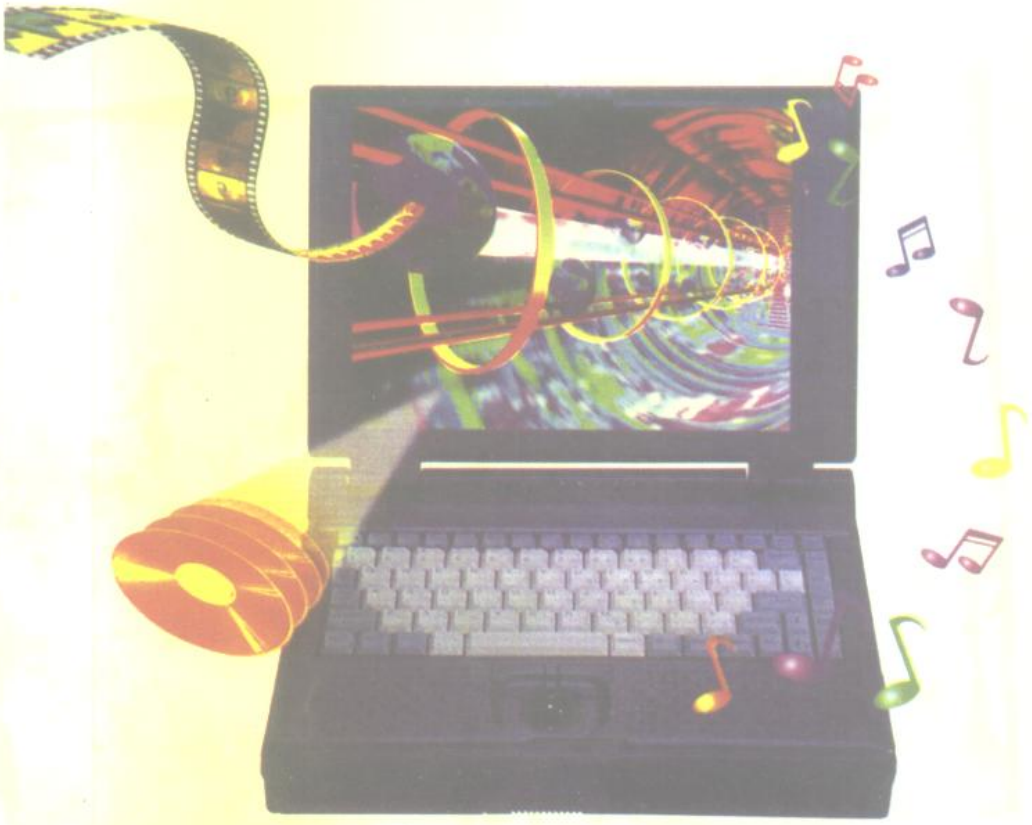
# 软件开发文集

## 第九辑

《软件开发文集》编委会 编



- 用 MFC 编写 Windows 95 程序(IV)——键盘输入处理
- 揭开 Microsoft Visual J + + 的秘密——第一个支持 COM 对象的 Java 开发环境
- Internet 部件卸载服务使安全的 Web 再现辉煌
- 学习使用 ODBC

科学出版社

406672

# 软件开发文集

第九辑

《软件开发文集》编委会　编

科学出版社

1997

TS/99/31

## 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》系列书的第九辑，围绕专题内容，向读者提供了应用设计策略、开发技术规范、开发管理、开发平台等方面的技术资料。

本辑的重点文章是：用 MFC 编写 Windows 95 程序(IV)——键盘输入处理，揭开 Microsoft Visual J++的秘密——第一个支持 COM 对象的 Java 开发环境，Internet 部件卸载服务使安全的 Web 再现辉煌，学习使用 ODBC。

本书适用于软件开发人员和广大计算机用户。

**图书在版编目(CIP)数据**

软件开发文集　第九辑/《软件开发文集》编委会编. -北京：科学出版社，1997.1

ISBN 7-03-005620-5

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 19645 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 1 月第　一　版　　开本：787×1092　1/16

1997 年 1 月第一次印刷　　印张：5 1/2

印数：1－3000　　字数：110 000

**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

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



此为试读，需要完整PDF请访问：www.ertongbook.com

# 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。

# 目　录

# 1

# 用 MFC 编写 Windows 95 程序(IV)

## ——键盘输入处理

无论是鼠标还是键盘，总是通过消息对应用程序通知输入事件。在 MFC 场境和 MFC 应用程序框架中我们考查了鼠标输入，这次我将集中介绍键盘。和鼠标一样，键盘也是一个全局硬件资源，必须为所有程序共享。Windows 95 决定鼠标消息发送到哪个窗口，这主要是由哪个窗口在鼠标下决定的。键盘消息直接发送到具有输入焦点的窗口。具有输入焦点的窗口通常是活动的应用程序的主窗口，也可是主窗口的子窗口或对话框中的子窗口，如果你的应用程序中没有子窗口，键盘处理就相对简单。如果输入焦点转移到一个子控制上，到主窗口的键盘消息就会暂停。程序员通常通过确保输入焦点停留在主窗口上来避免这个问题(有时必须使用一些非传统的方法，如细分子窗口或创建一个消息钩子来监视消息流，这两种方法都超出了本书的讨论范围)。

Windows 通过 WM_SETFOCUS 和 WM_KILLFOCUS 两个消息来通知窗口将接受或失去输入焦点，这两个消息在 MFC 应用程序中由 OnSetFocus 和 OnKillFocus 处理程序处理。应用程序可以通过::SetFocus API 函数将输入焦点移到另一个窗口。这个函数以即将接受输入焦点的窗口句柄作为它的一个参数。

```
::SetFocus (hwnd);
```

::GetFocus API 函数和它的 MFC 对等函数 CWnd::GetFocus，标认出具有输入焦点的窗口。在 Win32 中，如果当前具有输入焦点的窗口和当前线程不相连，这两个函数都返回 NULL。

### (一)击键消息

Windows 通过发送 WM_KEYDOWN 和 WM_KEYUP 击键消息给具有输入焦点的窗口，来向应用程序报告按下和放开键的事件。如果在有一个键被按住的情况下又有一个键被按下，一组 WM_KEYDOWN 和 WM_KEYUP 消息会将另外一组 WM_KEYDOWN 和 WM_KEYUP 消息分开，表示被按住的键的按下和释放过程。Windows 按照发生的顺序来报告键盘事件，因此通过检查击键消息到来的顺序，可以精确地知道用户输入的时间。

如果你想知道应在什么时候按下 Page Up 或 Page Down 键的键码，以便你的应用程序能做出反应，则处理 WM_KEYDOWN 消息和检查标识 Page Up 和 Page Down 键的键码；如果你想知道这个键是什么时候放开的，你可以改为处理 WM_KEYUP 消息。对于大多数的键，你可以忽略按下和放开消息，而让 Windows 给你发送字符消息来指示用户的

输入(在后面你将会看到,如果你使用的应用程序对象是从 CWinApp 派生来的,这种情况会自动发生)。使用字符而不使用原始击键消息可以简化处理,这是由于 Windows 可以为你处理一些事件和与击键有关的环境,如 Shift 键是否按下,Caps Lock 是 On 还是 Off 状态,以及不同的键盘布局等。

除了两个键外,每个键都产生 WM_KEYDOWN 和 WM_KEYUP 消息,这两个键是 Alt 和 F10,它们属于系统键。当这两个键被按下和释放时,窗口将接受 WM_SYSKEYDOWN和 WM_SYSKEYUP 消息,而不是非系统击键消息。如果在按下 Alt 时,又按下其他的键,它们也会产生系统击键消息。

应用程序可以通过对感兴趣消息提供消息处理程序来处理击键消息或通过消息映射在处理循环中钩住它们的方法来实现。WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN和 WM_SYSKEYUP 消息可以用一个类的 OnKeyDown, OnKeyUp, OnSysKeyDown 和 OnSysKeyUp 成员函数来处理。当激活时,这些处理程序接受关于击键的信息,包括区分每个键是按下或释放的编码。消息处理程序的原型如下:

```
void OnMsgName(UINT nChar,UINT nRepCnt,UINT nFlags)
```

nChar 区分键是按下还是释放,nRepCnt 是重复次数,即消息中编码的击键次数。对于 WM_KEYDOWN 或 WM_SYSKEYDOWN 消息来说 nRepCnt 通常是 1,对于 WM_KEYUP或 WM_SYSKEYUP 总是 1。如果键按下消息来得太快,你的应用程序跟不上的话,Windows 将两个和多个 WM_KEYDOWN 或 WM_SYSKEYDOWN 消息合并成一个,同时增加重复次数。大部分程序忽略重复次数,将组合的键按下消息(重复次数大于 1)作为一个击键消息处理,可以避免一个程序连续滚动的过载情形,或者在用户手指离开键之后响应击键消息。和 PC 键盘 BIOS 不同,它是缓冲到来的击键,然后单独报告每个击键,Windows 键盘处理逻辑提供一种内在的保护,以防止键盘溢出。

传递给击键消息处理程序的 nFlags 参数包含这个键的 8 位 OEM 扫描码和一系列位标记,用于区分转变状态、以前键状态、上下文码,以及这个键是否是扩展键(见表 1)。扩展键标记允许应用程序区分许多键盘上存在的相同键。绝大多数 PC 上使用 101 和 102 键盘,这个标记为键盘右边的 Ctrl 和 Alt 键,主键盘和数字小键盘的 Home, End, Insert, Delete, PageUp, PageDown 和箭头键,以及小键盘的 Enter 和斜线(/)键设置。对于所有其他的键,扩展键标记总是 0。OEM 扫描码是 8 位值,键盘 BIOS 用来区分键。绝大多数应用程序忽略这个域,因为键通常用它们的虚拟键值来区分(如需要,可用 ::MapVirtualKey

**表 1 传送给击键消息处理程序的 nFlags 参数**

| 位 | 意 义 | 说 明 |
|---|---|---|
| 0~7 | OEM 扫描码 | 键盘硬件产生的 8 位 OEM 扫描码 |
| 8 | 扩展键标记 | 如果键是扩展键为 1,否则为 0 |
| 9~12 | 保留 | 不可用 |
| 13 | 上下文码 | 如果 Alt 键被按下为 1,否则为 0 |
| 14 | 以前键状态 | 如果这个键以前是被按下为 1,否则为 0 |
| 15 | 转换状态 | 如果这个键被按下为 0,如果放开为 1 |

将扫描码转换成虚拟键值)。转换状态、以前键状态和上下文码通常也被忽略,偶尔这些也是有用的。以前键状态值等于1,用于区分某种复合击键消息:当按下一个键而且按住一段时间后产生的键盘消息。例如,按住Shift键几秒钟,产生一个消息系列,相应的虚拟键值和以前键状态值如表2所示。

**表2 复合击键消息**

| 消 息 | 虚拟键码 | 以前键状态 |
|---|---|---|
| WM_KEYDOWN | VK_SHIFT | 0 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYDOWN | VK_SHIFT | 1 |
| WM_KEYUP | VK_SHIFT | 1 |

要忽略复合击键动作产生的击键消息,你可以忽略以前键状态值等于1的键按下消息。对于大多数应用程序来说,转换状态值是冗余的,这是因为对于WM_KEYDOWN和WM_SYSKEYDOWN消息它总是0,而对于WM_KEYUP和WM_SYSKEYUP是1。最后,上下文码指示Alt键在消息产生时是否被按下。有一点特殊的地方(通常不重要),对于WM_SYSKEYDOWN和WM_SYSKEYUP消息它是1,而对WM_KEYDOWN和WM_KEYUP它是0。

一般情况下,应用程序不需要处理WM_SYSKEYDOWN和WM_SYSKEYUP消息,而让Windows来处理。如果这些消息没有送到DefWindowProc去处理,系统键组合如Alt-Tab和Alt-Esc将不会工作。Windows将所有的鼠标和键盘消息先发送给你的应用程序,使得你有很大的权力来处理各种消息,虽然有些消息仅仅对操作系统有效。对系统击键消息和非用户区的鼠标消息处理不当(特殊情况),没有将这些消息传送给操作系统,会导致各种奇怪的事情发生。

### (二)虚拟键码

传送给击键消息处理程序的一个重要参数是nChar值,区分这个键是按下还是放开。因此应用程序不需要依赖因键盘不同而不同的硬编码值或OEM扫描码。Windows提供的虚拟键码如表3所示。如果你查看一下WINUSER.H文件,你会发现定义的各种虚拟键码。你可找到下图未列的一些键码,包括VK_SELECT,VK_PRINT,VK_EXECUTE和VK_Fi3到VK_F24。这些键码用于其他的应用平台,在一般的IBM键盘上不会产生。Windows不提供虚拟键码的键主要是基本的字母数字键和符号键,如";"和"]"键,在处理键按下和放开消息时,最好避开,这是因为,这些键的ID在国际键盘上会有所不同。这

不意味着你的应用程序不能处理标点符号和其他没有 VK_xxx 标识存在的字符；它仅仅意味着有一种比依赖按下和放开消息更好的方法来处理。这个较好的方法就是字符消息，在稍后我会讲到。

**表 3　虚拟键码**

| 虚拟键码 | 对应的键 |
|---|---|
| VK_F1 到 VK_F12 | 功能键 F1 到 F12 |
| VK_NUMPAD0 到 VK_NUMPAD9 | 在 Num Lock on 状态时数字键盘 0 到 9 |
| VK_CANCEL | Ctrl-Break |
| VK_RETURN | Enter |
| VK_BACK | Backspace |
| VK_TAB | Tab |
| VK_CLEAR | 在 Num Lock off 状态时数字键盘 5 |
| VK_SHIFT | Shift |
| VK_CONTROL | Ctrl |
| VK_MENU | Alt |
| VK_PAUSE | Pause |
| VK_ESCAPE | Esc |
| VK_SPACE | 空格 |
| VK_PRIOR | Page Up 和 PgUp |
| VK_NEXT | Page Down 和 PgDn |
| VK_END | End |
| VK_HOME | Home |
| VK_LEFT | 左箭头← |
| VK_UP | 上箭头↑ |
| VK_RIGHT | 右箭头→ |
| VK_DOWN | 下箭头↓ |
| VK_SNAPSHOT | 打印屏幕 |
| VK_INSERT | Insert 和 Ins |
| VK_DELETE | Delete 和 Del |
| VK_MULTIPLY | 数字键盘 * |
| VK_ADD | 数字键盘＋ |
| VK_SUBSTRACT | 数字键盘－ |
| VK_DECIMAL | 数字键盘 |
| VK_DIVIDE | 数字键盘/ |
| VK_CAPITAL | Caps Lock(大写锁定) |
| VK_NUMLOCK | Num Lock(数字锁定) |
| VK_SCROLL | Scroll Lock(滚动锁定) |



此为试读，需要完整PDF请访问：www.ertongbook.com

### (三)Shift 状态和切换

当你为 WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN 或 WM_SYSKEYUP 消息写一个消息处理程序时,在决定如何响应之前,你可能需要知道 Shift,Ctrl 和 Alt 键是否被按下。Shift 状态信息,包括 Shift 和 Ctrl 键,没有编入键盘消息中,这一点和鼠标消息不同,因此 Windows 提供::GetKeyState 函数。指定一个虚拟键码,::GetKeyState返回对应的键是否被按下。对于如 Num Lock 这样的切换键,表示该键是否选中。语句:

```
::GetKeyState (VK_SHIFT)
```

如果 Shift 键按下,返回负值,否则非负值。与此类似,语句:

```
::GetKeyState (VK_CONTROL)
```

如果 Ctrl 键按下,返回负值。因此,下面 OnKeyDown 处理程序的代码片断括号中的语句,只有 Ctrl 和左箭头同时按下时才执行:

```
if ((nChar==VK_LEFT)&&
   (::GetKeyState (VK_CONTROL)<0)){
   .
   .
   .
   }
```

要查询 Alt 键状态,你可以简单地检查 nFlags 参数中的上下文代码位或调用::GetKeyState,使用 VK_MENU 参数。通常这些不是必需的,因为如果按下 Alt 键,你的窗口会接受到一个 WM_SYSKEYDOWN 和 WM_SYSKEYUP 消息,而不是 WM_KEYDOWN或 WM_KEYUP 消息。换句话说,另外一种消息可能已经告诉你足够的 Alt 键情况了。你可以使用标识符 VK_LBUTTON,VK_MBUTTON 和 VK_RBUTTON 去决定鼠标左、中或右按钮是否被按下。

应用程序也可以使用::GetKeyState 来判断 Num Lock,Caps Lock 和 Scroll Lock 键是否被选中或未选中。其实,返回代码的高位(高位为 1 时是负值)指示这个键是否按下,低位(为 0)表示切换的状态。下行的代码:

```
::GetKeyState (VK_NUMLOCK)& 0x01
```

如果 Num Lock 选中是真(非零),否则是假(零)。用同样的方法可以测试 VK_CAPITAL (Caps Lock)和 VK_SCROLL (Scroll Lock)键的状态。在测试之前屏蔽掉返回值的除最低位以外的所有位是很重要的,因为高位仍然表示这个键是否被按下。

在各种情况下,::GetKeyState 报告在键盘消息产生时的键或鼠标按钮的状态,而不是在函数被调用时的那一时刻的状态。因此你不必担心你的消息处理程序在查询键状态时,Shift 键已被释放。::GetKeyState 不能在键盘消息处理程序以外调用,因为它返回的信息只有在从消息队列中获取键盘消息以后才是合法的。如果你确实想知道当前键或鼠标按钮的状态,或如果你想在键盘处理程序以外检查键或鼠标按钮状态,你可以使用::GetAsyncKeyState函数。

### (四)字符消息

如果你依赖互斥的键释放和按下键盘输入消息时,在下面的情况你会遇到一些困难。假设你在写一个字符输入程序,它将字符键按下的消息转换成字符显示在屏幕上。A 键按下时,就会有一个 WM_KEYDOWN 消息到来,虚拟键码等于 VK_A。在你将 A 显示在屏幕之前,你要调用::GetKeyState 决定 Shift 是否按下。如果按下,你输出大写的 A;否则,你输出小写的 a。到目前为止,这还是对的。但,如果 Caps Lock 起作用呢? Caps Lock 影响 Shift 键的效果,将 A 转换成 a 或 a 转换成 A。因此,有字母 A 的四个不同的排列需要考虑(见表 4)。

**表 4 字母 A 的四种排列**

| 虚拟键码 | VK_SHIFT | Caps Lock | 结 果 |
|---|---|---|---|
| 0x41 | No | Off | a |
| 0x41 | Yes | Off | A |
| 0x41 | No | On | A |
| 0x41 | Yes | On | a |

你可能希望写一段代码来处理上键和切换的各种可能来克服这些困难。在你将 Ctrl 键也按下时,你的工作会更复杂。当你的应用程序在不同布局的键盘的计算机上运行时,困难会更大。美国用户按住 Shift 和 0 来输入")"符号,而在大多数国际键盘上,Shift-0 会产生"="符号,在荷兰键盘上会产生省略号。如果你的程序显示的字符不符合他们输入的键,用户会感觉不了解。

这就是 Windows 为什么提供 WM_CHAR 消息。这样字符输入可以与键盘类型无关。大多数消息循环中调用的 TranslateMessage 函数在分发一个消息之前,将 WM_KEYDOWN/UP 和 WM_SYSKEYDOWN/UP 消息转换成 WM_CHAR 和 WM_SYSCHAR消息,用来描述一个字符,它考虑了相关的 Shift 状态、切换状态和不同的键盘布局等。通过在 CWinApp 消息循环中调用::TranslateMessage MFC 可以做到这些。当你处理字符消息时,除了没有对应的字符存在的键以外(像功能键、箭头和 Page Up 和 Page Down 键),你不再需要担心虚拟键码和上键状态,取而代之的是,你接收一个 ANSI 码。假设 Caps Lock 没有选中,按 Shift-A 产生消息结果如表 5 所示。现在你可以忽略键按下和释放消息,因为你所需的所有信息都在 WM_CHAR 消息中。在 Alt 键按住时,你的应用程序可能会接收一个 WM_SYSCHAR 消息(见表 6)。

**表 5 按 Shift-A**

| 消 息 | 虚拟键码 | ANSI 码 |
|---|---|---|
| WM_KEYDOWN | VK_SHIFT | |
| WM_KEYDOWN | 0x41 | |
| WM_CHAR | | 0x41(A) |
| WM_KEYUP | 0x41 | |
| WM_KEYUP | VK_SHIFT | |

表6 按 Alt-A

| 消　息 | 虚拟键码 | ANSI 码 |
|---|---|---|
| WM_SYSKEYDOWN | VK_SHIFT | |
| WM_SYSKEYDOWN | 0x41 | |
| WM_SYSCHAR | | 0x41(A) |
| WM_SYSKEYUP | 0x41 | |
| WM_SYSKEYUP | VK_SHIFT | |

由于 Alt 加一个键组合通常用作特殊目的，多数应用程序忽略 WM_SYSCHAR 消息，而用处理 WM_CHAR 消息代之。

MFC 应用程序处理 WM_CHAR 消息，可以通过在窗口或视图类中提供一个 OnChar 成员函数和相关的消息映射入口实现。

```
void OnChar (UINT nChar,UINT nRepCnt,UINT nFlags)
```

nRepCnt 和 nFlags 在击键消息中具有同样的意义。nChar 包含一个 ANSI 字符码。你必须注意的一件事情是，通过 WM_CHAR(或 WM_SYSCHAR)消息显示接收的字符时，你所使用的字体是 ANSI 字体，而不是 OEM 字体。对于字符码从 32 到 127，ANSI 中的字符和 OEM 中的一样。但在这范围以外，字符是不同的。如果用户使用德国键盘输入一个大写 O 和一个变音符号(…)，你的应用程序会接收一个 WM_CHAR 消息，参数 nChar 等于 214。如果你用 ANSI 字体来显示这个字符，会正确显示出来；如果用 OEM 字体来显示，Windows 会显示出来一个奇怪的图形字符。庆幸的是，Windows 选入设备上下文中缺省系统字体是 ANSI 字体。Windows 的大多数其他字体你也可能会遇到。一个例外是 OEM_FIXED_FONT，这是一个固有的字体，你可以使用 CDC::SelectStockObject 来选中它。

在某些事件中，你需要将一个字符串码从 OEM 转换到 ANSI 格式，或进行相反的工作，MFC CString 类提供名为 AnsiToOem 和 OemToAnsi 的函数。OemToAnsi 的一个用途是将用基于 MS-DOS 的应用程序产生的文本文件转换成 ANSI 格式，以便在基于 Windows 的应用程序中正确显示。必须注意，这种转换有时是不太完美的，如果要转换的字符串中的一个字符在另外一个字符集中并不存在，这种情况就会发生，此时，Windows 会尽量用键盘驱动程序提供的转换表来发现一个合理的替代字符。输出结果有可能没有原来的好看。

### (五)插入符

应用程序中用来表示下一个输入字符将要插入的位置的闪烁的垂直杆叫做插入符。MFC CWnd 类提供如表 7 所示的 7 个函数来处理插入符。表中没有的一个基本函数是::DestroyCaret，它必须从 Windows API 中直接调用，因为它没有等同的 MFC 对等函数。

和鼠标光标一样，插入符是一个全局资源。在某一时间，一个插入符只能有一个程序拥有它。因此，一个程序不能按照它所想的任何方法来使用插入符：它必须注意其他程序的需要。使用插入符的规则是简单的。

表 7　CWnd 插入符函数

| 函　数 | 描　述 |
|---|---|
| CreateCaret | 从位图中创建一个插入符 |
| CreateSolidCaret | 创建一个插入符，看起来像一个单色块或线 |
| CreateGrayCaret | 创建一个插入符，看起来像一个灰色块或线 |
| GetCaretPos | 获得当前插入符位置 |
| SetCarePos | 设置插入符位置 |
| ShowCaret | 显示插入符 |
| HideCaret | 隐藏插入符 |

• 使用插入符的窗口在它接收到输入焦点时必须使用创建函数 CreateCaret，CreateSolidCaret 或 CreateGrayCaret 来创建一个插入符。在失去输入焦点时，使用 ::DestroyCaret函数来注销插入符。

• 一旦创建了插入符，在调用 ShowCaret 使它可见之前，它是不可见的。通过调用 HideCaret 可使插入符重新不可见。如果调用 HideCaret 嵌套，必须调用相等数目次数的 ShowCaret 来使它可见。

• 如果在 OnPaint 处理程序之外，在包含插入符的窗口区域画图，必须先隐藏插入符，以避免显示冲突。在绘画结束以后再将插入符显示出来。

• 程序通过调用 SetCaretPos 来移动插入符，Windows 不替你移动插入符。处理送来的键盘消息(可能有鼠标消息)和相对应的处理插入符是你的程序的工作。如需要，可用 GetCaretPos 来获得当前插入符的位置。

下列的消息处理程序创建一个插入符，设置它的位置，在窗口重新得到输入焦点时显示插入符：

```
void CMain Window::OnSetFocus (Cwnd * pWnd)
{
  CreateSolidCaret (2,m_cyChar);
      SetCaretPos(m_pointCaretPos);
  ShowCaret();
}
```

下面的程序处理保存插入符的位置，在窗口失去输入焦点时隐藏并注销插入符：

```
void CMain Window::OnKillFocus (Cwnd * pWnd)
{
  HideCaret ();
  m_pointCaretPos=GetCaretPos();
  ::DestroyCaret();
}
```

在这些例子中，m_cyChar 保存插入符的高度，m_pointCaretPos 保存插入符的位置，在失去输入焦点时，保存到这个变量中去，重新获得焦点时恢复。由于在一个时刻只有一个窗口能拥有输入焦点，键盘消息发送到具有输入焦点的窗口，插入符的这种处理途径保

证拥有键盘的窗口也将拥有插入符。如果所有的程序都这样处理，就不会同时有两个程序竞争插入符的情况发生。

创建插入符的函数有好几个目的：定义插入符的外观和声明拥有它的窗口。插入符实际上是一个位图，因此你可以将它定义成任何形式，只需为 CWnd::CreateCaret 提供一个定制的位图即可。大部分情况下，你会发现使用 CreateSolidCaret 函数比较容易（因为它不需要位图）。CreateSolidCaret 创建一个单色块的插入符，依赖于你怎么指定它的形状，它可以像一个矩形，也可以是一个水平或垂直线，或介于两者之间。在上面的 OnSetFocus 例子中，语句：

```
CreateSolidCaret (2,m_cyChar);
```

创建了一个垂直线插入符，它有两个像素宽，高度等于当前字体（m_cyChar）的高度。使用变宽字体时，这是创建插入符的传统方法。有时有些应用程序也会将垂直线插入符的宽度指定为用::GetSystemMetrics 获得的系统变量如 SM_CXBORDER 等。对于固定间距的字体，你可能会用一个块用做插入符，块的高度和宽度等于一个字符的高度和宽度，就像：

```
CreateSolidCaret (m_cxChar,m_cyChar);
```

在使用比例空间字体时，由于可变的字符宽度，块插入符有可能不太好。除了创建的是一个灰色的插入符而不是单色黑块以外，CWnd 的 CreateGrayCaret 函数和 CreateSolidCaret 类似。插入符的尺寸用逻辑单位来表示，因此在调用插入符创建函数之前，你可以改变映射模式，你指定的尺寸也会随之改变。

上面已经提到过，CWnd::SetCaretPos 可以定位插入符，它有一个 CPoint 对象参数，其中包括新光标位置的 x 和 y 用户区坐标。如果你使用的是固定间距的字体，在文本字符串中定位插入符的位置很简单。将插入符位置和字符宽度相乘，你就可以计算出新 x 偏移量。如果字体是变宽的，你需要做较多的工作。MFC 的 CDC::GetTextExtent 和 CDC::GetTabbedTextExtent 函数可让应用程序计算出比例字体中的字符串宽度，用逻辑单位表示（如果字符串中包括制表符必须使用 GetTabbedTextExtent）。有了给定字符的位置，应用程序就可决定相对应的插入符的位置，方法是调用 GetTextExtent 或 GetTabbedTextExtent 来找到到字符位置的累计宽度。如果字符串 Hello，world 是在由 point 指定的位置显示的，下面的语句将插入符的位置定在 w 后（第8个字符）：

```
Csize size=dc. GetTextExtent("Hello,w",8);
SetCaretPos(CPoint (point. x+size. cx,point. y);
```

GetTextExtent 返回 CSize 对象，它的 cx 和 cy 成员反映字符串的宽度和高度。将插入符定位在 point 右边 cx 单位，也就将插入符放在 world 中的 w 和 o 之间。

如果你使用的是比例字体，插入符的位置会变得稍微复杂一些。你不能使用字符偏移量，在你写 OnLButtonDown 处理程序时，这正是鼠标左按钮按下时的位置。假设你的应用程序中维持一个变量叫 m_nCurrentPos，表示当前字符位置，也将是字符串中下一个输入字符将插入的位置。当左箭头或右箭头按下时，计算新插入符的位置是很简单的：只需减少或增加 m_nCurrentPos，然后调用 GetTextExtent 或 GetTabbedTextExtent，使用新字符位置来计算新偏移量。如果鼠标左按钮在字符串的任意位置按下会怎样。鼠标位置和

m_nCurrentPos 之间没有任何关系，因此你必须使用鼠标位置和字符串开始位置来向后找到这个字符，然后计算出最后插入符的位置。这不可避免地包括一些重复，因为没有 Windows API 函数或 MFC 类的成员函数可以接受字符串和像素偏移量，并返回偏移量位置的字符。幸运的是，自己写这样的一个函数并不太困难，在下一节你可看到它是如何工作的。

### (六)VisualKB 应用程序

让我们把已经讨论过的函数综合起来写一个样本应用程序，这个程序接收键盘的文本输入，在窗口中显示文本，并允许用户执行一些简单的文本编辑功能，包括使用箭头和鼠标移动插入符位置。作为教学演示目的，我们增加一个接收各种消息及这些消息的参数的滚动显示功能，这一点和 Charles Petzold 所著的 Programming Windows 3.1(微软出版社，1992)一书中的 KEYLOOK 程序相类似。这个程序，VisualKB，可演示一些技术，包括处理比例空间文本等。它同样也是一个得心应手的工具，可用来检测从键盘来的消息流和试验特定键和键组合会导致什么消息。

图1示出在程序启动和字母 MFC 输入后 VisualKB 窗口的样子。输入的字符在窗口上部的矩形内显示，相对应的键盘消息在下部的消息矩形内显示。当 SHIFT 键按下和释放时产生第一和最后一个消息。在它们中间，你可以看到由 M，F 和 C 键产生的 WM_KEYDOWN，WM_CHAR 和 WM_KEYUP 消息。注意 WM_CHAR 消息处于 WM_KEYDOWN和 WM_KEYUP 之间，表示字符键按下和释放。消息矩形中消息名右边，VisualKB 显示消息参数。在 nChar 中传递给消息处理程序的 Char 是虚拟键码或字符码，Rep 是 nRepCnt 中的重复次数。Scan 是 nFlags 参数的0位代表的 OEM 扫描码。Ext，Con，Prev 和 Trn 分别代表扩展键标记、上下文代码、以前键状态和转换状态值。VisualKB 同样显示 WM_SYSKEYDOWN，WM_SYSCHAR 和 WM_SYSKEYUP 消息，你可以通过按下 Alt 和一个键的组合来测试。

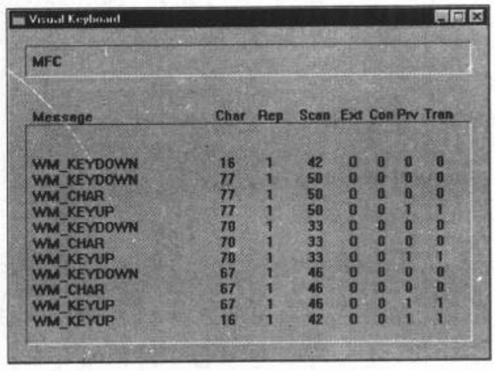



图1 Visual KB

除了输入文本以外，你可以对 Visual KB 使用下述的编辑键：

·左箭头和右箭头将插入符向左或向右移动一个字符。Home 和 End 将插入符移动到行首或行尾。通过单击鼠标左按钮也可以移动插入符的位置。

·Backspace 键将插入符左边的字符删去，并将插入符向左移动一个字符。



此为试读，需要完整PDF请访问：www.ertongbook.com