软件开发文集

第十辑

《软件开发文集》编委会 编

- 用 MFC 编写 Windows 95 程序(V)
  ——菜单、工具条和状态条
- OLE 和 COM 如何解决部件软件设计问题(上)
- 关于 C++ 的问题解答

科学出版社

软件开发文集

第十辑

《软件开发文集》编委会　编

科学出版社
1997

内 容 简 介

本书是为软件开发人员和计算机用户编写的《软件开发文集》系列书第十辑。本书向读者提供了开发技术规范、软件开发管理、开发平台、应用设计策略等方面的技术资料。本书的重点文章是：用 MFC 编写 Windows 95 程序(V)　菜单、工具条、状态条和 OLE 和 COM 如何解决部件软件设计问题(上)，关于 C++的问题解答等。

本书适用于软件开发人员、计算机用户学习和参考。

图书在版编目(CIP)数据

软件开发文集　第十辑/《软件开发文集》编委会编.
-北京：科学出版社，1997.2
ISBN 7-03-005800-3

Ⅰ.软…　Ⅱ.软…　Ⅲ.软件开发-文集　Ⅳ.TP311.52-53

中国版本图书馆 CIP 数据核字(96)第 25671 号

科学出版社出版
北京东黄城根北街 16 号
邮政编码：100717
北京管庄永胜印刷厂印刷
新华书店北京发行所发行　各地新华书店经售
*
1997 年 2 月第　一　版　开本：787×1092　1/16
1997 年 2 月第　一次印刷　印张：6
印数：1－3000　字数：120 000

定价：9.00 元

编委会名单

顾　问

杜家滨　冯玉琳　秦人华　林资山

主　编

郑茂松

副主编

（按姓氏笔画为序）

刘晓融　李　浩　廖恒毅

编　委

（按姓氏笔画为序）

王淑兰　巴建芬　华京梅　查良钿

编者的话

近年来，我国的计算机产业发展迅速，PC 机销量已居世界第六位。同时，我国有一批人数众多、技术水平相当不错的软件开发人员，他们分布在全国各地、各个行业、各个领域。然而，由于计算机产业，特别是软件产业充满了变化，极富动态性，加之，我国幅员辽阔，通信技术又落后于较发达的国家，致使，软件开发人员面临着这样的问题：了解最新技术动态不及时、不方便，获取最新技术信息比较困难，所得到的信息往往不够全面、完整和深入，相互之间缺乏交流。因此，重复开发、重复别人走过的弯路的情况屡见不鲜。另一方面，全世界积累的资料浩如烟海，价格亦非常昂贵，鉴于国内经济承受能力有限，许多技术信息不能及时获得。有鉴于此，我们组织有关人员翻译、编写了以技术专集为特色的《软件开发文集》系列书，该套书将陆续出版、发行，及时向用户提供最新技术资料，力图为解决上述问题做一点尝试。

《软件开发文集》系列书的宗旨是为软件开发人员开辟一个相互交流经验和体会的园地，提供一条了解新技术、新工具、新产品的渠道，设立一个探讨软件开发理论和开发技术的论坛，帮助软件开发人员更快、更直接地获得有关软件开发的信息，提高软件开发人员的技术水平和工作效率，充分发挥软件的作用，提高软件的使用价值，促进我国软件产业与世界同步发展。

《软件开发文集》是面向软件开发人员的中、高档次的技术专集，所收集的文章向软件开发人员提供了最新科技动态，以及开发技术规范、开发经验和技巧、软件开发管理、应用设计策略、开发平台方面的内容等，同时，根据用户在软件使用和开发过程中遇到的难点、疑点给予一定的解答和帮助。《软件开发文集》资料主要来源于微软公司的“Microsoft System Journal”，“Developer News”，“Microsoft Development Library”，“TechNet”等，并收入国内软件开发人员撰写的有关文章。欢迎国内广大软件开发人员为《软件开发文集》撰稿，介绍自己的经验、心得、体会。特别要注重文章内容适合我国国情。

目前，在国内面向计算机最终用户的图书、杂志丰富多彩，但面向软件开发人员的图书、杂志却寥寥无几。我们期望《软件开发文集》能够弥补这方面的空白，并且不辜负广大用户的期望，把握好学术方向，确定好信息服务和技

术支持的层次和深度，把《软件开发文集》办成深受广大用户喜爱的丛书

由于时间仓促，《软件开发文集》难免存在这样或那样的问题，敬请广大的用户及读者提出宝贵意见和建议，以利改进，为我国软件产业的发展作出应有的贡献。

《软件开发文集》编委会

目　录

1

用 MFC 编写 Windows 95 程序(V)

——菜单、工具条和状态条

下拉菜单可能是世界上用得最广的用户界面元件。任何坐在计算机前的人，看到一个菜单都知道单击菜单条中的一个菜单项会显示出一列命令。

在一个基于 Windows 95 的应用中，菜单经常在资源模板(resource template)中另外定义，并在运行时装入，当然，菜单也可以在程序中以编程方式创建。当一个菜单项被选中时，Windows 95 给菜单的宿主窗口发一个 WM-COMMAND 消息，该消息的 WPARAM 的低位字记录菜单项的整数标识符在 MFC 应用中，相应的成员函数通过消息映射被自动调用。MFC 应用框架提供一个方便的机制使菜单项与应用中保持的状态信息同步，还能在文档/视图应用中向不同对象间发送其各自的命令消息，这种机制进一步加强了 MFC 的菜单消息映射机制。而且，从 MFC 的 CToolBar 类和 CStatasBar 类创建的各种对象可以方便地与菜单集成在一起，从而可以用按钮代替普通的命令和菜单项的描述帮助文本。如果你用 MFC 方式创建过菜单，就不会再想用老方法做了。

本章讨论在 MFC 环境中和应用框架中怎样处理菜单的几方面的内容：创建菜单和处理菜单项选择，更新、修改菜单项的处理函数，提供菜单项的键盘快捷键(加速键)，使用工具条和状态条增强用户界面，创建单击鼠标右键的文本菜单。下面开始第一步的讨论：创建一个基本菜单，并使之与应用相连。

(一) 菜单和菜单资源

定义菜单最简单的方法是在应用程序的 RC 文件中增加一个菜单模板。编译后，模板被编译成二进制资源并链接到 EXE 文件中，可以用资源名或资源标识符对它进行引用。

当用 Visual C++创建一个应用的框架时，AppWizard(应用程序向导)创建一个类似图 1 的菜单模板。IDR_MAINFRAME 是资源的整型标识符，用户可以使用自己喜欢的字符串或整数标识符。PRELOAD 和 DISCARDABLE 定义资源的属性。

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N"              ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",         ID_FILE_OPEN
        MENUITEM "&Save\tCtrl+S",            ID_FILE_SAVE
        MENUITEM "Save &As…",                ID_FILE_SAVE_AS
```

```
        MENUITEM SEPARATOR
        MENUITEM "Recent File"                  ID_FILE_MRU_FILE1,GRAYED
        MENUITEM SEPARATOR

        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "CU&t\tCtrl+X"                 ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",               ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",              ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar"                  ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About MyApp…",               ID_APP_ABOUT
    END
END
```

图1　简单的菜单模板

BEGIN 和 END 之间的语句定义菜单内容。POPUP 语句说明顶端菜单项(出现在窗口顶端的菜单条里的菜单项)。实际上,每一个顶端菜单项都是一个弹出菜单。MENUITEM 语句说明弹出菜单的菜单项。引号中的字符串定义菜单项的文本,"&"符号后的第一个字母是快捷键。

带有加速键的菜单项通常包含该加速键的文本描述。\t 把菜单项文本与加速键文本分开,中间插入一个制表符使得加速键能对齐。程序员有时用\a 来右对齐加速键文本。图2显示了由图1的模板创建的当 File 菜单拉下时的菜单状态。

ID_XXX 值是命令标识符,紧跟 MENUITEM 语句中的菜单文本后面。当用户选中一个菜单项后,WM_COMMAND 消息的低位字传送该值给属主窗口。每个菜单项必须有唯一的标识符值,从1到 OXDFFF。MFC 程序员习惯在 RESOURCE. H 文件中加入 # define 语句来定义菜单项标识符。其实,MFC 框架已经在 AFXRES. H 头文件中为你定义了图1中的所有菜单项(其他的也一样),你不必自己定义。事实上,如果你既把头文件 AFXRES. H 包含在你的 RC 文件中,又用包含 # define 语句来定义相应的已预定义了的标识符的话,编译时将通不过。为了避免这些冲突,MFC 程序员通常给常规定义的菜单项增加一个 IDM_前缀的标识符。如果你不打算使用任何预定义的标识符,不要在 RC 文件中包含 AFXRES. H 头文件。

当用 MENUITEM 语句定义一个菜单项时,你可以定义菜单项的初始状态。图1中

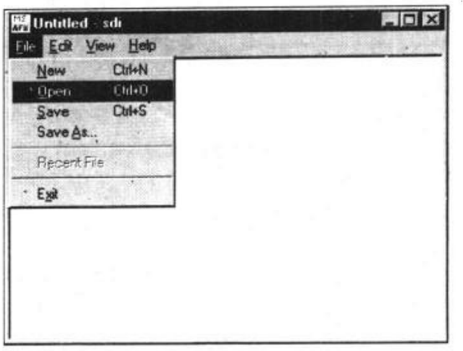

图2　简单的文件菜单

File/PecentFile 菜单项后面的 GRAYED 关键字使菜单项变灰，用户不能选择它。另一个经常使用的关键字是 CHECKED，它在菜单项旁边加一个检查标记。而一般利用 SDK 编写的基于 Windows 的 C 语言程序中，菜单状态说明关键字不常用，因为 MFC 框架的 UI 修改机制能自动处理菜单项的状态。你将学到更多有关 UI 修改机制的知识。

当 AppWizand 应用程序向导为你生成一个缺省菜单的同时，也为它生成了一个加速键表。AppWizard 为图1生成的加速键表如图3所示。图中每一行标识一个加速键或加速组合键。每行行首标识一个虚拟的键码。第二项说明加速键的命令标识符。关键字 VIRTKEY 后面是可选的宏：CONTROL 是 Ctrl 键，SHIFT 是 Shift 键，ALT 是 Alt 键。注

```
IDR_MAINFRAME ACCELERATORS PRELOAD MCVEABLE
BEGIN
    "N",        ID_FILE_NEW,        VIRTKEY,CONTROL
    "O",        ID_FILE_OPEN,       VIRTKEY,CONTROL
    "S",        ID_FILE_SAVE,       VIRTKEY,CONTROL
    "Z",        ID_EDIT_UNDO,       VIRTKEY,CONTROL
    "X",        ID_EDIT_CUT,        VIRTKEY,CONTROL
    "C",        ID_EDIT_COPY,       VIRTKEY,CONTROL
    "V",        ID_EDIT_PASTE,      VIRTKEY,CONTROL
    VK_BACK,    ID_EDIT_UNDO,       VIRTKEY,ALT
    VK_DELETE,  ID_EDIT_CUT,        VIRTKEY,SHIFT
    VK_INSERT,  ID_EDIT_COPY,       VIRTKEY,CONTROL
    VK_INSERT,  ID_EDIT_PASTE,      VIRTKEY,SHIFT
    VK_F6,      ID_NEXT_PANE,       VIRTKEY
    VK_F6,      ID_PREV_PANE,       VIRTKEY,SHIFT
```

图3　简单的加速键表

意，图3中的加速键和图1中的菜单项标识符之间的对应关系，按 Ctrl-C 键等价于在 Edit 菜单中选择 Copy 命令，因为两者具有相同的命令标识符。

（二）装载菜单和加速键

一个菜单在应用的显示之前必须先装载并连接到某个窗口。这一步就可完成用 Create 函数创建窗口时提供菜单标识符。但也可以通过生成一个 CMenu 对象来完成，用

CMenu::Load Menu 载入菜单，然后用 CWnd::Set Wenu 把它连接到一个窗口。下面的语句创建一个基于 MFC 的 CFramWnl 类的有边框窗口，并自动载入和连接资源标识符为 IDR_MAINFRAME 的菜单。

```
Create (NULL,"My Application",WS_OVERLAPPEDWINDOW,
    rectDefault,NULL,
    MAKEINTRESOURCE (IDR_MAINFRAME);
```

类似地，下面的语句单独载入菜单并把它连接到已经存在的有边框窗口：

```
CMenu menu;
menu.LoadMenu(IDR_MAINFRAME);
SetMenu (&menu);
menu.Detach ();
```

第二种技术对于动态装载和链接菜单特别有用。有时你可能想要使用动态菜单，例如，如果你的应用既要支持长菜单又要支持短菜单，此时，要使用动态菜单。如果当调用 SetMenu 时，窗口在屏幕上是可见的，必须调用 CWnd::DrawMenuBar 函数重画菜单来反映菜单的变化。上例中，菜单对象的 Detach 函数被调用了，使菜单对象脱离菜单资源，这样，当菜单越界时，菜单资源不会被毁掉（查看 AFXWINI.INL 文件，你将发现 CMenu 的析构函数调用了 DestroyMenu）。当边框窗口撤销时，菜单资源自动被删除。只要菜单与一个窗口相连，你就不用担心由于某个菜单被遗漏而造成资源被删除。加速键在应用中也必须由应用程序显示装载。给边框窗口装载一个加速键表很简单，调用 CFrameWnd::LoadAccelTable 函数，参数为加速键表的资源描述符。语句：

```
LoadAccelTable(MAKEINTRESOURCE (IDR_MAINFRAME)
```

装入一个 ID 为 IDR_MAINFRAME 的加速键表，并使之与边框窗口相关。在 C 程序中，要激活加速键表，必须修改消息循环，在把消息传给::TranslateMessage 和::Dispatch Message 之前，先调用::TranslateAccelerator 对消息进行预处理。在 MFC 中，CFrameWnd 类重载虚函数 CWnd::PreTranslateMessage，如果有一个加速键表被装入，它将调用::TranslateAccelerator 函数。如果边框窗口对象的 m_hAccelTable 的数据成员包含一个非 NULL 值的加速键表句柄，则说明有一个加速键表被装入。

当装入加速键表的是无边框窗口时，处理有所不同。因为，无边框窗口没有类似于 CFrameWnd 类支持加速键的机制。假如你从 CWnd 类派生出一个定制的窗口类，想装入一个加速键表，可以按照以下步骤做：

·给窗口类增加一个 m-hAccel 数据成员，类型为 HACCEL。

·在应用程序刚刚开始时，在主窗口的构造函数中调用 Create 函数后立刻调用：LoadAccelerators 装入加速键表。把其返回值复制到 m-hAccel 数据成员中。

·重载窗口类的 PreTranslateMessage 函数，调用::Translate Accelerator，参数是存储在 m-hAccel 中的句柄。把其返回值作为 PreTranslateMessage 的返回值，这样当::TranslateAccelerator 已经处理了该消息时不会再对它调用::TranslateMessage 和::DispatchMessage。

下面是代码：

```
// In the CMyWindow constructor...
m_hAccel = ::LoadAccelerators
            (AfxGetInstanceHandle(),
            MAKEINTRESOURCE(IDR_MAINFRAME));

// PreTranslateMessage override
BOOL CMyWindow::PreTranslateMessage(MSG* pMsg)
{
    return ((hAccel!=NULL) &&
        ::TranslateAccelerator(m_hWnd,m_hAccel,
                               pMsg));
}
```

有了这种方法，CWnd 类型的窗口使用加速键时和边框窗口就一样了。注意，用::LoadAccelerators(或::LoadAccelTable)装入的加速键不必在程序中止前删除，因为在程序结束前 Windows 会自动删除它们。

如果查看一个 AppWizand 生成的应用，你并不能发现装载菜单和加速键的代码。这是因为 AppWizard 创建一个文档/视图应用，文档/视图应用使用一个从 MFC 的 CDbcTemplate 类派生出来的文档模板来定义边框窗口、文档和视图之间的关系。在文档/视图应用中，装载菜单和加速键由框架完成。当开始执行应用时，框架调用文档模板的 CreateNewFrame 函数创建一个边框窗口，Create New Frame 函数构造一个边框窗口对象并调用该对象的 Load Frame 函数。LoadFrame 函数创建一个有边框窗口并装载菜单、加速键表、图标和窗口标题，所有这些要一步完成。也可以在非文档/视图应用中使用 LoadFrame 函数。假设 pFrame Wnd 指向一个没有初始化的边框窗口，语句：

```
PFrameWnd->LoadFrame(IDR_MAINFRAME)
```

创建一个边框窗口，把它连接到该边框窗口对象，并自动装载任何标识符为 IDR_MAINFRAME 的菜单，加速键、图标或字符串资源。

（三）命令和消息映射

当用户从菜单中选择一项菜单项或按下一个加速键时，相关的窗口接收到一个 WM_COMMAND 消息，WParam 的低位字保存命令标识符，高位字如果是菜单选择则为0，如果是加速键则为1。MFC 应用把两类事件用 ON_COMMAND 消息映射入口映射到相应类成员函数。下面一个 CFrameWnd 派生窗口类 CMyFrame 的消息映射，它把菜单项 File New 和 File Open(标识符分别为 ID_FILE_NEW 和 ID_FILE_OPEN)分别映射到成员函数 OnFileNew 和 OnFileOpen。

函数名并不重要，你可以随便用什么函数名都行。如上所述，每当从应用的 File 菜单中选择 New 或 Open 菜单项或按下等价的加速键时，CMyFrame::OnFileNew 和 CMyFrame::On File Open 就被调用。

许多 C 程序员习惯于用 Switch-case 结构来处理不同命令标识符的 WM_COMMAND 消息。从这个例子你可以看出，MFC 提供了一个更好的消息结构，即使用消息映射把 WParam 中命令标识符不同的 WM_COMMAND 消息分配到相应的处理函

数。如果 WParam 的定义有所改变的话(Windows 3.1和 Window 95有些不同),只要简单地重新编译就可以修改你的程序,因为 ON_COMMAND 宏也会做出相应的改变。

在一个非文档/视图应用中,菜单消息总是被映射到菜单的属主窗口的成员函数。在文档/视图应用中,菜单消息能在视图类、框架类、文档类,甚至应用类中处理。MFC 框架提供的这个复杂的命令分配系统(Command-routing System)使每个对象都有机会处理来自菜单和控制的消息。当这一系列结束时会介绍文档/视图结构,那时,你将看到命令分配系统到底怎样工作。

(四) 命令范围

有时,用一个成员函数处理某一系列的菜单项的标识符,比单独为每个菜单项标识符都提供一个成员函数更为有效。一个画图应用包含一个颜色菜单,从中用户可选红色、绿色和蓝色。从菜单中选择一种颜色就把该菜单项的标识符赋给成员变量 m_nCurrentColor。然后,改变屏幕上用户所画物体的颜色。下面是菜单的消息映射函数和与这些菜单项相关的成员函数:

```
ON_COMMAND (ID_COLOR_RED,OnColorRed)
ON_COMMAND (ID_COLOR_GREEN,OnColorGreen)
ON_COMMAND (ID_COLOR_BLUE,OnColorBlue)
void CMyFrame::OnColorRed ()
{
    m_nCurrentColor = ID_COLOR_RED;
}
void CMyFrame::OnColorGreen ()
{
    m_nCurrentColor = ID_COLOR_GREEN;
}
void CMyFrame::OnColorBlue ()
{
    m_nCurrentColor = ID_COLOR_BLUE;
}
```

然而,用这种方法来处理来自颜色菜单的消息并不很有效,因为,每个消息函数所做的事情本质上都一样。如果颜色不止3个,而是10个或20个,那就更加低效了。

降低这种菜单处理方式的冗余性的一种方法是把所有的菜单项都映射到同一个 CMyFrame 成员函数,并用 CWnd::GetCurrentMessage 得到菜单项的标识符,如下:

```
ON_COMMAND (ID_COLOR_RED,OnColor)
ON_COMMAND (ID_COLOR_GREEN,OnColor)
ON_COMMAND (ID_COLOR_BLUE,OnColor)
void CMyFrame::OnColor ()
{
    UINT nID = (UINT) LOWORD
            (GetCurrentMessage ()->wParam);
    m_nCurrentColor = nID;
```

这种方法行得通，但由于它依赖于 wParam 的值，所以并非是一个很完美的解决方案。

一个更佳的解决方案是使用 MFC 的 ON_COMMAND_RANGE 宏，该宏把某一系列的标识符都映射到一个普通的成员函数，并把相应的标识符传给该成员函数作为参数。现在，假设 ID_COLOR_RED，ID_COLOR_GREEN，ID_COLOR_BLUE 是三个连续值，并且，ID_COLOR_RED 和 ID_COLOR_BLUE 分别是最小值和最大值，例如，ID_COLOR_RED=100，ID_COLOR_GREEN=101，ID_COLOR_BLUE=102，你可以用下面的代码重写颜色菜单。

```
ON_COMMAND_RANGE (ID_COLOR_RED,ID_COLOR_BLUE,OnColor)
void CMyFrame::OnColor (UINT nID)
{
    m_nCurrentColor = nID;
}
```

当用户选择一种颜色并调用 CMain Window::OnColor 函数时，nID 保存的值是区别不同选择的 ID_COLOR_RED，ID_COLOR_GREEN 或 ID_COLOR_BLUE，简单的一条语句便完成了给 m-nCurrentColor 正确赋值的任务，而不要管到底选择的是哪一项。

（五）修改处理函数

在许多应用中，菜单项要经常修改，以便反映应用的不同状态。例如，在颜色菜单中，选择一种颜色可能要在相应的菜单项旁边加上一个选中标志；在 Edit 菜单中，当没有选择文本或数据时，Cut 和 Copy 菜单项通常是禁止的，而 Paste 菜单项只能当剪贴板上有东西才被激活。

C 程序员通常有两种方法来设置菜单项的状态：要么，当相关的动作发生时立刻改变菜单项；要么，捕获 WM_INIT MENUPOPUP 消息，该消息表明将弹出一个弹出菜单，修改菜单中的所有菜单项。MFC 提供了一个更为简单的（独立于平台）机制来修改菜单项。在消息映射中增加一个 ON_UPPATE_COMMAND_UI 宏，你可以给每个菜单项设计相应的更新成员函数。当用户拉下菜单时，产生 WM_INITMENUPOPOP 消息，框架捕捉到该消息并调用对应于菜单中各菜单项的所有修改函数。框架传给每个修改函数一个指向 CCmdUI 对象的指针，该对象代表产生修改消息的菜单项，该对象的成员函数被用于做适当修改。而且，因为 CCmdUI 并非某种类型的用户界面元素专用，同样的修改函数可用于菜单、工具条和其他 UI 对象。

使用 ON_UPDATE_COMMAND_UI 消息映射入口给不同标识的对象赋于不同的修改函数。图4所示的消息映射可以命令处理函数和修改函数用于一个应用中的 Edit Copy 菜单项，该应用允许在一个名为 m_edit 的 CEdit 对象（编辑控制）中输入文本。一旦 Edit 菜单被拉下，OnUpdateEditCopyUI 函数便调用 CEdit::Get Sel 函数来判断当前是否有文本被选择。如果有，即 if nStart!=nEnd，就调用 CCmdUI::Enable 函数使 Copy 菜单项有效，如果没有，即 nstart=nEnd，则禁止该菜单项。

```
BEGIN_MESSAGE_MAP (CMyFrame,CFrameWnd)
        ON_COMMAND (ID_EDIT_COPY,OnEditCopy)
        ON_UPDATE_COMMAND_UI (ID_EDIT_COPY,OnUpdateEditCopyUI)
                                        .
                                        .
                                        .
END_MESSAGE_MAP
void CMyFrame::OnEditCopy ()
{
    m_edit.Copy ();
}
void CMainWindow::OnUpdateEditCopyUI (CCmdUI* pCmdUI)
{
    int nStart,nEnd;
    m_edit.GetSel (nStart.nEnd);
    pCmdUI->Enable (nStart !=nEnd);
}
```

图4 Edit Copy 菜单项

Enable 函数是 CCmdUI 类用于修改菜单项的四个常用成员函数之一，其余的成员函数在表1中列出。SetCheck 函数依据其参数是非0值或0(缺省为0)选中或选菜单项。SetText 函数改变菜单项的文本，其参数是一个指向常规字符串的指针或一个 Cstnng 对象。SetRadio 函数用来给一系列互斥的菜单项作单一选中标志，它依赖于是否读给它参数或读给它的参数是0还是非零(单选)；而 SetCheck 相当于是一个 ON/OFF 开关(多选)。

表1 用于菜单项修改的 CCmdUI 成员函数

函 数	描 述
Enable	激活或禁止一菜单项
SetCheek	复选或去选一菜单项(用复选标志)
Setkadio	复选或去选一菜单项(用圆钮标志)
SetText	设置菜单项的文本

正如 ON_COMMAND_RANGE 函数能用来把一定范围的命令标识符映射到单个处理函数一样，ON_UPDATE_COMMAND_UI_RANGE 函数也能用来把一定范围的命令标识符映射到一个普通的修改处理函数。下面是刚才讨论的颜色菜单的命令和修改处理函数：

```
ON_COMMAND_RANGE (ID_COLOR_RED,ID_COLOR_BLUE,
                        OnColor)
ON_UPDATE_COMMAND_UI_RANGE (ID_COLOR_RED,
                                ID_COLOR_BLUE,
```

```
                                        OnUpdateColorUI)
void CMyFrame::OnColor (UINT nID)
{
    m_nCurrentColor = nID;
}
void CMyFrame::OnUpdateColorUI (CCmdUI * pCmdUI)
{
 pCmdUI->SetRadio (pCmdUI->m_nID==m_nCurrentColor);
}
```

m_nID 是 CCmdUI 类的一个公共数据成员，用来保存菜单项的标识符，处理函数中比较 m_nID 和 m_nCurrentColor 的值，比较结果作为 SetRadio 的参数，确保只复选当前颜色。

（六）工具条

近年，越来越多的应用程序使用工具条。使用 CToolBar 类，MFC 可以很方便地制作工具条。CToolBar 类封装了一个包含有图标的按钮的工具条窗口。每个按钮有一个命令标识符，单击一个按钮产生一条 WM_COMMAND 消息，这和菜单项一样。工具条按钮经常与菜单项赋有相同的标识符，这样，同一个成员函数两者都能使用。

相同标识符的工具条按钮和菜单项也共享更新函数。修改一个菜单项时，MFC 框架调用处理函数，参数是一个指向该菜单项的 CCmdUI 对象，修改一个工具条按钮时，MFC 框架也调用该处理函数，参数是一个指向该按钮的 CCmdUI 对象，因此，用来激活或禁止（看当前状态）Edit Copy 菜单项的代码也可用来激活或禁止工具条中的 Copy 按钮。唯一的差别是修改菜单项时是在每次显示菜单时，而修改工具条按钮是在每个空闲消息循环——此时消息队列是空的。

Windows 95在普通控制库中提供自己的实现工具条的方法。在 MFC 4.0中，MFC 框架将自动使用 Windows 95提供的工具条而不是自己为 CToolBar 对象实现的私有工具条。因此，使用 CToolBar 类可以很方便地在应用中使用 Windows 95工具条，而不必学习更加复杂的新的 API 机制和消息机制。

CToolBar 类用处很大，是因为在应用中增加一个工具条，程序员不需做什么工作。例如，只要给工具条按钮提供一个带几幅图像的位图，在按钮被禁止或释放时 MFC 将产生代表不同按钮各种位图，只要提供一列命令标识符，MFC 框架将自动生成各种按钮，想增加一些当光标在按钮上停留半秒钟左右时，应用会给出提示信息，只要在 RC 文件中增加一个字符串表就可以，若想让工具条能停靠在任何地方，只要调用工具条的 EnableDocking 函数和边框窗口的 EnableDocking 函数，MFC 将自动处理。工具条还能和其他控制（如组合框）定制在一起，只要稍做处理就行。

工具条和边框窗口在一起能很好地工作，因为 MFC 的 CFrameWnd 类和 CToolBar 类是设计成在一起工作的，典型方法是在一个边框窗口中用 OnCreate 函数创建一个工具条，下面的语句创建一个工具条，装载描述工具条按钮的位图，设定按钮的标识符。

```
static UINT nButtonIDs [] = {
```

```
    ID_FILE_NEW,
    ID_FILE_OPEN,
    ID_FILE_SAVE,
    ID_SEPARATOR,
    ID_EDIT_CUT,
    ID_EDIT_COPY,
    ID_EDIT_PASTE,
    ID_SEPARATOR,
    ID_FILE_PRINT,
    ID_APP_ABOUT,
};
m_toolbar.Create (this);
m_toolbar.LoadBitmap (IDR_TOOLBAR);
m_toolbar.SetButtons (nButtonIDs,10)
```

m_toolbar 是边框窗口类的一个成员，因此，在有边框窗口被创建时，工具条被自动创建，框架窗口撤销时工具条也随之撤销。这个工具条包括8个按钮和2个按钮分隔符，按钮分隔符用来把几组按钮分开几个像素的距离。Create 函数创建工具条窗口；LoadBitMap 函数装载一个带8幅图像的位图资源，每幅图像16像素宽，15像素高；SetButtons 函数给不同按钮赋标识符，按钮标识符保存在 nButtonIDs 数组中，ID_SEPARATCR 是用来标识分隔符及专用标识符。结果如图5所示：

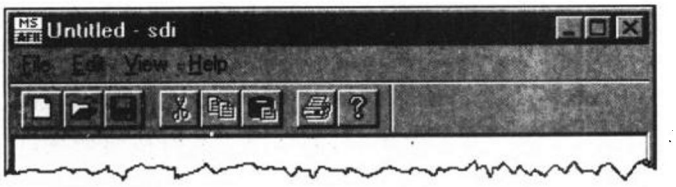

图5　工具条位图

CTooBar 类提供了各种成员函数，用来定制工具条的外表和行为并得到各种工具条的信息。例如，CToolBar::SetSizes 函数用来改变图像尺寸和按钮大小，图像缺省大小是16×15，按钮缺省大小是24×22。给定一个基于0的按钮索引后 CToolBar::GetItemID 返回按钮的命令标识符。查看 CToolBar 类通过成员函数提供的各种功能时，别忘了看看基类 CControlBar。在 CControlBar 类中你将发现一些方便易用的函数，如 EnableDocking——用来说明把工具条放在什么地方，SetBarStyle 函数——用来改变工具条的风格属性。在 CControl 类中，还将发现一个公共数据成员 m_bAutoDelete，当它为 TRUE 时，使得在工具条窗口撤销时，相应的 CToolBar 对象自动被删除。

表2列出了 CToolBar 类和 CControd 类提供的各种成员函数。

给工具条增加提示信息很简单。在创建工具条时，只要包括 CBRS_TOOLTIPS 风格就可以。或在在工具条创建后用 GetBarStyle 和 SetBarstyle 函数来增加 CBR_TOOLTIPS 类型：

```
m_toolbar.SetBarStyle(m_toolbar.GetBarStylel)
                    |CBRS_TOOLTIPS);
```