的最大传送单元（MTU）还大的广播数据包。（Windows Sockets API 没有提供发现 MTU 的机制，但是它不会小于 512 个字节）

lpVendorInfo

指向销售商的数据结构的指针。这个结构的定义（如果有）超出了 Windows

Sockets 规范的范围。更多的信息"参见 Win32 SDK 文档"中的"Windows Sockets 编程考虑"。

注意 在 MFC 中，WSADATA 结构是由 AfxSocketInit 函数返回的，你在 InitInstance 函数中调用这个函数。如果你需要在以后使用这些信息，你可以获得这个结构并将它保存在程序中。

请参阅 AfxSocketInit

XFORM 结构

XFORM 结构具有如下形式：

```
typedef struct   tagXFORM {   /* xfrm */
    FLOAT eM11;
    FLOAT eM12;
    FLOAT eM21;
    FLOAT eM22;
    FLOAT eDx;
    FLOAT eDy;
} XFORM;
```

注释

XFORM 结构指定了从世界空间到页面空间的转换。EDx 和 eDy 成员分别指定了水平和垂直转换成分。下面的表格显示了其它成员是如何被使用的，依赖于其它操作：

操作	eM11	eM12	eM21	eM22
旋转	旋转角度的余弦	旋转角度的正弦	旋转角度的反正弦	旋转角度的余弦
缩放	水平缩放成分	空	空	垂直缩放成分
修剪	空	水平比例常量	垂直比例常量	空
反射	水平反射成分	空	空	垂直反射成分

请参阅　CRgn::CreateFromData

MFC 使用的风格

在多数情况下，下面的主题中描述的风格是在 dwstyle 参数中指定的。进一步的信息参考每种风格的“请参阅”部分中列出的成员函数。

按钮风格

- BS_AUTOCHECKBOX 与复选框相同，但是当用户选择复选框时，检查标记出现在复选框中，而当用户再一次选择复选框时，检查标记就消失。
- BS_AUTORADIOBUTTON 与单项按钮相同，但是当用户选择它的时候，这个按钮自动加亮显示自己并去掉同组中相同风格的其它单项按钮的选择状态。
- BS_AUTO3STATE 与三态复选框相同，但是当用户选择该框时它会改变自己的状态。
- BS_CHECKBOX 创建一个小方块，在它的右边显示文本（除非这个风格与 BS_LEFTTEXT 风格一起使用）。
- BS_DEFPUSHBUTTON 创建一个具有深黑边界的按钮。用户可以按下 ENTER 键以选择这个按钮。这个风格使用户可以快速地选择最相似的选项（缺省选项）。
- BS_GROUPBOX 创建一个矩形区域，其中的按钮是成组的。与这种风格相关的任何文本将显示在矩形的左上角。
- BS_LEFTTEXT 当与单项按钮风格或复选框风格一起使用时，文本出现在单项按钮或复选框的左边。
- BS_OWNERDRAW 创建一个自画按钮。当按钮的视觉状态发生改变时，框架调用 DrawItem 成员函数。当使用 CBitmapButton 类的时候，必须设置这个风格。

- BS_PUSHBUTTON 创建一个按钮，当用户选择该按钮时向所有者窗口发送一个 WM_COMMAND 消息。
- BS_RADIOBUTTON 创建一个小圆形区域，在它的右边显示文本（除非这个风格与 BS_LEFTTEXT 风格一起使用）。单项按钮通常成组使用但是只能独占选择。
- BS_3STATE 与复选框类似，但是这个框不仅可以被选中，还可以被变灰。变灰状态通常用来标识该复选框已经被禁止。

请参阅 CButton::Create

组合框风格

- CBS_AUTOHSCROLL 当用户在行尾输入一个字符时，自动把编辑控件中的文本向右滚动。如果没有设置该风格，则输入的文本信息只能多到填满矩形边框。
- CBS_DROPDOWN 与 CBS_SIMPLE 类似，但是除非用户选择了编辑控件旁边的图标，否则不会显示列表框。
- CBS_DROPDOWNLIST 与 CBS_DROPDOWN 类似，但是编辑控件被静态文本项代替，其中显示了列表框中的当前选择。
- CBS_HASSTRINGS 包含了字符串组成的项的自画组合框。组合框维护着字符串的内存和指针，因此应用程序可以使用 GetText 成员函数从某个项获得文本。

- CBS_OEMCONVERT 在组合框的编辑控件内输入的文本将从 ANSI 字符集转换到 OEM 字符集，然后再回到 ANSI。当应用程序调用 Windows 的 AnsiToOem 函数把组合框中的一个 ANSI 字符串转换到 OEM 字符时，这能确保进行了合适的字符转换。这个风格对那些包含了文件名的组合框最有用，仅适用于用 CBS_SIMPLE 或 CBS_DROPDOWN 风格创建的组合框。
- CBS_OWNERDRAWFIXED 列表框的拥有者负责画出其内容，列表框中所有项的高度是一样的。
- CBS_OWNERDRAWVARIABLE 列表框的拥有者负责画出其内容，列表框中各项的高度是不一致的。
- CBS_SIMPLE 任何时候都显示列表框。列表框的当前选择显示在编辑控件中。
- CBS_SORT 自动排列输入到列表框的字符串。
- CBS_DISABLENOSCROLL 当列表框没有足够的项以供滚动时，列表框将显示一个被禁止的垂直滚动条。如果没有这种风格，当列表框不包含足够的项时，这个滚动条将会被隐藏。
- CBS_NOINTEGRALHEIGHT 指明组合框的大小就是应用程序在创建该组合框时指定的大小。通常，Windows 会调整一些组合框的大小，使得组合框不需要显示部分项。

请参阅 CComboBox::Create

编辑风格

- ES_AUTOHSCROLL 当用户在行尾输入字符时，自动将文本向右滚动 10 个字符。当用户按下 ENTER 键时，控件将文本滚动回起始位置。
- ES_AUTOVSCROLL 当用户在最后一行输入 ENTER 时，自动将文本向上滚动一页。
- ES_CENTER 在单行或多行编辑控件中将文本对中。
- ES_LEFT 在单行或多行编辑控件中将文本靠左对齐。
- ES_LOWERCASE 将用户输入到编辑控件的字符全部转换为小写。
- ES_MULTILINE 指明了一个多行编辑控件（缺省的是单行的）。如果指定了 ES_AUTOVSCROLL 风格，编辑控件将显示尽可能多的文本，并且当用户按下 ENTER 键时会自动地垂直滚动文本。如果没有指定 ES_AUTOVSCROLL 风格，则编辑控件将显示尽可能多的行，如果在按下 ENTER 键却没有更多的行要显示的话，就发出蜂鸣声。如果指定了 ES_AUTOHSCROLL 风格，当光标到达控件的右边时，多行编辑控件会自动地水平滚动文本。如果要开始一个新行，用户必须按下 ENTER 键。如果没有指定 ES_AUTOHSCROLL 风格，控件会在有必要时自动将单词折合到下一行的开始。如果按下 ENTER 键，则另起一行。折回单词的位置是由窗口的大小决定的。如果窗口的大小发生改变，折回单词的位置也会反生改变，将会重新显示文本。多行编辑控件可以有滚动条。具有滚动条的编辑控件会处理它自己的滚动条消息。没有滚动条的编辑控件按照前面描

述的方式进行滚动，并且处理父窗口发出的任何滚动消息。

- ES_NOHIDESEL　通常，当编辑控件失去输入焦点时，它会隐藏选择区域，当它获得输入焦点时，它会反转显示选择区域。如果指定了ES_NOHIDESEL风格则去掉了这个缺省的动作。
- ES_OEMCONVERT　输入到编辑控件的文本将被从ANSI字符集转换到OEM字符集，然后转换回ANSI字符集。这使得在应用程序调用Windows的AnsiToOem函数以把编辑控件中的ANSI字符串转换为OEM字符时，能够进行正确的字符转换。这个风格对包含文件名的编辑控件最有用。
- ES_PASSWORD　在编辑控件中输入字符时，将所有的字符显示为星号（*）。应用程序可以通过SetPasswordChar成员函数来改变显示的字符。
- ES_RIGHT　在单行或多行编辑控件中将文本靠右对齐。
- ES_UPPERCASE　在编辑控件中输入字符时，将所有的字符转换为大写。
- ES_READONLY　禁止用户输入或修改编辑控件中的文本。
- ES_WANTRETURN　指定当用户在对话框中的多行编辑控件中输入文本时，如果按下了ENTER键，则插入回车换行符。如果不使用这个风格，按下ENTER键的效果与按下对话框的缺省按钮相同。这个风格对单行编辑控件不起作用。

请参阅　CEdit::Create

框架窗口风格

- FWS_ADDTOTITLE 指定了要附加到框架窗口标题末尾的信息。例如，“Microsoft Draw - Drawing in Document1”。你可以指定显示在 AppWizard 的 Advanced Options 对话框中的字符串。如果你希望关闭这个选项，重载 CWnd::PreCreateWindow 消息。
- FWS_PREFIXTITLE 在框架窗口的标题中，在应用程序的名字之前显示文档的名字。例如，“Document - WordPad”。你可以指定显示在 AppWizard 的 Advanced Options 对话框中的字符串。如果你希望关闭这个选项，重载 CWnd::PreCreateWindow 消息。
- FWS_SNAPTOBARS 控制框架窗口的大小，该窗口围绕着一个控制条，这个控制条是一个浮动窗口，而不是固定在框架窗口中。这个风格调整窗口的大小以使用控制条。

列表框风格

- LBS_EXTENDEDSEL 用户可以通过鼠标和 SHIFT 键或者其它特殊键组合来选取多个项。
- LBS_HASSTRINGS 指定自画列表框中包含的项是由字符串组成的。列表框维护着字符串的内存和指针，应用程序可以使用 GetText 成员函数来获得特定项的文本。

- LBS_MULTICOLUMN 指定一个可以水平滚动的多列列表框。SetColumnWidth 成员函数设置列的宽度。
- LBS_MULTIPLESEL 当用户单击或双击字符串时，切换字符串的选择状态。可以选择任意数目的字符串。
- LBS_NOINTEGRALHEIGHT 列表框的大小与应用程序创建它的时候指定的大小完全相等。通常，Windows 会调整列表框的大小，是列表框不会只显示部分项。
- LBS_NOREDRAW 当列表框发生变化时不更新显示。这个风格可以通过发送 WM_SETREDRAW 消息在任何时间改变。
- LBS_NOTIFY 当用户单击或双击字符串时，父窗口接收到一个输入消息。
- LBS_OWNERDRAWFIXED 列表框的所有者负责画出它的内容，列表框中的各项是等高的。
- LBS_OWNERDRAWVARIABLE 列表框的所有者负责画出其内容，列表框中的各项的高度不相同。
- LBS_SORT 列表框中的字符串是按照字母表顺序排列的。
- LBS_STANDARD 列表框中的字符串是按照字母表顺序排序的，当用户单击或双击字符串时，父窗口接收到一个输入消息。列表框在每条边上都有边界。
- LBS_USETABSTOPS 允许列表框在显示字符串的时候识别并扩展制表字符。缺省的制表位置是 32 个对话框单位。（对话框单位是水平或垂直距

离。水平对话框单位等于当前对话框基准宽度单位的四分之一。对话框基准单位是通过当前系统字体的宽度和高度来计算的。Windows 的 GetDialogBaseUnits 函数返回以象素为单位的当前对话框基准单位。）

- LBS_WANTKEYBOARDINPUT 不论什么时候，只要用户在列表框具有输入焦点的时候按下了键，列表框就接收到 WM_VKEYTOITEM 或 WM_CHARTOITEM 消息。这使得应用程序能够对键盘输入进行特别处理。
- LBS_DISABLENOSCROLL 当列表框中没有足够多的项，不需要滚动时，就显示一个被禁止的垂直滚动条。如果不使用这个风格，当列表框不包含足够多的项时，它就隐藏滚动条。

请参阅 CListBox::Create

消息框风格

消息框类型

- MB_ABORTRETRYIGNORE 消息框包含三个按钮：Abort，Retry 和 Ignore。
- MB_OK 消息框包含一个按钮：OK。
- MB_OKCANCEL 消息框包含两个按钮：OK 和 Cancel。

- MB_RETRYCANCEL 消息框包含两个按钮：Retry 和 Cancel。
- MB_YESNO 消息框包含两个按钮：Yes 和 No。
- MB_YESNOCANCEL 消息框包含三个按钮：Yes，No 和 Cancel。

消息框模式

- MB_APPLMODAL 用户在当前窗口中继续工作之前必须先响应消息框。但是，用户可以移动到其它应用程序的窗口中并在那些窗口中工作。如果没有指定 MB_SYSTEMMODAL 和 MB_TASKMODAL，则缺省值为 MB_APPLMODAL。
- MB_SYSTEMMODAL 在用户响应消息框之前，所有的应用程序都被挂起。系统模式消息框被用来向用户通知严重的、潜在的毁灭性错误，需要立即注意，小心对待。
- MB_TASKMODAL 与 MB_APPLMODAL 类似，但是在微软基础类应用程序中没有用处。这个标志是为那些没有窗口句柄的调用应用程序或库保留的。

消息框图标

- MB_ICONEXCLAMATION 在消息框中显示感叹号图标。
- MB_ICONINFORMATION 在消息框中显示一个圆包围着字母“i”的图标。

- MB_ICONQUESTION 在消息框中显示问号图标。
- MB_ICONSTOP 在消息框中显示停止标志图标。

消息框缺省按钮

- MB_DEFBUTTON1 第一个按钮是缺省按钮。注意，除非指定了 MB_DEFBUTTON2 或 MB_DEFBUTTON3 风格，否则第一个按钮总是缺省按钮。
- MB_DEFBUTTON2 第二个按钮是缺省按钮。
- MB_DEFBUTTON3 第三个按钮是缺省按钮。

请参阅 AfxMessageBox

滚动条风格

- SBS_BOTTOMALIGN 与 SBS_HORZ 风格一起使用。滚动条的底边与 Create 成员函数中指定的矩形的底边对齐。滚动条具有系统滚动条的缺省高度。
- SBS_HORZ 指明了一个水平滚动条。如果既没有指定 SBS_BOTTOMALIGN 风格又没有指定 SBS_TOPALIGN 风格，则滚动条具有 Create 成员函数中指定的高度，宽度和位置。
- SBS_LEFTALIGN 与 SBS_VERT 风格一起使用。滚动条的左边与 Create

成员函数中指定的矩形的左边对齐。滚动条具有系统滚动条的缺省宽度。

- SBS_RIGHTALIGN 与 SBS_VERT 风格一起使用。滚动条的右边与 Create 成员函数中指定的矩形的右边对齐。滚动条具有系统滚动条的缺省宽度。
- SBS_SIZEBOX 指明了一个尺寸框。如果 SBS_SIZEBOXBOTTOMRIGHTALIGN 风格和 SBS_SIZEBOXTOPLEFTALIGN 风格都没有指定，尺寸框具有 Create 成员函数中指定的高度，宽度和位置。
- SBS_SIZEBOXBOTTOMRIGHTALIGN 与 SBS_SIZEBOX 风格一起使用。尺寸框的右下角与 Create 成员函数中指定的矩形的右下角对齐。尺寸框具有系统尺寸框的缺省大小。
- SBS_SIZEBOXTOPLEFTALIGN 与 SBS_SIZEBOX 风格一起使用。尺寸框的左上角与 Create 成员函数中指定的矩形的左上角对齐。尺寸框具有系统尺寸框的缺省大小。
- SBS_TOPALIGN 与 SBS_HORZ 风格一起使用。滚动条的顶边与 Create 成员函数中指定的矩形的顶边对齐。滚动条具有系统滚动条的缺省高度。
- SBS_VERT 指明了一个垂直滚动条。如果既没有指定 SBS_RIGHTALIGN 风格也没有指定 SBS_LEFTALIGN 风格，滚动条具有 Create 成员函数中指定的高度，宽度和位置。

请参阅 CScrollBar::Create

静态文本风格

- SS_BLACKFRAME 指定了一个带边框的方框，用与窗口边框相同的颜色画出。缺省的颜色是黑色。
- SS_BLACKRECT 指定一个矩形，用窗口边框的颜色填充。缺省颜色是黑色。
- SS_CENTER 指定一个简单的矩形，在矩形的中央显示给定的文本。文本将在显示之前格式化。超出行尾的单词将自动折回到下一行的开始。
- SS_GRAYFRAME 指定一个带边框的方框，用屏幕的背景色（桌面颜色）画出。缺省的颜色是灰色。
- SS_GRAYRECT 指定一个矩形，用屏幕的背景色填充。缺省的颜色是灰色。
- SS_ICON 指定了在对话框中显示的图标。给定的文本是资源文件中定义的图标名（不是文件名）。*nWidth* 和 *nHeight* 参数被忽略，图标自动调整自己的大小。
- SS_LEFT 指定一个简单的矩形，在矩形内显示左对齐的给定文本。文本在显示之前格式化。超出行尾的单词将自动折回到下一行的开始。
- SS_LEFTNOWORDWRAP 指定一个简单的矩形，在矩形内显示左对齐的给定文本。制表符被扩展，但是不会折回单词。超出行尾的单词被裁剪。
- SS_NOPREFIX 除非指定了这个风格，否则 Windows 将控制文本中所有的“&”字符解释为加速键前缀。在这种情况下，“&”被移去，字符串

中的下一个字符被加上下划线。如果一个包含文本的静态文本控件不需要这个特性，可能需要加入 SS_NOPREFIX。这个风格可以用于任何静态控件。你可以用位或操作符把 SS_NOPREFIX 与其它风格组合起来。最常使用这个风格的情况是，可能要在对话框的静态控件中显示带有“&”字符的文件名或其它字符串。

- SS_RIGHT 指定一个简单的矩形，在矩形内显示右对齐的给定文本。文本在显示之前格式化。超出行尾的单词将自动折回到下一行的开始。
- SS_SIMPLE 指定一个简单的矩形，在矩形内显示一行左对齐的文本。文本行不能用任何方法缩短或改变。（控件的父窗口或对话框不能处理 WM_CTLCOLOR 消息）
- SS_USERITEM 指定一个用户自定义的项。
- SS_WHITEFRAME 指定一个带边框的方框，用窗口背景色画出。缺省值为白色。
- SS_WHITERECT 指定一个矩形，用窗口背景色填充。缺省值为白色。

请参阅 CStatic::Create

窗口风格

- WS_BORDER 创建一个有边界的窗口。
- WS_CAPTION 创建一个有标题条的窗口（隐含 WS_BORDER 风格）。不能与 WS_DLGFRAME 风格一起使用。

- WS_CHILD 创建一个子窗口。不能与 WS_POPUP 风格一起使用。
- WS_CLIPCHILDREN 当你在父窗口中绘图时，除去子窗口所占的区域。在创建父窗口的时候使用。
- WS_CLIPSIBLINGS 剪裁相关的子窗口，这意味着，当一个特定的子窗口接收到重绘消息时，WS_CLIPSIBLINGS 风格将在子窗口要重画的区域中去掉与其它子窗口重叠的部分。（如果没有指定 WS_CLIPSIBLINGS 风格，并且子窗口有重叠，当你在一个子窗口的客户区绘图时，它可能会画在相邻的子窗口的客户区中。）只与 WS_CHILD 风格一起使用。
- WS_DISABLED 创建一个初始状态为禁止的窗口。
- WS_DLGFRAME 创建一个窗口，具有双重边界，但是没有标题条。
- WS_GROUP 指定一组控件中的第一个，用户可以用箭头键在这组控件中移动。在第一个控件后面把 WS_GROUP 风格设置为 FALSE 的控件都属于这一组。下一个具有 WS_GROUP 风格的控件将开始下一组（这意味着一个组在下一组的开始处结束）。
- WS_HSCROLL 创建一个具有水平滚动条的窗口。
- WS_MAXIMIZE 创建一个最大化的窗口。
- WS_MAXIMIZEBOX 创建一个具有最大化按钮的窗口。
- WS_MINIMIZE 创建一个初始状态为最小化的窗口。仅与 WS_OVERLAPPED 风格一起使用。
- WS_MINIMIZEBOX 创建一个具有最小化按钮的窗口。
- WS_OVERLAPPED 创建一个重叠窗口。重叠窗口通常具有标题条和边

界。

- WS_OVERLAPPEDWINDOW 创建一个具有WS_OVERLAPPED，WS_CAPTION，WS_SYSMENU，WS_THICKFRAME，WS_MINIMIZEBOX和WS_MAXIMIZEBOX风格的重叠式窗口。
- WS_POPUP 创建一个弹出式窗口，不能与WS_CHILD风格一起使用。
- WS_POPUPWINDOW 创建一个具有WS_BORDER，WS_POPUP和WS_SYSMENU风格的弹出窗口。为了使控制菜单可见，必须与WS_POPUPWINDOW一起使用WS_CAPTION风格。
- WS_SYSMENU 创建一个在标题条上具有控制菜单的窗口。仅对带标题条的窗口使用。
- WS_TABSTOP 指定了一些控件中的一个，用户可以通过TAB键来移过它。TAB键使用户移动到下一个用WS_TABSTOP风格定义的控件。
- WS_THICKFRAME 创建一个具有厚边框的窗口，可以通过厚边框来改变窗口大小。
- WS_VISIBLE 创建一个最初可见的窗口。
- WS_VSCROLL 创建一个具有垂直滚动条的窗口。

请参阅 CWnd::Create, CWnd::CreateEx

扩展窗口风格

- WS_EX_ACCEPTFILES 指明用这个风格创建的窗口能够接受拖放文

件。

- WS_EX_CLIENTEDGE 指明窗口具有3D外观，这意味着，边框具有下沉的边界。
- WS_EX_CONTEXTHELP 在窗口的标题条中包含问号。当用户单击问号时，鼠标光标的形状变为带指针的问号。如果用户随后单击一个子窗口，子窗口将接收到一个WM_HELP消息。
- WS_EX_CONTROLPARENT 允许用户用TAB键遍历窗口的子窗口。
- WS_EX_DLGMODALFRAME 指明一个具有双重边界的窗口，当你在*dwStyle*参数中指定了WS_CAPTION风格标志时，它可以具有标题条（可选）。
- WS_EX_LEFT 指定窗口具有左对齐属性。这是缺省值。
- WS_EX_LEFTSCROLLBAR 将垂直滚动条放在客户区的左边。
- WS_EX_LTRREADING 按照从左到右的方式显示窗口文本。这是缺省方式。
- WS_EX_MDICHILD 创建一个MDI子窗口。
- WS_EX_NOPARENTNOTIFY 指定用这个风格创建的子窗口在被创建或销毁的时候将不向父窗口发送WM_PARENTNOTIFY消息。
- WS_EX_OVERLAPPEDWINDOW 组合了WS_EX_CLIENTEDGE和WS_EX_WIND-
OWEDGE风格。
- WS_EX_PALETTEWINDOW 组合了WS_EX_WINDOWEDGE和

WS_EX_TOPMOST 风格。

- WS_EX_RIGHT 赋予窗口右对齐属性。这与窗口类有关。
- WS_EX_RIGHTSCROLLBAR 将垂直滚动条（如果有）放在客户区的右边。这是缺省方式。
- WS_EX_RTLREADING 按照从右到左的顺序显示窗口文本。
- WS_EX_STATICEDGE 创建一个具有三维边界的窗口，用于不接受用户输入的项。
- WS_EX_TOOLWINDOW 创建一个工具窗口，目的是被用作浮动工具条。工具窗口具有标题条，比通常的标题条要短，窗口的标题是用小字体显示的。工具窗口不出现在任务条或用户按下 ALT+TAB 时出现的窗口中。
- WS_EX_TOPMOST 指定用这个风格创建的窗口必须被放在所有非顶层窗口的上面，即使这个窗口已经不处于激活状态，它还是保留在最上面。应用程序可以用 SetWindowsPos 成员函数来加入或去掉这个属性。
- WS_EX_TRANSPARENT 指定了用这个风格创建的窗口是透明的。这意味着，在这个窗口下面的任何窗口都不会被这个窗口挡住。用这个风格创建的窗口只有当它下面的窗口都更新过以后才接收 WM_PAINT 消息。
- WS_EX_WINDOWEDGE 指定了具有凸起边框的窗口。

请参阅 CWnd::CreateEx

MFC 使用的回调函数

在微软基础类型中出现了三个回调函数。下面的主题中描述了传递给 CDC::EnumObjects，CDC::GrayString 和 CDC::SetAbortProc 的回调函数。有关回调函数的一般用法参见这些处于函数的说明部分。注意所有的回调函数在返回 Windows 前都必须捕捉异常，因为异常不能被抛到回调边界之外。有关异常的更多信息参见"Visual C++ 程序员指南"中的文章"异常"。

CDC::EnumObjects 的回调函数

```
int CALLBACK EXPORT ObjectFunc( LPSTR lpszLogObject, LPSTR* lpData );
```

参数

lpszLogObject

指向一个 LOGPEN 或 LOGBRUSH 数据结构的指针，其中包含了有关对象的逻辑属性的信息。

lpData

指向应用程序提供的传递给 EnumObjects 函数的数据。

返回值

这个回调函数返回一个整数。返回的值是用户定义的。如果回调函数返回 0，则 EnumObjects 中止枚举。

说明

ObjectFunc 名为应用程序提供的函数名预留的位置。实际的名字必须要引出。

请参阅 CDC::EnumObjects

CDC::GrayString 的回调函数

BOOL CALLBACK EXPORT OutputFunc(HDC *hDC*, LPARAM *lpData*, int *nCount*);

返回值

如果成功，回调函数的返回值必须是 TRUE；否则为 FALSE。

参数

hDC

标识了位图的内存设备环境，其宽度和高度至少为 GrayString 涉及的

nWidth 和 nHeight。

lpData

指向要显示的字符串。

nCount

指定了要输出的字符个数。

说明

OutputFunc 为应用程序提供的回调函数名预留了位置。回调函数（OutputFunc）必须从（0，0）而不是（x，y）开始画出图形。

请参阅 CDC::GrayString

CDC::SetAbortProc 的回调函数

BOOL CALLBACK EXPORT AbortFunc(HDC *hPr*, int *code*);

返回值

如果要继续打印任务，则 abort 处理函数的返回值必须为非零值。如果要取消，则返回值为 0。

参数

hPr

标识设备环境。

code

指明是否发生了错误。如果没有发生错误，则为 0。如果打印管理器已经用完了磁盘空间，并且如果应用程序等待就能获得更多磁盘空间，则返回 SP_OUTOFDISK。如果 *code* 为 SP_OUTOFDISK，应用程序不必放弃打印任务。如果 *code* 不是 SP_OUTOFDISK，它必须通过调用 Windows 的 PeekMessage 或 GetMessage 函数服从打印管理器。

说明

AbortFunc 为应用程序提供的函数名预留了位置。实际的名字必须要引出，如 CDC::SetAbortProc 的说明部分所描述的。

请参阅 CDC::SetAbortProc

消息映射

这个部分列出了所有的消息映射宏和所有的 CWnd 消息映射条目及其对应的成

员函数原型。

分类	描述
WM_COMMAND 消息处理函数	处理用户菜单选择或菜单键产生的 WM_COMMAND 消息
子窗口通知消息处理函数	处理子窗口发出的通知消息
WM_消息处理函数	处理 WM_消息，如 WM_PAINT
用户自定义消息处理函数	处理用户自定义消息

（本参考中使用的术语和阅读的解释参见“如何使用消息映射交叉参考”）

由于 Windows 是一个面向消息的操作系统，在 Windows 环境下很大一部分编程工作涉及消息处理。每当发生一个事件，如击键或鼠标点击，就会向应用程序发送一个消息，然后由它来处理事件。

微软基础类库提供了为基于消息的编程而优化的编程模式。在这种模式下，“消息映射”被用于指明哪个函数将为特定的类处理不同的消息。消息映射包含了一个或多个宏，用以指定哪个函数处理哪个消息。例如，一个包含 ON_COMMAND 宏的消息映射看起来可能象这样：

```
BEGIN_MESSAGE_MAP( CMyDoc, CDocument )
    //{{AFX_MSG_MAP( CMyDoc )
    ON_COMMAND( ID_MYCMD, OnMyCommand )
    // ...其它入口，用于处理另外的消息
    //}}AFX_MSG_MAP
```

END_MESSAGE_MAP()

ON_COMMAND宏被用于处理菜单、按钮和加速键产生的命令消息。可以用宏来映射下列消息：

Windows消息

- 控件通知
- 用户自定义消息

命令消息

- 注册的用户自定义消息
- 用户界面更新消息

消息范围

- 命令
- 更新处理消息
- 控件通知

尽管消息映射宏很重要，通常你并不需要直接使用它们。这是因为当你用

ClassWizard把消息处理函数与消息关联在一起的时候，它将会在源文件中自动创建消息映射入口。不论何时你希望编辑或加入消息映射条目，你都可以使用ClassWizard。

> **注意** ClassWizard不支持消息映射范围。你必须自己写入这些消息映射入口。

但是，消息映射是微软基础类库中很重要的一个部分。你必须理解它们的作用，类库也提供了有关文档。

消息映射宏

为了支持消息映射，MFC提供了下列宏：

消息映射的声明和分界宏

DECLARE_MESSAGE_MAP	声明将在一个类中使用消息映射，把消息映射到函数（必须用在类声明中）
BEGIN_MESSAGE_MAP	开始消息映射的定义（必须用在类实现中）
END_MESSAGE_MAP	结束消息映射的定义（必须用在类实现中）

消息映射宏

ON_COMMAND	指明将由哪个函数处理指定的命令消息

续表

ON_CONTROL	指明将由哪个函数处理指定的控件通知消息
ON_MESSAGE	指明将由哪个函数处理用户自定义的消息
ON_OLECMD	指明将由哪个函数处理 DocObject 或其容器发出的菜单命令
ON_REGISTERED_MESSAGE	指明将由哪个函数处理注册的用户自定义消息
ON_REGISTERED_THREAD_MESSAGE	指明当你拥有一个 CWinThread 类时，将由哪个函数处理注册的用户自定义消息
ON_THREAD_MESSAGE	指明当你拥有一个 CWinThread 类时，将由哪个函数处理用户自定义消息
ON_UPDATE_COMMAND_UI	指明将由哪个函数处理指定的用户界面更新命令消息

消息映射范围宏

ON_COMMAND_RANGE	指明将由哪个函数处理该宏的前两个参数所指定的范围内的命令 ID
ON_UPDATE_COMMAND_UI_RANGE	指明将由哪个更新处理器处理该宏的前两个参数所指定的范围内的命令 ID

续表

ON_CONTROL_RANGE	指明将由哪个函数处理该宏的第二和第三个参数所指定的范围内的控制 ID 发出的通知。第一个参数是一个控件通知消息，例如 BN_CLICKED

有关消息映射、消息映射的声明和分界宏以及消息映射宏的更多的信息参见“消息处理”和“映射主题”。有关消息映射范围的更多信息参见“消息映射范围的处理函数”。有关如何使用 ClassWizard 的更多信息参见“使用 ClassWizard”。这些参考都在“Visual C++ 程序员指南”中。

如何使用消息映射交叉参考

在标着<memberFxn>的条目中，写入你的 CWnd 派生类的成员函数。可以给你的函数取任意一个你喜欢的名字。其它函数，例如 OnActive，是 CWnd 类的成员函数。如果调用了它们，它们将消息传递给 Windows 的 DefWindowProc 函数。如果要处理 Windows 通知消息，在你自己的类中重载相应的 CWnd 成员函数。你可以在函数中调用基类的被重载的函数，使基类和 Windows 能够回应消息。

在所有的情况下，将函数原型放入 CWnd 派生类的头文件中，并按照下面的方式编写消息映射入口。

使用了下面的术语：

术语	定义
id	任何用户自定义的菜单项 ID（WM_COMMAND 消息）或控件（子窗口通知消息）ID
"message"和"wNotifyCode"	WINDOWS.H 中定义的 Windows 消息 ID
nMessageVariable	包含了 Windows 的 RegisterWindowsMessage 函数的返回值的变量的名字。它必须被声明为 NEAR

WM_COMMAND 消息处理函数

映射入口	函数原型
ON_COMMAND(<id>,<memberFxn>)	afx_msg void memberFxn( );

子窗口通知消息处理函数

子窗口通知消息有五个部分：

分类	描述
通用控件处理函数	处理通用控件通知代码
用户按钮处理函数	处理用户按钮通知代码
组合框处理函数	处理组合框通知代码

续表

编辑控件处理函数	处理编辑控件通知代码
列表框处理函数	处理列表框通知代码

一般控件处理函数

映射入口	函数原型
ON_CONTROL(<wNotifyCode>,<id>, <memberFxn>)	afx_msg void memberFxn( );

用户按钮处理函数

映射入口	函数原型
ON_BN_CLICKED(<id>, <memberFxn>)	afx_msg void memberFxn( )
ON_BN_DISABLE(<id>, <memberFxn>)	afx_msg void memberFxn( )
ON_BN_DOUBLECLICKED(<id>, <memberFxn>)	afx_msg void memberFxn( )
ON_BN_HILITE(<id>, <memberFxn>)	afx_msg void memberFxn( )
ON_BN_PAINT(<id>, <memberFxn>)	afx_msg void memberFxn( )
ON_BN_UNHILITE(<id>, <memberFxn>)	afx_msg void memberFxn( )

组合框处理函数

映射入口	函数原型
ON_CBN_CLOSEUP(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_DBLCLK(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_DROPDOWN(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_EDITCHANGE(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_EDITUPDATE(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_ERRSPACE(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_KILLFOCUS(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_SELCHANGE(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_SELENDCANCEL(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_SELENDOK(<id>, <memberFxn>)	afx_msgvoid memberFxn( )
ON_CBN_SETFOCUS(<id>, <memberFxn>)	afx_msgvoid memberFxn( )

编辑控件处理函数

映射入口	函数原型
ON_EN_CHANGE(<id>, <memberFxn>)	afx_msg void memberFxn( )

续表

ON_EN_ERRSPACE(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_EN_HSCROLL(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_EN_KILLFOCUS(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_EN_MAXTEXT(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_EN_SETFOCUS(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_EN_UPDATE(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_EN_VSCROLL(<id>, <memberFxn>)	afx_msg void memberFxn()

列表框控件处理函数

映射入口	函数原型
ON_LBN_DBLCLK(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_LBN_ERRSPACE(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_LBN_KILLFOCUS(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_LBN_SELCHANGE(<id>, <memberFxn>)	afx_msg void memberFxn()
ON_LBN_SETFOCUS(<id>, <memberFxn>)	afx_msg void memberFxn()

WM_消息处理函数

主题	映射入口
A – C	ON_WM_ACTIVATE 到 ON_WM_CTLCOLOR
D – E	ON_WM_DEADCHAR 到 ON_WM_ERASEBKGND
F – K	ON_WM_FONTCHANGE 到 ON_WM_KILLFOCUS
L – M	ON_WM_LBUTTONDBLCLK 到 ON_WM_MOVING
N – O	ON_WM_NCACTIVATE 到 ON_WM_NCRBUTTONUP
P – R	ON_WM_PAINT 到 ON_WM_RENDERFORMAT
S	ON_WM_SETCURSOR 到 ON_WM_SYSKEYUP
T – Z	ON_WM_TIMECHANGE 到 ON_WM_WININICHANGE

WM_ 消息处理函数：A - C

映射入口	函数原型
ON_WM_ACTIVATE( )	afx_msg void OnActivate(UINT, CWnd*, BOOL)
ON_WM_ACTIVATEAPP( )	afx_msg void OnActivateApp(BOOL, HANDLE)
ON_WM_ASKCBFORMATNAME( )	afx_msgvoid OnAskCbFormatName(UINT, LPSTR)
ON_WM_CANCELMODE( )	afx_msg void OnCancelMode( )

续表

ON_WM_CAPTURECHANGED( )	afx_msgvoid OnCaptureChanged(CWnd*)
ON_WM_CHANGECBCHAIN( )	afx_msgvoid OnChangeCbChain(HWND, HWND)
ON_WM_CHAR( )	afx_msg void OnChar(UINT, UINT, UINT)
ON_WM_CHARTOITEM( )	afx_msg int OnCharToItem(UINT, CWnd*, UINT)
ON_WM_CHILDACTIVATE( )	afx_msg void OnChildActivate( )
ON_WM_CLOSE( )	afx_msg void OnClose( )
ON_WM_COMPACTING( )	afx_msgvoid OnCompacting(UINT)
ON_WM_COMPAREITEM( )	afx_msgint OnCompareItem (LPCOMPAREITEMSTRUCT)
ON_WM_CONTEXTMENU( )	afx_msgvoid OnContextMenu (CWnd*, CPoint)
ON_WM_COPYDATA( )	afx_msgBOOL OnCopyData (CWnd* Pwnd, COPYDATASTRUCT* pCopyDataStruct)

续表

ON_WM_CREATE( )	afx_msgint OnCreate(LPCREATESTRUCT)
ON_WM_CTLCOLOR( )	afx_msg HBRUSH OnCtlColor(CDC*, CWnd*, UINT)

WM_ 消息处理函数：D-E

映射入口	函数原型
ON_WM_DEADCHAR( )	afx_msg void OnDeadChar(UINT, UINT, UINT)
ON_WM_DELETEITEM( )	afx_msgvoid OnDeleteItem (LPDELETEITEMSTRUCT)
ON_WM_DESTROY( )	afx_msg void OnDestroy( )
ON_WM_DESTROYCLIPBOARD( )	afx_msg void OnDestroyClipboard( )
ON_WM_DEVICECHANGE( )	afx_msgvoid OnDeviceChange(UINT, DWORD)
ON_WM_DEVMODECHANGE( )	afx_msgvoid OnDevModeChange(LPSTR)
ON_WM_DRAWCLIPBOARD( )	afx_msg void OnDrawClipboard( )

续表

ON_WM_DRAWITEM( )	afx_msgvoid OnDrawItem (LPDRAWITEMSTRUCT)
ON_WM_DROPFILES( )	afx_msg void OnDropFiles(HDROP)
ON_WM_ENABLE( )	afx_msg void OnEnable(BOOL)
ON_WM_ENDSESSION( )	afx_msg void OnEndSession(BOOL)
ON_WM_ENTERIDLE( )	afx_msg void OnEnterIdle(UINT, CWnd*)
ON_WM_ERASEBKGND( )	afx_msg BOOL OnEraseBkgnd (CDC *)

WM_消息处理函数：F－K

映射入口	函数原型
ON_WM_FONTCHANGE( )	afx_msg void OnFontChange( )
ON_WM_GETDLGCODE( )	afx_msg UINT OnGetDlgCode( )
ON_WM_GETMINMAXINFO( )	afx_msgvoid OnGetMinMaxInfo(LPPOINT)
ON_WM_HELPINFO( )	afx_msgBOOL OnHelpInfo(HELPINFO*)
ON_WM_HSCROLL( )	afx_msg void OnHScroll(UINT, UINT, CWnd*)

续表

ON_WM_HSCROLLCLIPBOARD()	afx_msg void OnHScrollClipboard (CWnd*,UINT,UINT)
ON_WM_ICONERASEBKGND( )	afx_msg void OnIconEraseBkgnd (CDC *)
ON_WM_INITMENU( )	afx_msg void OnInitMenu(CMenu *)
ON_WM_INITMENUPOPUP( )	afx_msgvoid OnInitMenuPopup (CMenu *, UINT, BOOL)
ON_WM_KEYDOWN( )	afx_msg void OnKeyDown(UINT, UINT, UINT)
ON_WM_KEYUP( )	afx_msg void OnKeyUp(UINT, UINT, UINT)
ON_WM_KILLFOCUS( )	afx_msgvoid OnKillFocus(CWnd*)

WM_消息处理函数：L-M

映射入口	函数原型
ON_WM_LBUTTONDBLCLK( )	afx_msgvoid OnLButtonDblClk (UINT, Cpoint)

续表

ON_WM_LBUTTONDOWN( )	afx_msg void OnLButtonDown(UINT, CPoint)
ON_WM_LBUTTONUP( )	afx_msg void OnLButtonUp(UINT, CPoint)
ON_WM_MBUTTONDBLCLK( )	afx_msgvoid OnMButtonDblClk(UINT, CPoint)
ON_WM_MBUTTONDOWN( )	afx_msgvoid OnMButtonDown(UINT, CPoint)
ON_WM_MBUTTONUP( )	afx_msg void OnMButtonUp(UINT, CPoint)
ON_WM_MDIACTIVATE( )	afx_msgvoid OnMDIActivate(BOOL, CWnd*, CWnd*)
ON_WM_MEASUREITEM()	afx_msgvoid OnMeasureItem (LPMEASUREITEMSTRUCT)
ON_WM_MENUCHAR( )	afx_msgLONG OnMenuChar(UINT, UINT, CMenu *)
ON_WM_MENUSELECT( )	afx_msg void OnMenuSelect(UINT, UINT, HMENU)
ON_WM_MOUSEACTIVATE( )	afx_msgint OnMouseActivate(CWnd*, UINT, UINT)

续表

ON_WM_MOUSEMOVE( )	afx_msg void OnMouseMove(UINT, CPoint)
ON_WM_MOUSEWHEEL( )	afx_msgBOOL OnMouseWheel(UINT, short, CPoint)
ON_WM_MOVE( )	afx_msg void OnMove(int, int)
ON_WM_MOVING( )	afx_msg void OnMoving(UINT, LPRECT)

WM_ 消息处理函数：N－O

映射入口	函数原型
ON_WM_NCACTIVATE( )	afx_msgBOOL OnNcActivate(BOOL)
ON_WM_NCCALCSIZE( )	afx_msgvoid OnNcCalcSize (BOOL, NCCALCSIZE _PARAMSFAR*)
ON_WM_NCCREATE( )	afx_msgBOOL OnNcCreate (LPCREATESTRUCT)
ON_WM_NCDESTROY( )	afx_msg void OnNcDestroy( )
ON_WM_NCHITTEST( )	afx_msgUINT OnNcHitTest(Cpoint)

续表

ON_WM_NCLBUTTONDBLCLK( )	afx_msgvoid OnNcLButtonDblClk (UINT, CPoint)
ON_WM_NCLBUTTONDOWN( )	afx_msgvoid OnNcLButtonDown (UINT, CPoint)
ON_WM_NCLBUTTONUP( )	afx_msgvoid OnNcLButtonUp(UINT, CPoint)
ON_WM_NCMBUTTONDBLCLK( )	afx_msgvoid OnNcMButton DblClk(UINT, Cpoint)
ON_WM_NCMBUTTONDOWN( )	afx_msgvoid OnNcMButtonDown (UINT, CPoint)
ON_WM_NCMBUTTONUP( )	afx_msgvoid OnNcMButtonUp(UINT, CPoint)
ON_WM_NCMOUSEMOVE( )	afx_msgvoid OnNcMouseMove (UINT, CPoint)
ON_WM_NCPAINT( )	afx_msg void OnNcPaint( )
ON_WM_NCRBUTTONDBLCLK( )	afx_msgvoid OnNcRButton DblClk UINT, CPoint)
ON_WM_NCRBUTTONDOWN( )	afx_msgvoid OnNcRButtonDown (UINT, Cpoint)
ON_WM_NCRBUTTONUP( )	afx_msgvoid OnNcRButtonUp(UINT, CPoint)

WM_ 消息： P - R

映射入口	函数原型
ON_WM_PAINT( )	afx_msg void OnPaint( )
ON_WM_PAINTCLIPBOARD( )	afx_msgvoid OnPaintClipboard (CWnd*, HANDLE)
ON_WM_PALETTECHANGED( )	afx_msgvoid OnPaletteChanged(CWnd*)
ON_WM_PALETTEISCHANGING( )	afx_msgvoid OnPaletteIsChanging(CWnd*)
ON_WM_PARENTNOTIFY( )	afx_msgvoid OnParentNotify (UINT, LONG)
ON_WM_QUERYDRAGICON( )	afx_msgHCURSOR OnQueryDragIcon( )
ON_WM_QUERYENDSESSION( )	afx_msgBOOL OnQueryEndSession( )
ON_WM_QUERYNEWPALETTE( )	afx_msgBOOL OnQueryNewPalette( )
ON_WM_QUERYOPEN( )	afx_msg BOOL OnQueryOpen( )
ON_WM_RBUTTONDBLCLK( )	afx_msgvoid OnRButtonDblClk (UINT, CPoint)

续表

ON_WM_RBUTTONDOWN( )	afx_msgvoid OnRButtonDown (UINT, CPoint)
ON_WM_RBUTTONUP( )	afx_msg void OnRButtonUp (UINT, CPoint)
ON_WM_RENDERALLFORMATS( )	afx_msgvoid OnRenderAllFormats( )
ON_WM_RENDERFORMAT( )	afx_msgvoid OnRenderFormat(UINT)

WM_ 消息：S

映射入口	函数原型
ON_WM_SETCURSOR( )	afx_msg BOOL OnSetCursor (CWnd*, UINT, UINT)
ON_WM_SETFOCUS( )	afx_msg void OnSetFocus(CWnd*)
ON_WM_SHOWWINDOW( )	afx_msg void OnShowWindow (BOOL, UINT)
ON_WM_SIZE( )	afx_msg void OnSize(UINT, int, int)
ON_WM_SIZECLIPBOARD( )	afx_msgvoid OnSizeClipboard (CWnd*, HANDLE)

续表

ON_WM_SIZING( )	afx_msg void OnSizing(UINT, LPRECT)
ON_WM_SPOOLERSTATUS( )	afx_msg void OnSpoolerStatus (UINT, UINT)
ON_WM_STYLECHANGED( )	afx_msg void OnStyleChanged(int, LPSTYLESTRUCT)
ON_WM_STYLECHANGING( )	afx_msg void OnStyleChanging(int, LPSTYLESTRUCT)
ON_WM_SYSCHAR( )	afx_msg void OnSysChar(UINT, UINT, UINT)
ON_WM_SYSCOLORCHANGE( )	afx_msg void OnSysColorChange( )
ON_WM_SYSCOMMAND( )	afx_msg void OnSysCommand (UINT, LONG)
ON_WM_SYSDEADCHAR( )	afx_msg void OnSysDeadChar (UINT, UINT, UINT)
ON_WM_SYSKEYDOWN( )	afx_msg void OnSysKeyDown (UINT, UINT, UINT)
ON_WM_SYSKEYUP( )	afx_msg void OnSysKeyUp(UINT, UINT, UINT)

WM_ 消息：T-Z

映射入口	函数原型
ON_WM_TCARD( )	afx_msg void OnTCard(UINT, DWORD)
ON_WM_TIMECHANGE( )	afx_msg void OnTimeChange( )
ON_WM_TIMER( )	afx_msg void OnTimer(UINT)
ON_WM_VKEYTOITEM( )	afx_msg int OnVKeyToItem (UINT, CWnd*, UINT)
ON_WM_VSCROLL( )	afx_msg void OnVScroll(UINT, UINT, CWnd*)
ON_WM_VSCROLLCLIPBOARD( )	afx_msgvoid OnVScrollClipboard (CWnd*, UINT, UINT)
ON_WM_WINDOWPOSCHANGED( )	afx_msgvoid OnWindowPosChanged (WINDOWPOS* lpwndpos)
ON_WM_WINDOWPOSCHANGING( )	afx_msgvoid OnWindowPosChanging (WINDOWPOS*lpwndpos)
ON_WM_WININICHANGE( )	afx_msgvoid OnWinIniChange(LPSTR)

用户定义处理函数

映射入口	函数原型
ON_MESSAGE(<message>, <memberFxn>)	afx_msgLRESULT memberFxn (WPARAM, LPARAM)；
ON_REGISTERED_MESSAGE (<nMessageVariable>, <memberFxn>)	afx_msgLRESULT memberFxn WPARAM, LPARAM)；
ON_THREAD_MESSAGE(<message>, <memberFxn>)	afx_msgvoid memberFxn(UINT, LONG)
ON_REGISTERED_THREAD_MESSAGE (<nMessageVariable>, <memberFxn>)	afx_msgvoid memberFxn (UINT, LONG)；

旧的 MFC 函数

下面的函数在 MFC 6.0 中没有实现：

- CDatabase::InWaitForDataSource
- CDatabase::OnWaitForDataSource
- CDatabase::SetSynchronousMode

- CRecordset::OnWaitForDataSource
- CTabCtrl::GetBkColor
- CTabCtrl::SetBkColor
- CTabCtrl::SetItemExtra
- CWinApp::InitApplication
- CWnd::GetSuperWndProcAddr

CDatabase::InWaitForDataSource

说明

在 MFC4.2 中，CDatabase::OnWaitForDataSource 已经过时了。现在 MFC ODBC 类只使用同步处理。如果要开始异步操作，你必须直接调用 ODBC API 函数 SQLSetConnetOption。更多的信息参见 ODBC SDK 程序员指南中的“异步执行函数”主题。

CDatabase::OnWaitForDataSource

说明

在 MFC4.2 **中，**CDatabase::WaitForDataSource **已经过时了。现在** MFC ODBC **类只使用同步处理。如果要开始异步操作，你必须直接调用** ODBC API **函数** SQLSetConnetOption。**更多的信息参见** ODBC SDK **程序员指南中的“异步执行函数”主题。**

CDatabase::SetSynchronousMode

说明

在 MFC4.2 **中，**CDatabase::SetSynchronousMode **已经过时了。现在** MFC ODBC **类只使用同步操作。如果要开始异步操作，你必须直接调用** ODBC API **函数** SQLSetConnetOption。**更多的信息参见** ODBC SDK **程序员指南中的“异步执行函数”主题。**

> **注意** 尽管 SetSynchronousMode 已经过时，但是如果你使用了这个成员函数，你的代码还是能够被编译。SetSynchronousMode 会产生一个 TRACE 消息，不做任何操作。

CRecordset::OnWaitForDataSource

说明

在 MFC4.2 中，CRecordset::OnWaitForDataSource 已经过时了。现在 MFC ODBC 类只使用同步操作。如果要开始异步操作，你必须直接调用 ODBC API 函数 SQLSetConnetOption。更多的信息参见 ODBC SDK 程序员指南中的“异步执行函数”主题。

CTabCtrl::GetBkColor

说明

GetBkColor 和 SetBkColor 成员函数对 CTabCtrl 类不再有效。这些成员函数在 MFC 中没有实现，因为它们依赖的 Windows95 的 TCM_SETCOLOR 消息没有实现。

请参阅 Win32 程序员参考中的 COLORREF

CTabCtrl::SetBkColor

说明

SetBkColor 和 GetBkColor 成员函数对 CTabCtrl 类不再有效。这些成员函数在 MFC 中没有实现，因为它们依赖的 Windows95 的 TCM_SETCOLOR 消息没有实现。

请参阅 Win32 程序员参考中的 COLORREF

CWinApp::InitApplication

说明

在 MFC 中，CWinApp::InitApplication 成员函数已经过时了。你可能在 InitApplication 中做的初始化工作应该被移到 InitInstance 中。如果你重载了 CWinApp::InitApplication，并且你没有调用基类函数，你将会漏出一个 CWinApp::AddDocTemplate 加入的 CDocTemplate 对象。

请参阅 CWinApp::InitInstance

CWnd::GetSuperWndProcAddr

说明

这个函数已经过时。你不需要重载这个函数，因为 CWnd 的缺省实现将这个指针保存在所有的 CWnd 对象中。