

键盘和鼠标输入

项目 • 2023/06/13

以下部分介绍捕获用户输入的方法。

本节内容

名称	说明
键盘输入	讨论系统如何生成键盘输入，以及应用程序如何接收和处理该输入。
鼠标输入	讨论系统如何向应用程序提供鼠标输入，以及应用程序如何接收和处理该输入。
原始输入	讨论系统如何向应用程序提供原始输入，以及应用程序如何接收和处理该输入。

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

键盘输入

项目 · 2024/02/06

本部分介绍系统如何生成键盘输入，以及应用程序如何接收和处理该输入。

本节内容

展开表

名称	描述
关于键盘输入	讨论键盘输入。
使用键盘输入	涵盖与键盘输入关联的任务。
键盘输入引用	包含 API 引用。

函数

展开表

名称	描述
ActivateKeyboardLayout	设置调用线程或当前进程的输入区域设置标识符（以前称为键盘布局句柄）。输入区域设置标识符指定区域设置以及键盘的物理布局。
BlockInput	阻止键盘和鼠标输入事件到达应用程序。
EnableWindow	启用或禁用指定窗口或控件的鼠标和键盘输入。禁用输入时，窗口不会接收鼠标单击和按键等输入。启用输入时，窗口会接收所有输入。
GetActiveWindow	检索附加到调用线程消息队列中的活动窗口的句柄。
GetAsyncKeyState	确定调用函数时键是向上还是向下，以及上次调用 GetAsyncKeyState 后是否按下了该键。
GetFocus	如果窗口附加到调用线程的消息队列，则检索具有键盘焦点的窗口的句柄。
GetKeyboardLayout	检索指定线程的活动输入区域设置标识符（以前称为键盘布局）。如果 idThread 参数为零，则返回活动线程的输入区域设置标识符。
GetKeyboardLayoutList	检索与系统中的当前输入区域设置集相对应的输入区域设置标识符（以前称为键盘布局句柄）。该函数将标识符复制到指定的缓冲区。
GetKeyboardLayoutName	检索活动输入区域设置标识符的名称（以前称为键盘布局）。

名称	描述
GetKeyboardState	将 256 个虚拟密钥的状态复制到指定的缓冲区。
GetKeyNameText	检索表示键的名称的字符串。
GetKeyState	检索指定虚拟键的状态。 状态指定键是向上、向下还是切换（每次按键时交替打开、关闭）。
GetLastInputInfo	检索最后一个输入事件的时间。
IsWindowEnabled	确定是否针对鼠标和键盘输入启用指定的窗口。
LoadKeyboardLayout	将新的输入区域设置标识符（以前称为键盘布局）加载到系统中。 一次可以加载多个输入区域设置标识符，但每个进程一次只有一个处于活动状态。 加载多个输入区域设置标识符可以在它们之间快速切换。
MapVirtualKey	将虚拟键代码转换为（映射到）扫描代码或字符值，或将扫描代码转换为虚拟键代码。 若要指定用于转换指定代码的键盘布局的句柄，请使用 MapVirtualKeyEx 函数。
MapVirtualKeyEx	将虚拟键代码映射到扫描代码或字符值，或将扫描代码转换为虚拟键代码。 该函数使用输入语言和输入区域设置标识符转换代码。
OemKeyScan	将 OEMASCII 代码 0 到 0xOFF 映射到 OEM 扫描代码和偏移状态。 该函数提供的信息允许程序通过模拟键盘输入将 OEM 文本发送到另一个程序。
RegisterHotKey	定义系统范围内的热键。
SendInput	合成键击、鼠标动作和按钮单击。
SetActiveWindow	激活窗口。 窗口必须附加到调用线程的消息队列。
SetFocus	将键盘焦点设置为指定的窗口。 窗口必须附加到调用线程的消息队列。
SetKeyboardState	将键盘键状态的 256 字节数组复制到调用线程的键盘输入状态表中。 这是由 GetKeyboardState 和 GetKeyState 函数访问的同一个表。 对此表所做的更改不会影响任何其他线程的键盘输入。
ToAscii	将指定的虚拟键代码和键盘状态转换为相应的一个或多个字符。 该函数使用输入语言和由键盘布局句柄标识的物理键盘布局转换代码。 若要指定用于转换指定代码的键盘布局的句柄，请使用 ToAsciiEx 函数。
ToAsciiEx	将指定的虚拟键代码和键盘状态转换为相应的一个或多个字符。 该函数使用输入语言和由输入区域设置标识符标识的物理键盘布局转换代码。

名称	描述
ToUnicode	将指定的虚拟键代码和键盘状态转换为相应的一个或多个 Unicode 字符。 若要指定用于转换指定代码的键盘布局的句柄，请使用 ToUnicodeEx 函数。
ToUnicodeEx	将指定的虚拟键代码和键盘状态转换为相应的一个或多个 Unicode 字符。
UnloadKeyboardLayout	卸载输入区域设置标识符（以前称为键盘布局）。
UnregisterHotKey	释放以前由调用线程注册的热键。
VkKeyScanEx	将字符转换为相应的虚拟键代码和偏移状态。该函数使用输入语言和由输入区域设置标识符标识的物理键盘布局转换字符。

以下函数已过时。

[+] 展开表

函数	说明
GetKBCodePage	检索当前代码页。
keybd_event	合成键击。系统可以使用这种合成的键击来生成 WM_KEYUP 或 WM_KEYDOWN 消息。键盘驱动程序的中断处理程序调用 keybd_event 函数。
VkKeyScan	将字符转换为当前键盘的相应虚拟键代码和偏移状态。

消息

[+] 展开表

名称	描述
WM_GETHOTKEY	确定与窗口关联的热键。
WM_SETHOTKEY	将热键与窗口相关联。当用户按下热键时，系统会激活窗口。

通知

[+] 展开表

名称	描述
WM_ACTIVATE	同步发送到正在激活的窗口和正在停用的窗口。如果窗口使用相同的输入队列，则消息将同步发送，首先发送到正在停用的顶级窗口的窗口过程，然后发送到正在激活的顶级窗口的窗口过程。如果窗口使用不同的输入队列，则消息将异步发送，因此会立即激活窗口。
WM_APPCOMMAND	通知窗口用户生成了应用程序命令事件，例如，使用鼠标单击应用程序命令按钮或在键盘上键入应用程序命令键。
WM_CHAR	在 TranslateMessage 函数对 WM_KEYDOWN 消息进行转换后发布给具有键盘焦点的窗口。WM_CHAR 消息包含所按的键的字符代码。
WM_DEADCHAR	在 TranslateMessage 函数对 WM_KEYUP 消息进行转换后发布给具有键盘焦点的窗口。WM_DEADCHAR 指定由死键生成的字符代码。死键是与其他字符组合形成复合字符的键，例如元音变音符（双点）。例如，通过键入元音变音符的死键，然后键入 O 键，生成元音变音符 O 字符。
WM_HOTKEY	在用户按下通过 RegisterHotKey 函数注册的热键时发送。此消息放置在与注册了热键的线程关联的消息队列的顶部。
WM_KEYDOWN	按下非系统键时，使用键盘焦点发布到窗口。非系统键是在未按下 ALT 键时按下的键。
WM_KEYUP	非系统键被释放时，发布到具有键盘焦点的窗口。非系统键是未按下 ALT 键的情况下按下的键，或者当窗口具有键盘焦点时按下的键盘键。
WM_KILLFOCUS	在失去键盘焦点之前立即发送到窗口。
WM_SETFOCUS	在获得键盘焦点后发送到窗口。
WM_SYSDEADCHAR	在 TranslateMessage 函数对 WM_SYSKEYDOWN 消息进行转换后，使用键盘焦点发送给窗口。WM_SYSDEADCHAR 指定系统死键的字符代码，即按住 ALT 键时按下的死键。
WM_SYSKEYDOWN	当用户按下 F10 键（这将激活菜单栏）或按住 ALT 键然后按下另一个键时，发布到具有键盘焦点的窗口。当目前没有窗口具有键盘焦点时也会发生这种情况；在这种情况下，WM_SYSKEYDOWN 消息被发送到活动窗口。接收消息的窗口可以通过检查 IParam 参数中的上下文代码来区分这两个上下文。
WM_SYSKEYUP	当用户释放在按住 ALT 键的同时按下的键时，发布到具有键盘焦点的窗口。当目前没有窗口具有键盘焦点时也会发生这种情况；在这种情况下，WM_SYSKEYUP 消息被发送到活动窗口。接收消息的窗口可以通过检查 IParam 参数中的上下文代码来区分这两个上下文。
WM_UNICHAR	在 TranslateMessage 函数对 WM_KEYDOWN 消息进行转换后发布给具有键盘焦点的窗口。WM_UNICHAR 消息包含所按的键的字符代码。

结构

 展开表

名称	描述
HARDWAREINPUT	包含有关由键盘或鼠标以外的输入设备生成的模拟消息的信息。
INPUT	包含用于合成输入事件的信息，例如键击、鼠标移动和鼠标单击。
KEYBDINPUT	包含有关模拟键盘事件的信息。
LASTINPUTINFO	包含最后一个输入的时间。
MOUSEINPUT	包含有关模拟鼠标事件的信息。

常量

 展开表

名称	描述
虚拟键代码	系统使用的虚拟键代码的符号常量名称、十六进制值和鼠标或键盘等效项。 代码按数字顺序列出。

另请参阅

- [关于键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

键盘输入概述

项目 · 2023/10/16

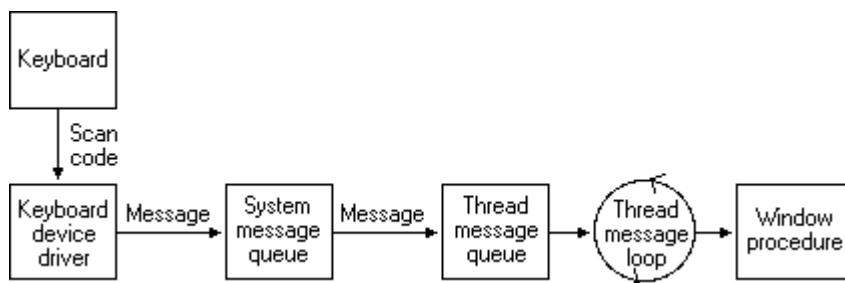
应用程序应接受来自键盘和鼠标的用户输入。 应用程序以发布到其窗口的消息的形式接收键盘输入。

键盘输入模型

系统通过安装适用于当前键盘的键盘设备驱动程序，为应用程序提供与设备无关的键盘支持。 系统通过使用用户或应用程序当前选择的特定于语言的键盘布局，提供与语言无关的键盘支持。 键盘设备驱动程序从键盘接收扫描代码，这些代码将发送到键盘布局，在键盘布局中，扫描代码将转换为消息并发布到应用程序中的相应窗口。

分配给键盘上每个键的唯一值称为扫描代码，是键盘上键的设备相关标识符。 当用户键入某个键时，键盘会生成两个扫描代码，一个在用户按下该键时生成，另一个在用户松开该键时生成。

键盘设备驱动程序解释扫描代码并将其转换（映射）为虚拟键代码，这是一个由系统定义的独立于设备的值，用于标识键的用途。 转换扫描代码后，键盘布局会创建一条消息，其中包含扫描代码、虚拟键代码和有关击键的其他信息，然后将该消息置于系统消息队列中。 系统将该消息从系统消息队列中删除，并将其发布到相应线程的消息队列。 最终，线程的消息循环会删除该消息，并将其传递给相应的窗口过程进行处理。 下图演示了键盘输入模型。



键盘焦点和激活

系统将键盘消息发布到创建具有键盘焦点的窗口的前台线程的消息队列。 键盘焦点是窗口的临时属性。 系统将键盘焦点从一个窗口移动到另一个窗口，从而在显示器上的所有窗口之间共享键盘。 具有键盘焦点的窗口（从创建它的线程的消息队列）接收所有键盘消息，直到焦点更改为其他窗口。

线程可以调用 [GetFocus](#) 函数来确定哪个窗口（如果有）当前具有键盘焦点。 线程可以通过调用 [SetFocus](#) 函数将键盘焦点赋予其中一个窗口。 将键盘焦点从一个窗口更改为另一

个窗口时，系统会向失去焦点的窗口发送 [WM_KILLFOCUS](#) 消息，然后向获得焦点的窗口发送 [WM_SETFOCUS](#) 消息。

键盘焦点的概念与活动窗口的概念相关。活动窗口是用户当前正在使用的顶级窗口。具有键盘焦点的窗口要么是活动窗口，要么是活动窗口的子窗口。为了帮助用户识别活动窗口，系统会将其置于 Z 顺序的顶部并突出显示其标题栏（如果有）和边框。

用户可以通过单击顶级窗口、使用 ALT+TAB 或 Alt+ESC 组合键选择它或从“任务列表”中选择它来激活该窗口。线程可以使用 [SetActiveWindow](#) 函数激活顶级窗口。它可以使用 [GetActiveWindow](#) 函数确定所创建的顶级窗口是否处于活动状态。

当一个窗口停用而另一个窗口被激活时，系统会发送 [WM_ACTIVATE](#) 消息。如果正在停用窗口，则 wParam 参数的低序字为零，如果正在激活窗口，则为非零。默认窗口过程收到 WM_ACTIVATE 消息后，便会将键盘焦点设置到活动窗口。

若要阻止键盘和鼠标输入事件到达应用程序，请使用 [BlockInput](#)。请注意，BlockInput 函数不会干扰异步键盘输入状态表。这意味着在输入被阻止时调用 [SendInput](#) 函数将更改异步键盘输入状态表。

击键消息

按下键会使 [WM_KEYDOWN](#) 或 [WM_SYSKEYDOWN](#) 消息被放入附加到具有键盘焦点的窗口的线程消息队列中。松开键会使 [WM_KEYUP](#) 或 [WM_SYSKEYUP](#) 消息被放入队列中。

按下键和松开键消息通常是相伴发生的，但如果用户长按一个键，以致键盘的自动重复功能启动，则系统会连续生成许多 [WM_KEYDOWN](#) 或 [WM_SYSKEYDOWN](#) 消息。然后，当用户松开键时，它会生成一条 [WM_KEYUP](#) 或 [WM_SYSKEYUP](#) 消息。

本部分涵盖了以下主题：

- 系统和非系统击键
- 描述的虚拟键代码
- 击键消息标志

系统和非系统击键

系统区分系统击键和非系统击键。系统击键会生成系统击键消息 [WM_SYSKEYDOWN](#) 和 [WM_SYSKEYUP](#)。非系统击键会生成非系统击键消息 [WM_KEYDOWN](#) 和 [WM_KEYUP](#)。

如果窗口过程必须处理系统击键消息，请确保在处理消息后，过程将其传递给 [DefWindowProc](#) 函数。否则，只要窗口具有键盘焦点，所有涉及 ALT 键的系统操作都将被禁用。也就是说，用户将无法访问窗口的菜单或系统菜单，或使用 Alt+ESC 或 ALT+TAB 组合键激活其他窗口。

系统击键消息主要供系统而不是应用程序使用。 系统使用它们向菜单提供其内置键盘界面，并允许用户控制哪个窗口处于活动状态。 当用户键入一个键并同时键入 ALT 键时，或者当用户键入且没有窗口具有键盘焦点时（例如，活动应用程序最小化时），将生成系统击键消息。 在这种情况下，消息将发布到附加到活动窗口的消息队列。

非系统击键消息供应用程序窗口使用；[DefWindowProc](#) 函数不对其执行任何操作。 窗口过程可以丢弃它不需要的任何非系统击键消息。

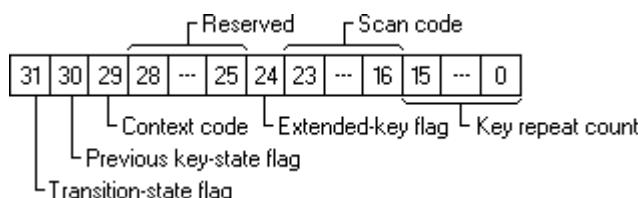
描述的虚拟键代码

击键消息的 wParam 参数包含按下或松开的键的[虚拟键代码](#)。 窗口过程根据虚拟键代码的值处理或忽略击键消息。

典型的窗口过程仅处理它接收到的击键消息的一小部分，而忽略其余部分。 例如，窗口过程可能仅处理 [WM_KEYDOWN](#) 击键消息，并且仅处理那些包含光标移动键、Shift 键（也称为控制键）和功能键的虚拟键代码的消息。 典型的窗口过程不处理来自字符键的击键消息。 相反，它使用 [TranslateMessage](#) 函数将消息转换为字符消息。 有关 [TranslateMessage](#) 和字符消息的详细信息，请参阅[字符消息](#)。

击键消息标志

击键消息的 lParam 参数包含有关生成该消息的击键的附加信息。 此信息包括[重复计数](#)、[扫描代码](#)、[扩展键标志](#)、[上下文代码](#)、[上一个键状态标志](#)和[转换状态标志](#)。 下图显示了这些标志和值在 lParam 参数中的位置。



应用程序可以使用以下值从 lParam 的高位字中获取击键标志。

值	说明
KF_EXTENDED 0x0100	操作 扩展键标志 。
KF_DLGMODE 0x0800	操作对话框模式标志，该标志指示对话框是否处于活动状态。
KF_MENUMODE 0x1000	操作菜单模式标志，该标志指示菜单是否处于活动状态。
KF_ALTDOWN 0x2000	操作 上下文代码标志 。

值	说明
KF_REPEAT 0x4000	操作 上一个键状态标志 。
KF_UP 0x8000	操作 转换状态标志 。

示例代码：

C++

```

case WM_KEYDOWN:
case WM_KEYUP:
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
{
    WORD vkCode = LOWORD(wParam);                                // virtual-key code

    WORD keyFlags = HIWORD(lParam);

    WORD scanCode = LOBYTE(keyFlags);                             // scan code
    BOOL isExtendedKey = (keyFlags & KF_EXTENDED) == KF_EXTENDED; // extended-key flag, 1 if scancode has 0xE0 prefix

    if (isExtendedKey)
        scanCode = MAKWORD(scanCode, 0xE0);

    BOOL wasKeyDown = (keyFlags & KF_REPEAT) == KF_REPEAT;        // previous key-state flag, 1 on autorepeat
    WORD repeatCount = LOWORD(lParam);                            // repeat count, > 0 if several keydown messages was combined into one message

    BOOL isKeyReleased = (keyFlags & KF_UP) == KF_UP;             // transition-state flag, 1 on keyup

    // if we want to distinguish these keys:
    switch (vkCode)
    {
        case VK_SHIFT:   // converts to VK_LSHIFT or VK_RSHIFT
        case VK_CONTROL: // converts to VK_LCONTROL or VK_RCONTROL
        case VK_MENU:     // converts to VK_LMENU or VK_RMENU
            vkCode = LOWORD(MapVirtualKeyW(scanCode, MAPVK_VSC_TO_VK_EX));
            break;
    }

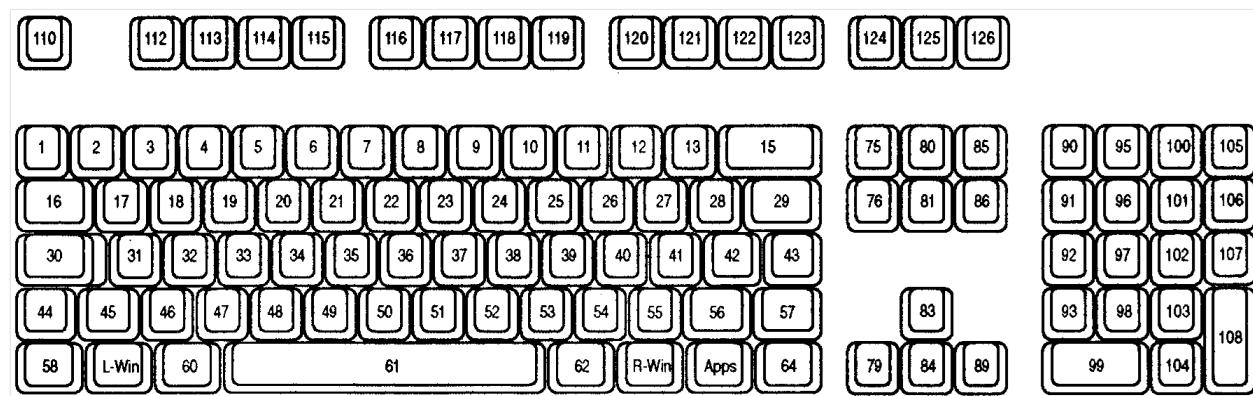
    // ...
}
break;

```

重复计数

可以检查重复计数，以确定一个击键消息是否表示多个击键。当键盘生成 [WM_KEYDOWN](#) 或 [WM_SYSKEYDOWN](#) 消息的速度快于应用程序处理它们的速度时，系统会增加计数。当用户长按某个键，以致键盘的自动重复功能启动时，通常会发生这种情况。系统不会将生成的按键消息填充到系统消息队列中，而是将消息组合成单个按键消息并增加重复计数。松开键无法启动自动重复功能，因此 [WM_KEYUP](#) 和 [WM_SYSKEYUP](#) 消息的重复计数始终设置为 1。

扫描代码



扫描代码是系统在用户按下某个键时生成的值。它是一个值，该值标识按下的键（不考虑活动[键盘布局](#)），而不是由键表示的字符。应用程序通常忽略扫描代码。它使用虚拟键代码来解释击键消息。

新式键盘使用[人机接口设备 \(HID\)](#) 规范与计算机通信。[键盘驱动程序](#)转换从键盘发送的已报告 HID 使用值以扫描代码并将其传递到应用程序。

① 备注

虽然虚拟键代码通常对桌面应用程序更有用，但在需要知道按下哪个键（不考虑当前的[键盘布局](#)）的特殊情况下，可能需要使用扫描代码。例如，用于游戏的 WASD（W 向上、A 向左、S 向下和 D 向右）键绑定，这可确保跨 US QWERTY 或 French AZERTY 键盘布局的一致键结构。

下表列出了 Windows 目前识别的扫描代码集。[HID 使用页面/HID 使用 ID/HID 使用名称](#)值引用 [HID 使用表](#) 文档。键位置值引用前面的键盘图像。

Scan 1 Make 代码在 [WM_KEYDOWN](#)/[WM_KEYUP](#)/[WM_SYSKEYDOWN](#)/[WM_SYSKEYUP](#) 和 [WM_INPUT](#) 消息中传递。

HID 使用页面名称	HID 用法名称	HID 使用页面	HID 用法 ID	Scan 1 Make	键位置
通用桌面	系统断电	0x0001	0x0081	0xE05E	
通用桌面	系统睡眠	0x0001	0x0082	0xE05F	
通用桌面	系统唤醒	0x0001	0x0083	0xE063	
键盘/小键盘	ErrorRollOver	0x0007	0x0001	0x00FF	
键盘/小键盘	键盘 A	0x0007	0x0004	0x001E	31
键盘/小键盘	键盘 B	0x0007	0x0005	0x0030	50
键盘/小键盘	键盘 C	0x0007	0x0006	0x002E	48
键盘/小键盘	键盘 D	0x0007	0x0007	0x0020	33
键盘/小键盘	键盘 E	0x0007	0x0008	0x0012	19
键盘/小键盘	键盘 F	0x0007	0x0009	0x0021	34
键盘/小键盘	键盘 G	0x0007	0x000A	0x0022	35
键盘/小键盘	键盘 H	0x0007	0x000B	0x0023	36
键盘/小键盘	键盘 I	0x0007	0x000C	0x0017	24
键盘/小键盘	键盘 J	0x0007	0x000D	0x0024	37
键盘/小键盘	键盘 K	0x0007	0x000E	0x0025	38
键盘/小键盘	键盘 L	0x0007	0x000F	0x0026	39
键盘/小键盘	键盘 M	0x0007	0x0010	0x0032	52
键盘/小键盘	键盘 N	0x0007	0x0011	0x0031	51
键盘/小键盘	键盘 O	0x0007	0x0012	0x0018	25
键盘/小键盘	键盘 P	0x0007	0x0013	0x0019	26
键盘/小键盘	键盘 Q	0x0007	0x0014	0x0010	17
键盘/小键盘	键盘 R	0x0007	0x0015	0x0013	20
键盘/小键盘	键盘 S	0x0007	0x0016	0x001F	32
键盘/小键盘	键盘 T	0x0007	0x0017	0x0014	21
键盘/小键盘	键盘 U	0x0007	0x0018	0x0016	23
键盘/小键盘	键盘 V	0x0007	0x0019	0x002F	49

HID 使用页面名称	HID 用法名称	HID 使用页面	HID 用法 ID	Scan 1 Make	键位置
键盘/小键盘	键盘 W	0x0007	0x001A	0x0011	18
键盘/小键盘	键盘 X	0x0007	0x001B	0x002D	47
键盘/小键盘	键盘 Y	0x0007	0x001C	0x0015	22
键盘/小键盘	键盘 Z	0x0007	0x001D	0x002C	46
键盘/小键盘	键盘 1 和 !	0x0007	0x001E	0x0002	2
键盘/小键盘	键盘 2 和 @	0x0007	0x001F	0x0003	3
键盘/小键盘	键盘 3 和 #	0x0007	0x0020	0x0004	4
键盘/小键盘	键盘 4 和 \$	0x0007	0x0021	0x0005	5
键盘/小键盘	键盘 5 和 %	0x0007	0x0022	0x0006	6
键盘/小键盘	键盘 6 和 ^	0x0007	0x0023	0x0007	7
键盘/小键盘	键盘 7 和 &	0x0007	0x0024	0x0008	8
键盘/小键盘	键盘 8 和 *	0x0007	0x0025	0x0009	9
键盘/小键盘	键盘 9 和 (0x0007	0x0026	0x000A	10
键盘/小键盘	键盘 0 和)	0x0007	0x0027	0x000B	11
键盘/小键盘	键盘回车 Enter 键	0x0007	0x0028	0x001C	43
键盘/小键盘	键盘 Escape 键	0x0007	0x0029	0x0001	110
键盘/小键盘	键盘 Delete 键	0x0007	0x002A	0x000E	15
键盘/小键盘	键盘 Tab 键	0x0007	0x002B	0x000F	16
键盘/小键盘	键盘空格键	0x0007	0x002C	0x0039	61
键盘/小键盘	键盘 - 和 _	0x0007	0x002D	0x000C	12
键盘/小键盘	键盘 = 和 +	0x0007	0x002E	0x000D	13
键盘/小键盘	键盘 {	0x0007	0x002F	0x001A	27
键盘/小键盘	键盘 }	0x0007	0x0030	0x001B	28
键盘/小键盘	键盘 和 \	0x0007	0x0031	0x002B	29
键盘/小键盘	键盘非美国	0x0007	0x0032	0x002B	42
键盘/小键盘	键盘 ; 和 :	0x0007	0x0033	0x0027	40

HID 使用页面名称	HID 用法名称	HID 使用页面	HID 用法 ID	Scan 1 Make	键位置
键盘/小键盘	键盘' 和 "	0x0007	0x0034	0x0028	41
键盘/小键盘	键盘` 和 ~	0x0007	0x0035	0x0029	1
键盘/小键盘	键盘 ,	0x0007	0x0036	0x0033	53
键盘/小键盘	键盘 .	0x0007	0x0037	0x0034	54
键盘/小键盘	键盘 ?	0x0007	0x0038	0x0035	55
键盘/小键盘	键盘 Caps Lock	0x0007	0x0039	0x003A	30
键盘/小键盘	键盘 F1	0x0007	0x003A	0x003B	112
键盘/小键盘	键盘 F2	0x0007	0x003B	0x003C	113
键盘/小键盘	键盘 F3	0x0007	0x003C	0x003D	114
键盘/小键盘	键盘 F4	0x0007	0x003D	0x003E	115
键盘/小键盘	键盘 F5	0x0007	0x003E	0x003F	116
键盘/小键盘	键盘 F6	0x0007	0x003F	0x0040	117
键盘/小键盘	键盘 F7	0x0007	0x0040	0x0041	118
键盘/小键盘	键盘 F8	0x0007	0x0041	0x0042	119
键盘/小键盘	键盘 F9	0x0007	0x0042	0x0043	120
键盘/小键盘	键盘 F10	0x0007	0x0043	0x0044	121
键盘/小键盘	键盘 F11	0x0007	0x0044	0x0057	122
键盘/小键盘	键盘 F12	0x0007	0x0045	0x0058	123
键盘/小键盘	键盘 PrintScreen 键	0x0007	0x0046	0xE037 0x0054 *注意 1	124
键盘/小键盘	键盘 Scroll Lock 键	0x0007	0x0047	0x0046	125
键盘/小键盘	键盘 Pause 键	0x0007	0x0048	0xE11D45 0xE046 *注意 2 0x0045 *注意 3	126
键盘/小键盘	键盘 Insert 键	0x0007	0x0049	0xE052	75
键盘/小键盘	键盘 Home 键	0x0007	0x004A	0xE047	80
键盘/小键盘	键盘 PageUp 键	0x0007	0x004B	0xE049	85

HID 使用页面名称	HID 用法名称	HID 使用页面	HID 用法 ID	Scan 1 Make	键位置
键盘/小键盘	键盘 Delete Forward 键	0x0007	0x004C	0xE053	76
键盘/小键盘	键盘 End 键	0x0007	0x004D	0xE04F	81
键盘/小键盘	键盘 PageDown 键	0x0007	0x004E	0xE051	86
键盘/小键盘	键盘向右键	0x0007	0x004F	0xE04D	89
键盘/小键盘	键盘向左键	0x0007	0x0050	0xE04B	79
键盘/小键盘	键盘向下键	0x0007	0x0051	0xE050	84
键盘/小键盘	键盘向上键	0x0007	0x0052	0xE048	83
键盘/小键盘	小键盘 Num Lock 和清除键	0x0007	0x0053	0x0045 0xE045 *注意 3	90
键盘/小键盘	小键盘 /	0x0007	0x0054	0xE035	95
键盘/小键盘	小键盘 *	0x0007	0x0055	0x0037	100
键盘/小键盘	小键盘 -	0x0007	0x0056	0x004A	105
键盘/小键盘	小键盘 +	0x0007	0x0057	0x004E	106
键盘/小键盘	小键盘 ENTER	0x0007	0x0058	0xE01C	108
键盘/小键盘	小键盘 1 和 End	0x0007	0x0059	0x004F	93
键盘/小键盘	小键盘 2 和向下键	0x0007	0x005A	0x0050	98
键盘/小键盘	小键盘 3 和 PageDn	0x0007	0x005B	0x0051	103
键盘/小键盘	小键盘 4 和向左键	0x0007	0x005C	0x004B	92
键盘/小键盘	小键盘 5	0x0007	0x005D	0x004C	97
键盘/小键盘	小键盘 6 和向左键	0x0007	0x005E	0x004D	102
键盘/小键盘	小键盘 7 和 Home	0x0007	0x005F	0x0047	91
键盘/小键盘	小键盘 8 和向上键	0x0007	0x0060	0x0048	96
键盘/小键盘	小键盘 9 和 PageUp	0x0007	0x0061	0x0049	101
键盘/小键盘	小键盘 0 和 Insert	0x0007	0x0062	0x0052	99
键盘/小键盘	小键盘 .	0x0007	0x0063	0x0053	104

HID 使用页面名称	HID 用法名称	HID 使用页面	HID 用法 ID	Scan 1 Make	键位置
键盘/小键盘	键盘非美国斜杠	0x0007	0x0064	0x0056	45
键盘/小键盘	键盘应用程序键	0x0007	0x0065	0xE05D	129
键盘/小键盘	键盘电源键	0x0007	0x0066	0xE05E	
键盘/小键盘	小键盘 =	0x0007	0x0067	0x0059	
键盘/小键盘	键盘 F13	0x0007	0x0068	0x0064	
键盘/小键盘	键盘 F14	0x0007	0x0069	0x0065	
键盘/小键盘	键盘 F15	0x0007	0x006A	0x0066	
键盘/小键盘	键盘 F16	0x0007	0x006B	0x0067	
键盘/小键盘	键盘 F17	0x0007	0x006C	0x0068	
键盘/小键盘	键盘 F18	0x0007	0x006D	0x0069	
键盘/小键盘	键盘 F19	0x0007	0x006E	0x006A	
键盘/小键盘	键盘 F20	0x0007	0x006F	0x006B	
键盘/小键盘	键盘 F21	0x0007	0x0070	0x006C	
键盘/小键盘	键盘 F22	0x0007	0x0071	0x006D	
键盘/小键盘	键盘 F23	0x0007	0x0072	0x006E	
键盘/小键盘	键盘 F24	0x0007	0x0073	0x0076	
键盘/小键盘	小键盘 ,	0x0007	0x0085	0x007E	107 *注意 4
键盘/小键盘	键盘 International1	0x0007	0x0087	0x0073	56 *注意 4、5
键盘/小键盘	键盘 International2	0x0007	0x0088	0x0070	133 *注意 5
键盘/小键盘	键盘 International3	0x0007	0x0089	0x007D	14 *注意 5
键盘/小键盘	键盘 International4	0x0007	0x008A	0x0079	132 *注意 5
键盘/小键盘	键盘 International5	0x0007	0x008B	0x007B	131 *注意 5
键盘/小键盘	键盘 International6	0x0007	0x008C	0x005C	

HID 使用页面名称	HID 用法名称	HID 使用页面	HID 用法 ID	Scan 1 Make	键位置
键盘/小键盘	键盘 LANG1	0x0007	0x0090	0x0072 *注意 0x00F2 *注意 3、6	
键盘/小键盘	键盘 LANG2	0x0007	0x0091	0x0071 *注意 0x00F1 *注意 3、6	
键盘/小键盘	键盘 LANG3	0x0007	0x0092	0x0078	
键盘/小键盘	键盘 LANG4	0x0007	0x0093	0x0077	
键盘/小键盘	键盘 LANG5	0x0007	0x0094	0x0076	
键盘/小键盘	键盘 LeftControl	0x0007	0x00E0	0x001D	58
键盘/小键盘	键盘 LeftShift	0x0007	0x00E1	0x002A	44
键盘/小键盘	键盘 LeftAlt	0x0007	0x00E2	0x0038	60
键盘/小键盘	键盘 Left GUI	0x0007	0x00E3	0xE05B	127
键盘/小键盘	键盘 RightControl	0x0007	0x00E4	0xE01D	64
键盘/小键盘	键盘 RightShift	0x0007	0x00E5	0x0036	57
键盘/小键盘	键盘 RightAlt	0x0007	0x00E6	0xE038	62
键盘/小键盘	键盘 Right GUI	0x0007	0x00E7	0xE05C	128
消费者	扫描下一个曲目	0x000C	0x00B5	0xE019	
消费者	扫描上一个曲目	0x000C	0x00B6	0xE010	
消费者	停止	0x000C	0x00B7	0xE024	
消费者	播放/暂停	0x000C	0x00CD	0xE022	
消费者	静音	0x000C	0x00E2	0xE020	
消费者	音量增大	0x000C	0x00E9	0xE030	
消费者	音量减小	0x000C	0x00EA	0xE02E	
消费者	AL 使用者控制配置	0x000C	0x0183	0xE06D	
消费者	AL 电子邮件阅读器	0x000C	0x018A	0xE06C	
消费者	AL 计算器	0x000C	0x0192	0xE021	
消费者	AL 本地计算机浏览器	0x000C	0x0194	0xE06B	

HID 使用页 面	HID 用法名称	HID 使用页 面	HID 用法	Scan 1 Make	键位置
		0x000C	0x0221	0xE065	
消费者	AC 开始	0x000C	0x0223	0xE032	
消费者	AC 返回	0x000C	0x0224	0xE06A	
消费者	AC 向前	0x000C	0x0225	0xE069	
消费者	AC 停止	0x000C	0x0226	0xE068	
消费者	AC 刷新	0x000C	0x0227	0xE067	
消费者	AC 上一个链接	0x000C	0x022A	0xE066	

注释：

1. *Alt+Print screen* 击键时发出 *SysRq* 键扫描代码
2. *Control+Pause* 击键时发出 *Break* 键扫描代码
3. 如[旧版键盘消息](#)所示
4. 键存在于巴西键盘上
5. 键存在于日本键盘上
6. 仅键释放事件中发出扫描代码

扩展键标志

扩展键标志指示击键消息是否源自增强型 101/102 键键盘上的一个附加键。 扩展键包括键盘右侧的 ALT 和 CTRL 键； INS、DEL、HOME、END、PAGE UP、PAGE DOWN 和数字键盘左侧键群中的方向键； NUM LOCK 键； BREAK (CTRL+PAUSE) 键； PRINT SCRN 键；以及数字键盘中的斜杠 (/) 和 ENTER 键。 右侧 SHIFT 键不被视为扩展键，它有一个单独的扫描代码。

如果指定，则扫描代码由两个字节的序列组成，其中第一个字节的值为 0xE0。

上下文代码

上下文代码指示生成击键消息时是否按下 ALT 键。 如果按下 ALT 键则代码为 1，如果不按则为 0。

上一个键状态标志

上一个键状态标志指示生成击键消息的键先前是否被按下。 如果键之前被按下，则为 1； 如果键之前未按下，则为 0。 可以使用此标志来标识键盘的自动重复功能生成的击键

消息。对于自动重复功能生成的 [WM_KEYDOWN](#) 和 [WM_SYSKEYDOWN](#) 击键消息，此标志设置为 1。对于 [WM_KEYUP](#) 和 [WM_SYSKEYUP](#) 消息，它始终设置为 1。

转换状态标志

转换状态标志指示是按下键还是松开键生成击键消息。对于 [WM_KEYDOWN](#) 和 [WM_SYSKEYDOWN](#) 消息，此标志始终设置为 0；对于 [WM_KEYUP](#) 和 [WM_SYSKEYUP](#) 消息，它始终设置为 1。

字符消息

击键消息提供有关击键的大量信息，但它们不提供字符击键的字符代码。若要检索字符代码，应用程序必须在其线程消息循环中包含 [TranslateMessage](#) 函数。

[TranslateMessage](#) 将 [WM_KEYDOWN](#) 或 [WM_SYSKEYDOWN](#) 消息传递到键盘布局。布局检查消息的虚拟键代码，如果它对应于字符键，则提供等效的字符代码（考虑到 SHIFT 和 CAPS LOCK 键的状态）。然后，它会生成包含字符代码的字符消息，并将消息置于消息队列的顶部。消息循环的下一次迭代会从队列中删除字符消息，并将消息调度到相应的窗口过程。

本部分涵盖了以下主题：

- 非系统字符消息
- 死字符消息

非系统字符消息

窗口过程可以接收以下字符消息：[WM_CHAR](#)、[WM_DEADCHAR](#)、[WM_SYSCHAR](#)、[WM_SYSDEADCHAR](#) 和 [WM_UNICHAR](#)。[TranslateMessage](#) 函数在处理 [WM_KEYDOWN](#) 消息时会生成 [WM_CHAR](#) 或 [WM_DEADCHAR](#) 消息。同样，它在处理 [WM_SYSKEYDOWN](#) 消息时会生成 [WM_SYSCHAR](#) 或 [WM_SYSDEADCHAR](#) 消息。

处理键盘输入的应用程序通常会忽略除 [WM_CHAR](#) 和 [WM_UNICHAR](#) 消息之外的所有消息，将任何其他消息传递给 [DefWindowProc](#) 函数。请注意，[WM_CHAR](#) 使用 UTF-16（16 位 Unicode 转换格式）或 ANSI 字符集，而 [WM_UNICHAR](#) 始终使用 UTF-32（32 位 Unicode 转换格式）。系统使用 [WM_SYSCHAR](#) 和 [WM_SYSDEADCHAR](#) 消息来实现菜单助记键。

所有字符消息的 wParam 参数包含按下的字符键的字符代码。字符代码的值取决于接收消息的窗口的窗口类。如果使用 [RegisterClass](#) 函数的 Unicode 版本注册窗口类，系统会为该类的所有窗口提供 Unicode 字符。否则，系统会提供 ANSI 字符代码。有关详细信息，请参阅[注册窗口类](#)和[在 Windows 应用中使用 UTF-8 代码页](#)。

字符消息的 IParam 参数的内容与被转换为生成字符消息的按键消息的 IParam 参数的内容相同。有关信息，请参阅[击键消息标志](#)。

死字符消息

某些非英语键盘包含不应自行生成字符的字符键。相反，它们用于向后续击键生成的字符添加变音符。这些键称为死键。德语键盘上的长音符键是死键的一个示例。若要输入包含带长音符的“o”的字符，德国用户会键入长音符键，然后键入“o”键。具有键盘焦点的窗口将收到以下消息序列：

1. [WM_KEYDOWN](#)
2. [WM_DEADCHAR](#)
3. [WM_KEYUP](#)
4. [WM_KEYDOWN](#)
5. [WM_CHAR](#)
6. [WM_KEYUP](#)

[TranslateMessage](#) 在处理来自死键的 [WM_KEYDOWN](#) 消息时生成 [WM_DEADCHAR](#) 消息。尽管 [WM_DEADCHAR](#) 消息的 wParam 参数包含死键变音符的字符代码，但应用程序通常会忽略该消息。相反，它会处理后续击键生成的 [WM_CHAR](#) 消息。[WM_CHAR](#) 消息的 wParam 参数包含带有变音符的字母的字符代码。如果后续击键生成的字符不能与变音符组合，系统会生成两条 [WM_CHAR](#) 消息。第一个 wParam 参数包含变音符的字符代码；第二个 wParam 参数包含后续字符键的字符代码。

[TranslateMessage](#) 函数在处理来自系统死键（与 ALT 键一起按下的死键）的 [WM_SYSKEYDOWN](#) 消息时生成 [WM_SYSDEADCHAR](#) 消息。应用程序通常会忽略 [WM_SYSDEADCHAR](#) 消息。

密钥状态

在处理键盘消息时，应用程序可能需要确定除生成当前消息的那个键之外的另一个键的状态。例如，允许用户按 SHIFT+END 选择文本块的文字处理应用程序必须在收到来自 END 键的击键消息时检查 SHIFT 键的状态。应用程序可以使用 [GetKeyState](#) 函数来确定生成当前消息时虚拟键的状态；它可以使用 [GetAsyncKeyState](#) 函数来检索虚拟键的当前状态。

键盘布局保留名称列表。生成单个字符的键的名称与键生成的字符的名称相同。非字符键（如 TAB 和 ENTER）的名称存储为字符串。应用程序可以通过调用 [GetKeyNameText](#) 函数从设备驱动程序检索任何键的名称。

击键和字符转换

该系统包括几个特殊用途函数，用于转换各种击键消息提供的扫描代码、字符代码和虚拟键代码。这些函数包括 [MapVirtualKey](#)、[ToAscii](#)、[ToUnicode](#) 和 [VkKeyScan](#)。

此外，Microsoft Rich Edit 3.0 还支持 [HexToUnicode IME](#)，它允许用户使用热键在十六进制字符和 Unicode 字符之间进行转换。这意味着，当 Microsoft Rich Edit 3.0 合并到应用程序中时，应用程序将继承 HexToUnicode IME 的功能。

热键支持

热键是一种组合键，用于生成 [WM_HOTKEY](#) 消息，即系统放置在线程消息队列顶部的消息，绕过队列中的任何现有消息。应用程序使用热键从用户处获取高优先级键盘输入。例如，通过定义由 CTRL+C 组合键组成的热键，应用程序可以允许用户取消冗长的操作。

为了定义热键，应用程序调用 [RegisterHotKey](#) 函数，指定生成 [WM_HOTKEY](#) 消息的键的组合、用于接收消息的窗口句柄以及热键的标识符。当用户按下热键时，[WM_HOTKEY](#) 消息将放入创建窗口的线程的消息队列中。消息的 wParam 参数包含热键的标识符。应用程序可以为一个线程定义多个热键，但线程中的每个热键必须具有唯一标识符。在应用程序终止之前，它应使用 [UnregisterHotKey](#) 函数销毁热键。

应用程序可以使用热键控件，便于用户轻松选择热键。热键控件通常用于定义激活窗口的热键；它们不使用 [RegisterHotKey](#) 和 [UnregisterHotKey](#) 函数。相反，使用热键控件的应用程序通常会发送 [WM_SETHOTKEY](#) 消息来设置热键。每当用户按下热键时，系统都会发送一条指定 SC_HOTKEY 的 [WM_SYSCOMMAND](#) 消息。有关热键控件的详细信息，请参阅[热键控件](#)中的“使用热键控件”。

用于浏览和其他功能的键盘键

Windows 支持带有特殊键的键盘，用于浏览器功能、媒体功能、应用程序启动和电源管理。[WM_APPCOMMAND](#) 支持额外的键盘键。此外，还修改了 [ShellProc](#) 函数以支持额外的键盘键。

组件应用程序中的子窗口不太可能能够直接实现这些额外键盘键的命令。因此，按下其中一个键时，[DefWindowProc](#) 会向窗口发送 [WM_APPCOMMAND](#) 消息。

[DefWindowProc](#) 还会将 [WM_APPCOMMAND](#) 消息发送到其父窗口（以气泡形式）。这类似于使用鼠标右键调用上下文菜单的方式，即 [DefWindowProc](#) 在右键单击时发送 [WM_CONTEXTMENU](#) 消息，并将其发送到其父项（以气泡形式）。此外，如果 [DefWindowProc](#) 收到顶级窗口的 [WM_APPCOMMAND](#) 消息，它将调用代码为 HSHELL_APPCOMMAND 的 shell 挂钩。

Windows 还支持 Microsoft IntelliMouse Explorer，它是一个有五个按钮的鼠标。两个额外的按钮支持向前和向后浏览器导航。有关详细信息，请参阅[XBUTTON](#)。

模拟输入

若要模拟一系列不间断的用户输入事件，请使用 [SendInput](#) 函数。函数接受三个参数。第一个参数 `cInputs` 指示将模拟的输入事件数。第二个参数 `rgInputs` 是一个 [INPUT](#) 结构数组，每个结构都描述一种输入事件类型和有关该事件的附加信息。最后一个参数 `cbSize` 接受 [INPUT](#) 结构的大小（以字节为单位）。

[SendInput](#) 函数的工作原理是将一系列模拟输入事件注入设备的输入流。效果类似于重复调用 [keybd_event](#) 或 [mouse_event](#) 函数，只是系统会确保没有其他输入事件与模拟事件混为一体。调用完成后，返回值指示成功播放的输入事件数。如果此值为零，则表示输入被阻止。

[SendInput](#) 函数不会重置键盘的当前状态。因此，如果用户在调用此函数时按下了任何键，它们可能会干扰此函数生成的事件。如果担心可能存在干扰，请使用 [GetAsyncKeyState](#) 函数检查键盘的状态，并根据需要进行更正。

语言、区域设置和键盘布局

语言是一种自然语言，例如英语、法语和日语。子语言是在特定地理区域使用的自然语言的变体，例如英式英语和美式英语。应用程序使用称为[语言标识符](#)的值来唯一标识语言和子语言。

应用程序通常使用区域设置来设置处理输入和输出所使用的语言。例如，设置键盘的区域设置会影响键盘生成的字符值。设置显示器或打印机的区域设置会影响显示或打印的字形。应用程序通过加载和使用键盘布局来设置键盘的区域设置。它们通过选择支持指定区域设置的字体来设置显示器或打印机的区域设置。

键盘布局不仅指定键在键盘上的物理位置，还确定通过按下这些键会生成的字符值。每个布局标识当前输入语言，并确定由哪些键和键组合生成哪些字符值。

每个键盘布局都有一个相应的句柄，用于标识布局和语言。句柄的低位字是语言标识符。高位字是设备句柄，用于指定物理布局；或者为零，表示默认物理布局。用户可以将任何输入语言与物理布局相关联。例如，偶尔使用法语的英语用户可以将键盘的输入语言设置为法语，而无需更改键盘的物理布局。这意味着用户可以使用熟悉的英语布局输入法语文本。

应用程序通常不会直接操作输入语言。相反，用户设置语言和布局组合，然后在它们之间切换。当用户单击使用其他语言标记的文本时，应用程序会调用 [ActivateKeyboardLayout](#) 函数来激活用户对该语言的默认布局。如果用户使用不在活动列表中的语言编辑文本，应用程序可以使用该语言调用 [LoadKeyboardLayout](#) 函数以获得基于该语言的布局。

[ActivateKeyboardLayout](#) 函数设置当前任务的输入语言。 hkl 参数可以是键盘布局的句柄或零扩展语言标识符。可以通过 [LoadKeyboardLayout](#) 或 [GetKeyboardLayoutList](#) 函数获取键盘布局句柄。也可使用 HKL_NEXT 和 HKL_PREV 值选择下一个或上一个键盘。

[GetKeyboardLayoutName](#) 函数检索调用线程的活动键盘布局的名称。如果应用程序使用 [LoadKeyboardLayout](#) 函数创建活动布局, [GetKeyboardLayoutName](#) 会检索用于创建布局的相同字符串。否则, 该字符串是对应于活动布局的区域设置的主要语言标识符。这意味着函数不一定能区分具有相同主要语言的不同布局, 因此无法返回有关输入语言的特定信息。但是, [GetKeyboardLayout](#) 函数可用于确定输入语言。

[LoadKeyboardLayout](#) 函数加载键盘布局并使该布局可供用户使用。应用程序可以使用 KLF_ACTIVATE 值使当前线程的布局立即处于活动状态。应用程序可以使用 KLF_REORDER 值对布局重新排序, 而无需同时指定 KLF_ACTIVATE 值。加载键盘布局时, 应用程序应始终使用 KLF_SUBSTITUTE_OK 值, 以确保选择用户的首选项 (如果有)。

对于多语言支持, [LoadKeyboardLayout](#) 函数提供了 KLF_REPLACELANG 和 KLF_NOTELLSHELL 标志。KLF_REPLACELANG 标志指示函数在不更改语言的情况下替换现有的键盘布局。尝试使用相同的语言标识符替换现有布局但不指定 KLF_REPLACELANG 是错误的。KLF_NOTELLSHELL 标志可阻止函数在添加或替换键盘布局时通知 shell。这对于在一系列连续调用中添加多个布局的应用程序很有用。除最后一次调用外, 所有调用都应使用此标志。

[UnloadKeyboardLayout](#) 函数受限, 无法卸载系统默认输入语言。这可确保用户始终有一种布局可用于输入文本, 所用字符集与 shell 和文件系统使用的字符集相同。

反馈

此页面是否有帮助?

是

否

使用键盘输入

项目 · 2024/01/28

窗口以击键消息和字符消息的形式接收键盘输入。附加到窗口的消息循环必须包含代码，才能将击键消息转换为相应的字符消息。如果窗口在其工作区中显示键盘输入，它应创建并显示插入点以指示将输入下一个字符的位置。以下部分介绍接收、处理和显示键盘输入所涉及的代码：

- [处理击键消息](#)
- [翻译字符消息](#)
- [处理字符消息](#)
- [使用插入符号](#)
- [显示键盘输入](#)

处理击键消息

具有键盘焦点的窗口的窗口过程在用户键入键盘时接收击键消息。击键消息是 [WM_KEYDOWN](#)、[WM_KEYUP](#)、[WM_SYSKEYDOWN](#) 和 [WM_SYSKEYUP](#)。典型的窗口过程会忽略 除 [WM_KEYDOWN](#) 之外的所有击键消息。当用户按下键时，系统会发布 [WM_KEYDOWN](#) 消息。

当窗口过程收到 [WM_KEYDOWN](#) 消息时，它应检查消息附带的虚拟键代码，以确定如何处理击键。虚拟密钥代码位于消息的 *wParam* 参数中。通常，应用程序只处理非字符键生成的击键，包括函数键、游标移动键以及 INS、DEL、HOME 和 END 等特殊用途键。

以下示例显示了典型应用程序用来接收和处理击键消息的窗口过程框架。

```
case WM_KEYDOWN:
    switch (wParam)
    {
        case VK_LEFT:
            // Process the LEFT ARROW key.

            break;

        case VK_RIGHT:
            // Process the RIGHT ARROW key.

            break;

        case VK_UP:
```

```
// Process the UP ARROW key.  
  
    break;  
  
case VK_DOWN:  
  
    // Process the DOWN ARROW key.  
  
    break;  
  
case VK_HOME:  
  
    // Process the HOME key.  
  
    break;  
  
case VK_END:  
  
    // Process the END key.  
  
    break;  
  
case VK_INSERT:  
  
    // Process the INS key.  
  
    break;  
  
case VK_DELETE:  
  
    // Process the DEL key.  
  
    break;  
  
case VK_F2:  
  
    // Process the F2 key.  
  
    break;  
  
// Process other non-character keystrokes.  
  
default:  
    break;  
}
```

翻译字符消息

从用户接收字符输入的任何线程都必须在其消息循环中包含 [TranslateMessage](#) 函数。此函数检查击键消息的虚拟键代码，如果代码对应于字符，将字符消息置于消息队列中。

在消息循环的下一次迭代中删除并调度字符消息;消息的 *wParam* 参数包含字符代码。

通常, 线程的消息循环应使用 [TranslateMessage](#) 函数来翻译每条消息, 而不仅仅是虚拟键消息。 尽管 [TranslateMessage](#) 对其他类型的消息没有影响, 但它保证键盘输入正确转换。 以下示例演示如何在典型的线程消息循环中包含 [TranslateMessage](#) 函数。

```
MSG msg;
BOOL bRet;

while (( bRet = GetMessage(&msg, (HWND) NULL, 0, 0)) != 0)
{
    if (bRet == -1);
    {
        // handle the error and possibly exit
    }
    else
    {
        if (TranslateAccelerator(hwndMain, haccl, &msg) == 0)
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

处理字符消息

当 [TranslateMessage](#) 函数转换对应于字符键的虚拟键代码时, 窗口过程会收到字符消息。 字符消息是 [WM_CHAR](#)、[WM_DEADCHAR](#)、[WM_SYSCHAR](#)和[WM_SYSDEADCHAR](#)。 典型的窗口过程会忽略 除[WM_CHAR](#)以外的所有字符消息。 当用户按下以下任一键时, [TranslateMessage](#) 函数将生成 [WM_CHAR](#) 消息:

- 任何字符键
- Backspace
- ENTER (回车)
- ESC
- SHIFT+Enter (换行符)
- Tab

当窗口过程收到 [WM_CHAR](#) 消息时, 它应检查消息附带的字符代码以确定如何处理字符。 字符代码位于消息的 *wParam* 参数中。

以下示例显示了典型应用程序用来接收和处理字符消息的窗口过程框架。

```
case WM_CHAR:
    switch (wParam)
    {
        case 0x08:
            // Process a backspace.

            break;

        case 0x0A:
            // Process a linefeed.

            break;

        case 0x1B:
            // Process an escape.

            break;

        case 0x09:
            // Process a tab.

            break;

        case 0x0D:
            // Process a carriage return.

            break;

        default:
            // Process displayable characters.

            break;
    }
```

使用插入符号

接收键盘输入的窗口通常显示用户在窗口的工作区中键入的字符。 窗口应使用插入符号来指示下一个字符将出现在工作区中的位置。 窗口还应在收到键盘焦点时创建和显示插入符号，并在失去焦点时隐藏和销毁插入符号。 窗口可以在[处理WM_SETFOCUS](#) 和[WM_KILLFOCUS](#) 消息时执行这些操作。 有关插入符号的详细信息，请参阅[Caret](#)。

显示键盘输入

本部分中的示例显示了应用程序如何从键盘接收字符、在窗口的工作区中显示字符，以及更新键入每个字符的插入点的位置。它还演示了如何移动插入符号以响应向左键、向右键、开始键和结束击键，并演示如何突出显示所选文本以响应 Shift+ 向右键组合。

在处理 [WM_CREATE](#) 消息期间，示例中所示的窗口过程分配用于存储键盘输入的 64K 缓冲区。它还检索当前加载的字体的指标，保存字体中字符的高度和平均宽度。高度和宽度用于处理 [WM_SIZE](#) 消息，根据工作区的大小计算行长度和最大行数。

窗口过程在处理 [WM_SETFOCUS](#) 消息时创建并显示插入符号。处理 [WM_KILLFOCUS](#) 消息时，它会隐藏和删除插入点。

处理 [WM_CHAR](#) 消息时，窗口过程会显示字符，将它们存储在输入缓冲区中，并更新插入点位置。窗口过程还会将制表符转换为四个连续空格字符。后空、换行符和转义字符会生成蜂鸣声，但不进行其他处理。

当处理 [WM_KEYDOWN](#) 消息时，窗口过程执行左、右、端和家庭插入点移动。处理向右键的操作时，窗口过程会检查 SHIFT 键的状态，如果关闭，请在插入点移动时选择插入点右侧的字符。

请注意，编写以下代码，以便可以编译为 Unicode 或 ANSI。如果源代码定义 UNICODE，字符串将作为 Unicode 字符进行处理;否则，它们将作为 ANSI 字符进行处理。

```
#define BUFSIZE 65535
#define SHIFTED 0x8000

LONG APIENTRY MainWndProc(HWND hwndMain, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;                      // handle to device context
    TEXTMETRIC tm;                // structure for text metrics
    static DWORD dwCharX;          // average width of characters
    static DWORD dwCharY;          // height of characters
    static DWORD dwClientX;        // width of client area
    static DWORD dwClientY;        // height of client area
    static DWORD dwLineLen;         // line length
    static DWORD dwLines;           // text lines in client area
    static int nCaretPosX = 0;      // horizontal position of caret
    static int nCaretPosY = 0;      // vertical position of caret
    static int nCharWidth = 0;       // width of a character
    static int cch = 0;              // characters in buffer
    static int nCurChar = 0;         // index of current character
    static PTCHAR pchInputBuf;      // input buffer
    int i, j;                      // loop counters
    int cCR = 0;                   // count of carriage returns
    int nCRIIndex = 0;              // index of last carriage return
    int nVirtKey;                  // virtual-key code
```

```
TCHAR szBuf[128];           // temporary buffer
TCHAR ch;                  // current character
PAINTSTRUCT ps;            // required by BeginPaint
RECT rc;                   // output rectangle for DrawText
SIZE sz;                   // string dimensions
COLORREF crPrevText;       // previous text color
COLORREF crPrevBk;         // previous background color
size_t * pcch;
HRESULT hResult;

switch (uMsg)
{
    case WM_CREATE:

        // Get the metrics of the current font.

        hdc = GetDC(hwndMain);
        GetTextMetrics(hdc, &tm);
        ReleaseDC(hwndMain, hdc);

        // Save the average character width and height.

        dwCharX = tm.tmAveCharWidth;
        dwCharY = tm.tmHeight;

        // Allocate a buffer to store keyboard input.

        pchInputBuf = (LPTSTR) GlobalAlloc(GPTR,
            BUFSIZE * sizeof(TCHAR));
        return 0;

    case WM_SIZE:

        // Save the new width and height of the client area.

        dwClientX = LOWORD(lParam);
        dwClientY = HIWORD(lParam);

        // Calculate the maximum width of a line and the
        // maximum number of lines in the client area.

        dwLineLen = dwClientX - dwCharX;
        dwLines = dwClientY / dwCharY;
        break;

    case WM_SETFOCUS:

        // Create, position, and display the caret when the
        // window receives the keyboard focus.

        CreateCaret(hwndMain, (HBITMAP) 1, 0, dwCharY);
        SetCaretPos(nCaretPosX, nCaretPosY * dwCharY);
        ShowCaret(hwndMain);
        break;
}
```

```
case WM_KILLFOCUS:

    // Hide and destroy the caret when the window loses the
    // keyboard focus.

    HideCaret(hwndMain);
    DestroyCaret();
    break;

case WM_CHAR:
    // check if current location is close enough to the
    // end of the buffer that a buffer overflow may
    // occur. If so, add null and display contents.

if (cch > BUFSIZE-5)
{
    pchInputBuf[cch] = 0x00;
    SendMessage(hwndMain, WM_PAINT, 0, 0);
}

switch (wParam)
{
    case 0x08: // backspace
    case 0x0A: // linefeed
    case 0x1B: // escape
        MessageBeep((UINT) -1);
        return 0;

    case 0x09: // tab

        // Convert tabs to four consecutive spaces.

        for (i = 0; i < 4; i++)
            SendMessage(hwndMain, WM_CHAR, 0x20, 0);
        return 0;

    case 0x0D: // carriage return

        // Record the carriage return and position the
        // caret at the beginning of the new line.

        pchInputBuf[cch++] = 0x0D;
        nCaretPosX = 0;
        nCaretPosY += 1;
        break;

    default: // displayable character

        ch = (TCHAR) wParam;
        HideCaret(hwndMain);

        // Retrieve the character's width and output
        // the character.

        hdc = GetDC(hwndMain);
        GetCharWidth32(hdc, (UINT) wParam, (UINT) wParam,
```

```

        &nCharWidth);
    TextOut(hdc, nCaretPosX, nCaretPosY * dwCharY,
            &ch, 1);
    ReleaseDC(hwndMain, hdc);

    // Store the character in the buffer.

    pchInputBuf[cch++] = ch;

    // Calculate the new horizontal position of the
    // caret. If the position exceeds the maximum,
    // insert a carriage return and move the caret
    // to the beginning of the next line.

    nCaretPosX += nCharWidth;
    if ((DWORD) nCaretPosX > dwLineLen)
    {
        nCaretPosX = 0;
        pchInputBuf[cch++] = 0x0D;
        ++nCaretPosY;
    }
    nCurChar = cch;
    ShowCaret(hwndMain);
    break;
}
SetCaretPos(nCaretPosX, nCaretPosY * dwCharY);
break;

case WM_KEYDOWN:
switch (wParam)
{
    case VK_LEFT: // LEFT ARROW

        // The caret can move only to the beginning of
        // the current line.

        if (nCaretPosX > 0)
        {
            HideCaret(hwndMain);

            // Retrieve the character to the left of
            // the caret, calculate the character's
            // width, then subtract the width from the
            // current horizontal position of the caret
            // to obtain the new position.

            ch = pchInputBuf[--nCurChar];
            hdc = GetDC(hwndMain);
            GetCharWidth32(hdc, ch, ch, &nCharWidth);
            ReleaseDC(hwndMain, hdc);
            nCaretPosX = max(nCaretPosX - nCharWidth,
                             0);
            ShowCaret(hwndMain);
        }
        break;
}

```

```
case VK_RIGHT: // RIGHT ARROW

    // Caret moves to the right or, when a carriage
    // return is encountered, to the beginning of
    // the next line.

    if (nCurChar < cch)
    {
        HideCaret(hwndMain);

        // Retrieve the character to the right of
        // the caret. If it's a carriage return,
        // position the caret at the beginning of
        // the next line.

        ch = pchInputBuf[nCurChar];
        if (ch == 0x0D)
        {
            nCaretPosX = 0;
            nCaretPosY++;
        }

        // If the character isn't a carriage
        // return, check to see whether the SHIFT
        // key is down. If it is, invert the text
        // colors and output the character.

        else
        {
            hdc = GetDC(hwndMain);
            nVirtKey = GetKeyState(VK_SHIFT);
            if (nVirtKey & SHIFTED)
            {
                crPrevText = SetTextColor(hdc,
                    RGB(255, 255, 255));
                crPrevBk = SetBkColor(hdc,
                    RGB(0,0,0));
                TextOut(hdc, nCaretPosX,
                    nCaretPosY * dwCharY,
                    &ch, 1);
                SetTextColor(hdc, crPrevText);
                SetBkColor(hdc, crPrevBk);
            }

            // Get the width of the character and
            // calculate the new horizontal
            // position of the caret.

            GetCharWidth32(hdc, ch, ch, &nCharWidth);
            ReleaseDC(hwndMain, hdc);
            nCaretPosX = nCaretPosX + nCharWidth;
        }

        nCurChar++;
        ShowCaret(hwndMain);
```

```
        break;
    }
    break;

case VK_UP:      // UP ARROW
case VK_DOWN:    // DOWN ARROW
    MessageBeep((UINT) -1);
    return 0;

case VK_HOME:    // HOME

    // Set the caret's position to the upper left
    // corner of the client area.

    nCaretPosX = nCaretPosY = 0;
    nCurChar = 0;
    break;

case VK_END:     // END

    // Move the caret to the end of the text.

    for (i=0; i < cch; i++)
    {
        // Count the carriage returns and save the
        // index of the last one.

        if (pchInputBuf[i] == 0x0D)
        {
            cCR++;
            nCRIndex = i + 1;
        }
    }
    nCaretPosY = cCR;

    // Copy all text between the last carriage
    // return and the end of the keyboard input
    // buffer to a temporary buffer.

    for (i = nCRIndex, j = 0; i < cch; i++, j++)
        szBuf[j] = pchInputBuf[i];
    szBuf[j] = TEXT('\0');

    // Retrieve the text extent and use it
    // to set the horizontal position of the
    // caret.

    hdc = GetDC(hwndMain);
    hResult = StringCchLength(szBuf, 128, pcch);
    if (FAILED(hResult))
    {
        // TODO: write error handler
    }
    GetTextExtentPoint32(hdc, szBuf, *pcch,
        &sz);
```

```
    nCaretPosX = sz.cx;
    ReleaseDC(hwndMain, hdc);
    nCurChar = cch;
    break;

    default:
        break;
}
SetCaretPos(nCaretPosX, nCaretPosY * dwCharY);
break;

case WM_PAINT:
if (cch == 0)          // nothing in input buffer
    break;

hdc = BeginPaint(hwndMain, &ps);
HideCaret(hwndMain);

// Set the clipping rectangle, and then draw the text
// into it.

SetRect(&rc, 0, 0, dwLineLen, dwClientY);
DrawText(hdc, pchInputBuf, -1, &rc, DT_LEFT);

ShowCaret(hwndMain);
EndPaint(hwndMain, &ps);
break;

// Process other messages.

case WM_DESTROY:
PostQuitMessage(0);

// Free the input buffer.

GlobalFree((HGLOBAL) pchInputBuf);
UnregisterHotKey(hwndMain, 0xAAAA);
break;

default:
    return DefWindowProc(hwndMain, uMsg, wParam, lParam);
}
return NULL;
}
```

反馈

此页面是否有帮助?

是

否

键盘输入参考

项目 · 2024/02/24

本节内容

- 键盘输入函数
- 键盘输入消息
- 键盘输入通知
- 键盘输入结构
- 键盘输入常量

反馈

此页面是否有帮助?

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

键盘输入函数

项目 · 2023/06/13

本节内容

- [ActivateKeyboardLayout](#)
- [BlockInput](#)
- [EnableWindow](#)
- [GetActiveWindow](#)
- [GetAsyncKeyState](#)
- [GetFocus](#)
- [GetKBCodePage](#)
- [GetKeyboardLayout](#)
- [GetKeyboardLayoutList](#)
- [GetKeyboardLayoutName](#)
- [GetKeyboardState](#)
- [GetKeyboardType](#)
- [GetKeyNameText](#)
- [GetKeyState](#)
- [GetLastInputInfo](#)
- [IsWindowEnabled](#)
- [keybd_event](#)
- [LoadKeyboardLayout](#)
- [MapVirtualKey](#)
- [MapVirtualKeyEx](#)
- [OemKeyScan](#)
- [RegisterHotKey](#)
- [SendInput](#)
- [SetActiveWindow](#)
- [SetFocus](#)
- [SetKeyboardState](#)
- [ToAscii](#)
- [ToAsciiEx](#)
- [ToUnicode](#)
- [ToUnicodeEx](#)
- [UnloadKeyboardLayout](#)
- [UnregisterHotKey](#)
- [VkKeyScan](#)
- [VkKeyScanEx](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

activateKeyboardLayout 函数 (winuser.h)

项目2023/08/27

设置调用线程或当前进程的输入区域设置标识符（以前称为键盘布局句柄）。输入区域设置标识符指定区域设置以及键盘的物理布局。

语法

C++

```
HKL ActivateKeyboardLayout(
    [in] HKL hkl,
    [in] UINT Flags
);
```

参数

[in] hkl

类型: HKL

要激活的输入区域设置标识符。

输入区域设置标识符必须已通过先前对 [LoadKeyboardLayout](#) 函数的调用加载。此参数必须是键盘布局的句柄或以下值之一。

值	含义
HKL_NEXT 1	选择系统维护的已加载区域设置标识符循环列表中的下一个区域设置标识符。
HKL_PREV 0	选择系统维护的已加载区域设置标识符循环列表中的上一个区域设置标识符。

[in] Flags

类型: UINT

指定如何激活输入区域设置标识符。此参数的取值可为下列值之一：

值	含义
---	----

KLF_REORDER 0x00000008	<p>如果设置了此位，则会通过将区域设置标识符移动到列表的头来重新排列系统已加载的区域设置标识符的循环列表。如果未设置此位，则无需更改顺序即可旋转列表。</p> <p>例如，如果用户有一个处于活动状态的英语区域设置标识符，并且将法语、德语和西班牙语区域设置标识符加载()顺序，则激活具有 KLF_REORDER 位集的德语区域设置标识符将生成以下顺序：德语、英语、法语、西班牙语。在未设置 KLF_REORDER 位的情况下激活德语区域设置标识符将生成以下顺序：德语、西班牙语、英语、法语。</p> <p>如果加载的区域设置标识符少于三个，则此标志的值无关紧要。</p>
KLF_RESET 0x40000000	<p>如果已设置 但未设置 KLF_SHIFTLOCK，则再次按 Caps Lock 键关闭 Caps Lock 状态。如果同时 设置了 KLF_SHIFTLOCK，则通过按任一 SHIFT 键关闭 Caps Lock 状态。</p> <p>这两种方法是互斥的，设置在注册表中作为用户配置文件的一部分保留。</p>
KLF_SETFORPROCESS 0x00000100	激活整个进程的指定区域设置标识符，并将 WM_INPUTLANGCHANGE 消息发送到当前线程的焦点或活动窗口。
KLF_SHIFTLOCK 0x00010000	这与 KLF_RESET 一起使用。有关说明，请参阅 KLF_RESET 。
KLF_UNLOADPREVIOUS	不支持此标志。请改用 UnloadKeyboardLayout 函数。

返回值

类型： **HKL**

返回值的类型为 **HKL**。如果函数成功，则返回值为上一个输入区域设置标识符。否则为零。

若要获取扩展的错误信息，请使用 [GetLastError](#) 函数。

注解

此函数仅影响当前进程或线程的布局。

此函数不限于键盘布局。*hkl* 参数实际上是输入区域设置标识符。这是一个比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(输入法编辑器)或

任何其他形式的输入。一次可以加载多个输入区域设置标识符，但一次只有一个处于活动状态。加载多个输入区域设置标识符可以在它们之间快速切换。

如果每个区域设置允许多个 IME，则传递输入区域设置标识符，其中高字（设备句柄）为零，将激活属于该区域设置的列表中的第一个 IME。

KLF_RESET 和 **KLF_SHIFTLOCK** 标志会更改关闭 Caps Lock 状态的方法。默认情况下，通过再次点击 Caps Lock 键关闭 Caps Lock 状态。如果仅设置 **KLF_RESET**，则重新建立默认状态。如果设置了 **KLF_RESET** 和 **KLF_SHIFTLOCK**，则通过按任一 Caps Lock 键关闭 Caps Lock 状态。此功能用于符合本地键盘行为标准以及个人首选项。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetKeyboardLayoutName](#)

[键盘输入](#)

[LoadKeyboardLayout](#)

引用

[UnloadKeyboardLayout](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

blockInput 函数 (winuser.h)

项目2023/08/27

阻止键盘和鼠标输入事件到达应用程序。

语法

C++

```
BOOL BlockInput(  
    [in] BOOL fBlockIt  
) ;
```

参数

[in] fBlockIt

类型: BOOL

函数的用途。如果此参数为 TRUE，则阻止键盘和鼠标输入事件。如果此参数为 FALSE，则取消阻止键盘和鼠标事件。请注意，只有阻止输入的线程才能成功取消阻止输入。

返回值

类型: BOOL

如果该函数成功，则返回值为非零值。

如果已阻止输入，则返回值为零。要获得更多的错误信息，请调用 GetLastError。

注解

阻止输入时，来自鼠标或键盘的实际物理输入不会影响 GetKeyState 和 GetKeyboardState) 报告的输入队列的同步键状态 (，也不会影响 GetAsyncKeyState) 报告的异步键状态 (。但是，阻止输入的线程可以通过调用 SendInput 影响这两种键状态。其他线程无法执行此操作。

在以下情况下，系统将取消阻止输入：

- 阻止输入的线程意外退出，但未调用 `BlockInput` 且 `fBlock` 设置为 `FALSE`。在这种情况下，系统会正确清理并重新启用输入。
- 用户按 `Ctrl+ALT+DEL` 或系统调用 **硬系统错误** 模式消息框（例如，当程序故障或设备）故障时。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetAsyncKeyState](#)

[GetKeyState](#)

[GetKeyboardState](#)

[键盘输入](#)

引用

[SendInput](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

EnableWindow 函数 (winuser.h)

项目2023/08/27

启用或禁用指定窗口或控件的鼠标和键盘输入。 禁用输入时，窗口不会接收鼠标单击和按键等输入。 启用输入时，窗口会接收所有输入。

语法

C++

```
BOOL EnableWindow(
    [in] HWND hWnd,
    [in] BOOL bEnable
);
```

参数

[in] hWnd

类型: **HWND**

要启用或禁用的窗口的句柄。

[in] bEnable

类型: **BOOL**

指示是启用或禁用窗口。 如果此参数为 **TRUE**，则启用窗口。 如果参数为 **FALSE**，则禁用窗口。

返回值

类型: **BOOL**

如果以前禁用窗口，则返回值为非零值。

如果以前未禁用窗口，则返回值为零。

注解

如果窗口处于禁用状态，系统会发送 [WM_CANCELMODE](#) 消息。如果窗口的启用状态正在更改，系统会在 [WM_CANCELMODE](#) 消息后发送 [WM_ENABLE](#) 消息。(这些消息在 [EnableWindow](#) 返回之前发送。) 如果窗口已禁用，则其子窗口将被隐式禁用，尽管它们不会发送 [WM_ENABLE](#) 消息。

必须先启用窗口才能激活它。例如，如果应用程序显示无模式对话框并禁用其main窗口，则应用程序必须在销毁对话框之前启用main窗口。否则，另一个窗口将接收键盘焦点并被激活。如果禁用了子窗口，当系统尝试确定哪个窗口应接收鼠标消息时，将忽略该窗口。

默认情况下，创建的窗口最初处于已启用状态。若要创建最初禁用的窗口，应用程序可以在 [CreateWindow](#) 或 [CreateWindowEx](#) 函数中指定 [WS_DISABLED](#) 样式。创建窗口后，应用程序可以使用 [EnableWindow](#) 启用或禁用窗口。

应用程序可以使用此函数来启用或禁用对话框中的控件。禁用的控件无法接收键盘焦点，用户也无法获取对它的访问权限。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-window-l1-1-4 ()

请参阅

概念性

[CreateWindow](#)

[CreateWindowEx](#)

[IsWindowEnabled](#)

[键盘输入](#)

引用

[WM_ENABLE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getActiveWindow 函数 (winuser.h)

项目2023/08/27

检索附加到调用线程消息队列中的活动窗口的句柄。

语法

C++

```
HWND GetActiveWindow();
```

返回值

类型: HWND

返回值是附加到调用线程的消息队列的活动窗口的句柄。 否则，返回值为 NULL。

注解

若要获取前台窗口的句柄，可以使用 [GetForegroundWindow](#)。

若要获取另一个线程的消息队列中活动窗口的窗口句柄，请使用 [GetGUIThreadInfo](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-window-l1-1-4 ()

请参阅

概念性

[GetForegroundWindow](#)

[GetGUIThreadInfo](#)

[键盘输入](#)

引用

[SetActiveWindow](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getAsyncKeyState 函数 (winuser.h)

项目2023/08/27

确定调用函数时键是向上还是向下，以及上次调用 GetAsyncKeyState 后是否按下了该键。

语法

C++

```
SHORT GetAsyncKeyState(  
    [in] int vKey  
);
```

参数

[in] vKey

类型： int

虚拟密钥代码。有关详细信息，请参阅 [虚拟密钥代码](#)。

可以使用左右区分常量来指定某些键。有关详细信息，请参阅备注部分。

返回值

类型： SHORT

如果函数成功，则返回值指定自上次调用 GetAsyncKeyState 以来是否按下了键，以及键当前是打开还是关闭。如果设置了最有效位，则键关闭，如果设置了最小有效位，则上一次调用 GetAsyncKeyState 后按下了键。但是，不应依赖于此最后一种行为;有关详细信息，请参阅备注。

对于以下情况，返回值为零：

- 当前桌面不是活动桌面
- 前台线程属于另一个进程，桌面不允许挂钩或日志记录。

注解

`GetAsyncKeyState` 函数适用于鼠标按钮。但是，它会检查物理鼠标按钮的状态，而不是物理按钮映射到的逻辑鼠标按钮。例如，调用 `GetAsyncKeyState(VK_LBUTTON)` 始终返回左物理鼠标按钮的状态，无论它是映射到左逻辑鼠标按钮还是右逻辑鼠标按钮。可以通过调用 `GetSystemMetrics(SM_SWAPBUTTON)` 来确定系统的物理鼠标按钮到逻辑鼠标按钮的当前映射。

如果已交换鼠标按钮，则返回 TRUE。

尽管返回值的最小有效位指示自上次查询以来是否已按下键，但由于 Windows 的抢占性多任务性质，另一个应用程序可以调用 `GetAsyncKeyState` 并接收“最近按下的”位，而不是应用程序。严格保留返回值中最小有效位的行为，以便与 16 位 Windows 应用程序兼容，(这些应用程序是非抢占) 且不应依赖的。

可以使用虚拟键代码常量 `VK_SHIFT`、`VK_CONTROL` 和 `VK_MENU` 作为 `vKey` 参数的值。这会提供 SHIFT、Ctrl 或 ALT 键的状态，而不区分左键和右键。

可以使用以下虚拟键代码常量作为 `vKey` 的值来区分这些键的左实例和右实例。

代码	含义
<code>VK_LSHIFT</code>	左移键。
<code>VK_RSHIFT</code>	右移键。
<code>VK_LCONTROL</code>	左控制键。
<code>VK_RCONTROL</code>	右侧控制键。
<code>VK_LMENU</code>	左侧菜单键。
<code>VK_RMENU</code>	右侧菜单键。

这些左右区分常量仅在调用 `GetKeyboardState`、`SetKeyboardState`、`GetAsyncKeyState`、`GetKeyState` 和 `MapVirtualKey` 函数时可用。

示例

C++

```
while (GetMessage(&msg, nullptr, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

```
switch (msg.message)
{
    case WM_KEYDOWN:
        if ((GetAsyncKeyState(VK_ESCAPE) & 0x01) && bRunning)
        {
            Stop();
        }
        break;
}
```

GitHub 上的 [Windows 经典示例](#) 中的例子。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [GetSystemMetrics](#)
- [MapVirtualKey](#)
- [SetKeyboardState](#)
- [键盘输入](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

GetFocus 函数 (winuser.h)

项目2024/03/04

如果窗口附加到调用线程的消息队列，则检索具有键盘焦点的窗口的句柄。

语法

C++

```
HWND GetFocus();
```

返回值

类型: HWND

返回值是具有键盘焦点的窗口的句柄。 如果调用线程的消息队列没有与键盘焦点关联的窗口，则返回值为 NULL。

注解

GetFocus 返回具有当前线程消息队列的键盘焦点的窗口。 如果 GetFocus 返回 NULL，则另一个线程的队列可能会附加到具有键盘焦点的窗口。

使用 [GetForegroundWindow](#) 函数检索用户当前正在使用的窗口的句柄。 可以使用 [AttachThreadInput](#) 函数将线程的消息队列与其他线程拥有的窗口相关联。

若要获取键盘焦点位于前台队列或其他线程队列上的窗口，请使用 [GetGUIThreadInfo](#) 函数。

示例

有关示例，请参阅使用组合框中的“[创建组合框工具栏](#)”。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	ext-ms-win-ntuser-window-l1-1-4 (在 Windows 10 版本 10.0.14393 中引入)

另请参阅

[AttachThreadInput](#)

概念性

[GetForegroundWindow](#)

[GetGUIThreadInfo](#)

[键盘输入](#)

其他资源

引用

[SetFocus](#)

[WM_KILLFOCUS](#)

[WM_SETFOCUS](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

getKBCodePage 函数 (winuser.h)

项目2023/08/27

检索当前代码页。

注意 提供此函数只是为了与 16 位版本的 Windows 兼容。 应用程序应使用 GetOEMCP 函数检索系统的 OEM 代码页标识符。

语法

C++

```
UINT GetKBCodePage();
```

返回值

类型: **UINT**

返回值为 OEM 代码页标识符;如果注册表值不可读，则返回值为默认标识符。有关 OEM 代码页标识符的列表，请参阅 [代码页标识符](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetACP](#)

[GetOEMCP](#)

[键盘输入](#)

引用

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getKeyboardLayout 函数 (winuser.h)

项目2024/03/04

检索以前称为键盘布局) (活动输入区域设置标识符。

语法

C++

```
HKL GetKeyboardLayout(
    [in] DWORD idThread
);
```

参数

[in] idThread

类型: DWORD

要查询的线程的标识符，对于当前线程，为 0。

返回值

类型: HKL

返回值是线程的输入区域设置标识符。低字包含输入 [语言的语言标识符](#)，高字包含键盘物理布局的设备句柄。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(输入法)或任何其他输入形式。

由于键盘布局可以动态更改，因此缓存有关当前键盘布局的信息的应用程序应处理 [WM_INPUTLANGCHANGE](#) 消息，以通知输入语言的更改。

若要获取当前活动 HKL 的 KLID(键盘布局 ID)，请调用 [GetKeyboardLayoutName](#)。

从 Windows 8 开始：检索与当前键盘布局或输入方法关联的语言的首选方法是调用 [Windows.Globalization.Language.CurrentInputMethodLanguageTag](#)。如果你的应用将

语言标记从 CurrentInputMethodLanguageTag 传递给任何 国家语言支持 函数，它必须首先通过调用 ResolveLocaleName 来转换标记。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

[ActivateKeyboardLayout](#)

概念性

[CreateThread](#)

[键盘输入](#)

[LoadKeyboardLayout](#)

其他资源

引用

[WM_INPUTLANGCHANGE](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

getKeyboardLayoutList 函数 (winuser.h)

项目2023/08/27

检索与系统中的当前输入区域设置集相对应的输入区域设置标识符（以前称为键盘布局句柄）。该函数将标识符复制到指定的缓冲区。

语法

C++

```
int GetKeyboardLayoutList(
    [in] int nBuff,
    [out] HKL *lpList
);
```

参数

[in] nBuff

类型: int

缓冲区可以容纳的最大句柄数。

[out] lpList

类型: HKL*

指向接收输入区域设置标识符数组的缓冲区的指针。

返回值

类型: int

如果函数成功，则返回值是复制到缓冲区的输入区域设置标识符的数目；如果 *nBuff* 为零，则返回值是接收所有当前输入区域设置标识符所需的缓冲区的大小（以数组元素为单位）。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 GetLastError。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(输入法)或任何形式的输入。

从 Windows 8 开始：检索与当前键盘布局或输入法关联的语言的首选方法是调用 [Windows.Globalization.Language.CurrentInputMethodLanguageTag](#)。如果你的应用将语言标记从 `CurrentInputMethodLanguageTag` 传递到任何 [国家/地区语言支持](#) 函数，它必须首先通过调用 [ResolveLocaleName](#) 来转换标记。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetKeyboardLayout](#)

[键盘输入](#)

引用

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getKeyboardLayoutNameA 函数 (winuser.h)

项目2023/07/21

检索活动输入区域设置标识符的名称，(以前称为调用线程的键盘布局)。

语法

C++

```
BOOL GetKeyboardLayoutNameA(
    [out] LPSTR pwszKLID
);
```

参数

[out] pwszKLID

类型: LPTSTR

缓冲区(至少 KL_NAMELENGTH 个字符的长度)，用于接收输入区域设置标识符的名称，包括终止 null 字符。除非进行了布局替换，否则这是提供给 [LoadKeyboardLayout](#) 函数的字符串的副本。

有关 Windows 提供的输入布局的列表，请参阅 [Windows 的键盘标识符和输入法编辑器](#)。

返回值

类型: BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 [GetLastError](#)。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(IME)或任何其他形式的输入。

从 Windows 8 开始: 检索与当前键盘布局或输入法关联的语言的首选方法是调用 `Windows.Globalization.Language.CurrentInputMethodLanguageTag`。如果你的应用将语言标记从 `CurrentInputMethodLanguageTag` 传递给任何 `国家语言支持` 函数，它必须首先通过调用 `ResolveLocaleName` 来转换标记。

① 备注

`winuser.h` 标头将 `GetKeyboardLayoutName` 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	<code>winuser.h</code> (包括 <code>Windows.h</code>)
Library	<code>User32.lib</code>
DLL	<code>User32.dll</code>

另请参阅

[ActivateKeyboardLayout](#)

概念性

[键盘输入](#)

[LoadKeyboardLayout](#)

引用

[UnloadKeyboardLayout](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GetKeyboardState 函数 (winuser.h)

项目2023/08/27

将 256 个虚拟密钥的状态复制到指定的缓冲区。

语法

C++

```
BOOL GetKeyboardState(  
    [out] PBYTE lpKeyState  
) ;
```

参数

[out] lpKeyState

类型: **PBYTE**

接收每个虚拟密钥的状态数据的 256 字节数组。

返回值

类型: **BOOL**

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 [GetLastError](#)。

注解

应用程序可以调用此函数来检索所有虚拟密钥的当前状态。当线程从其消息队列中删除键盘消息时，状态会更改。当键盘消息发布到线程的消息队列时，状态不会更改，也不会在将键盘消息发布到其他线程的消息队列或从其他线程的消息队列中检索时更改。（异常：通过 [AttachThreadInput](#) 连接的线程共享相同的键盘状态。）

函数返回时，*lpKeyState* 参数指向的数组的每个成员都包含虚拟密钥的状态数据。如果高阶位为 1，则键关闭；否则，它已启动。如果键是切换键（例如 CAPS LOCK），则切换键时低序位为 1；如果取消切换键，则为 0。对于非切换键，低序位毫无意义。切换键据

说在打开时处于切换状态。 切换键的指示灯 (键盘上的任何) 在切换键时处于打开状态，并在取消切换键时关闭。

若要检索单个密钥的状态信息，请使用 [GetKeyState](#) 函数。 若要检索单个键的当前状态，而不考虑是否已从消息队列检索相应的键盘消息，请使用 [GetAsyncKeyState](#) 函数。

应用程序可以使用虚拟键代码常量 `VK_SHIFT`、`VK_CONTROL` 和 `VK_MENU` 作为 *lpKeyState* 指向的数组中的索引。 这会提供 SHIFT、Ctrl 或 ALT 键的状态，而不区分左键和右键。 应用程序还可以使用以下虚拟键代码常量作为索引来区分这些键的左实例和右实例：

<code>VK_LSHIFT</code>
<code>VK_RSHIFT</code>
<code>VK_LCONTROL</code>
<code>VK_RCONTROL</code>
<code>VK_LMENU</code>
<code>VK_RMENU</code>

这些左右区分常量只能通过 `GetKeyboardState`、`SetKeyboardState`、`GetAsyncKeyState`、`GetKeyState` 和 `MapVirtualKey` 函数对应用程序可用。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-rawinput-l1-1-0 ()

另请参阅

- [GetAsyncKeyState](#)
 - [GetKeyState](#)
 - [GetKeyboardState](#)
 - [GetSystemMetrics](#)
 - [MapVirtualKey](#)
 - [SetKeyboardState](#)
 - [键盘输入](#)
-

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getKeyboardType 函数 (winuser.h)

项目2024/03/04

检索有关当前键盘的信息。

语法

C++

```
int GetKeyboardType(  
    [in] int nTypeFlag  
>;
```

参数

[in] nTypeFlag

类型: int

要检索的键盘信息的类型。此参数的取值可为下列值之一：

[] 展开表

值	含义
0	键盘类型
1	键盘子类型
2	键盘上的功能键数

返回值

类型: int

如果函数成功，则返回值指定请求的信息。

如果函数失败且 *nTypeFlag* 不为 1，则返回值为 0；当 *nTypeFlag* 为 1 (键盘子类型) 时，0 是有效的返回值。要获得更多的错误信息，请调用 GetLastError。

注解

有效的键盘类型为：

 展开表

值	说明
0x4	增强的 101 或 102 键键盘 (和兼容)
0x7	日语键盘
0x8	朝鲜语键盘
0x51	未知类型或 HID 键盘

键盘子类型是原始设备制造商 (OEM) 依赖值。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

[键盘输入函数](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

getKeyNameTextA 函数 (winuser.h)

项目2023/11/15

检索表示键的名称的字符串。

语法

C++

```
int GetKeyNameTextA(
    [in] LONG lParam,
    [out] LPSTR lpString,
    [in] int cchSize
);
```

参数

[in] lParam

类型: LONG

键盘消息的第二个参数 (, 例如要处理的 [WM_KEYDOWN](#))。 函数解释 *lParam* 中的以下位位置。

Bits	含义
16-	扫描代码。 此值取决于 OEM。
23	
24	指定键是否为扩展键, 例如出现在增强型 101 或 102 键键盘上的右侧 ALT 和 CTRL 键。 如果是扩展键, 则值为 1; 否则为 0。
25	“不在乎”位。 例如, 调用此函数的应用程序设置此位以指示该函数不应区分左右 Ctrl 和 SHIFT 键。

如需更多详细信息, 请参阅[击键消息标志](#)。

[out] lpString

类型: LPTSTR

将接收密钥名称的缓冲区。

[in] cchSize

类型: int

密钥名称的最大长度 (以字符为单位) , 包括终止 null 字符。 (此参数应等于 *lpString* 参数指向的缓冲区的大小。)

返回值

类型: int

如果函数成功, 则将以 null 结尾的字符串复制到指定的缓冲区中, 返回值是字符串的长度 (以字符为单位), 不计算终止 null 字符。

如果函数失败, 则返回值为零。要获得更多的错误信息, 请调用 GetLastError。

注解

键名称字符串的格式取决于当前键盘布局。

对于名称长于单个字符的键, 键盘布局以字符串的形式维护名称列表。键名称根据 [当前活动的键盘布局](#) 进行转换, 因此函数可能会针对不同的 [键盘布局](#) 返回不同的结果。

字符键的名称是字符本身。死键的名称已完整拼写。

此方法可能无法正常使用某些 (生成多个字符的 [键盘布局](#), 即连字) 和/或单键打印的补充 Unicode 字符。此外, 映射到 "A" 的键。 "Z" [虚拟键代码](#) 转换为大写 'A'。'Z' 字符, 而不考虑当前键盘布局。在这种情况下, 请使用 [ToUnicode](#) 或 [ToUnicodeEx](#) 方法。

winuser.h 标头将 GetKeyNameText 定义为别名, 该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配, 从而导致编译或运行时错误。有关详细信息, 请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

Library	User32.lib
DLL	User32.dll

请参阅

[键盘输入](#)

[键盘布局](#)

[键盘布局示例](#)

[ToUnicode](#)

[ToUnicodeEx](#)

反馈

此页面是否有帮助?

 是

 否

getKeyState 函数 (winuser.h)

项目2023/08/27

检索指定虚拟键的状态。 状态指定键是向上、向下还是切换，(打开、关闭—每次按下键时交替)。

语法

C++

```
SHORT GetKeyState(  
    [in] int nVirtKey  
);
```

参数

[in] nVirtKey

类型: int

虚拟密钥。 如果所需的虚拟键是字母或数字 (A 到 Z、a 到 z 或 0 到 9)，则必须将 *nVirtKey* 设置为该字符的 ASCII 值。 对于其他密钥，它必须是虚拟密钥代码。

如果使用非英语键盘布局，则使用值在 ASCII A 到 Z 和 0 到 9 范围内的虚拟键来指定大多数字符键。 例如，对于德语键盘布局，ASCII O (0x4F) 值虚拟键是指“o”键，而 VK_OEM_1 表示“o with umlaut”键。

返回值

类型: SHORT

返回值指定指定虚拟密钥的状态，如下所示：

- 如果高位为 1，则键关闭;否则，它已启动。
- 如果低序位为 1，则切换键。如果某个键 (如 CAPS LOCK 键) 处于打开状态，则会将其切换。如果低序位为 0，则键处于关闭状态并取消键。切换键的指示灯 (，如果键盘上的任何) 在切换键时将亮起，在取消切换键时处于关闭状态。

注解

当线程从其消息队列读取密钥消息时，此函数返回的密钥状态会发生变化。状态不反映与硬件关联的中断级别状态。使用 [GetAsyncKeyState](#) 函数检索该信息。

应用程序调用 [GetKeyState](#) 以响应键盘输入消息。此函数检索生成输入消息时键的状态。

若要检索所有虚拟密钥的状态信息，请使用 [GetKeyboardState](#) 函数。

应用程序可以使用 [虚拟密钥代码](#) 常量 `VK_SHIFT`、`VK_CONTROL` 和 `VK_MENU` 作为 *nVirtKey* 参数的值。这会提供 SHIFT、CTRL 或 ALT 键的状态，而不区分左键和右键。应用程序还可以使用以下虚拟键代码常量作为 *nVirtKey* 的值，以区分这些键的左实例和右实例：

`VK_LSHIFT` `VK_RSHIFT` `VK_LCONTROL` `VK_RCONTROL` `VK_LMENU` `VK_RMENU` 这些左右区分常量仅通过 [GetKeyboardState](#)、[SetKeyboardState](#)、[GetAsyncKeyState](#)、[GetKeyState](#) 和 [MapVirtualKey](#) 函数对应用程序可用。

示例

有关示例，请参阅 [显示键盘输入](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [MapVirtualKey](#)
- [SetKeyboardState](#)

- 键盘输入
-

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

getLastInputInfo 函数 (winuser.h)

项目2023/08/27

检索最后一个输入事件的时间。

语法

C++

```
BOOL GetLastInputInfo(  
    [out] PLASTINPUTINFO plii  
) ;
```

参数

[out] plii

类型: PLASTINPUTINFO

指向 [LASTINPUTINFO](#) 结构的指针，该结构接收最后一个输入事件的时间。

返回值

类型: BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。

注解

此函数可用于输入空闲检测。但是，`GetLastInputInfo` 不会在所有正在运行的会话中提供系统范围的用户输入信息。相反，`GetLastInputInfo` 仅为调用函数的会话提供特定于会话的用户输入信息。

收到最后一个输入事件时 ([LASTINPUTINFO](#)) 不保证是增量的。在某些情况下，该值可能小于先前事件的时钟周期计数。例如，这可能由原始输入线程和桌面线程之间的计时间隔或 [SendInput](#) 引发的事件（提供自己的时钟周期计数）导致。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[键盘输入](#)

[LASTINPUTINFO](#)

引用

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IsWindowEnabled 函数 (winuser.h)

项目2024/03/04

确定是否针对鼠标和键盘输入启用指定的窗口。

语法

C++

```
BOOL IsWindowEnabled(  
    [in] HWND hWnd  
>;
```

参数

[in] hWnd

类型: HWND

要测试的窗口的句柄。

返回值

类型: BOOL

如果启用窗口，则返回值为非零值。

如果未启用窗口，则返回值为零。

注解

子窗口仅当同时启用且可见时才接收输入。

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]

要求	值
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-window-l1-1-4 ()

请参阅

概念性

[EnableWindow](#)

[IsWindowVisible](#)

[键盘输入](#)

引用

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

keybd_event 函数 (winuser.h)

项目2023/08/27

合成键击。 系统可以使用这种合成的击键来生成 WM_KEYUP 或 WM_KEYDOWN 消息。 键盘驱动程序的中断处理程序调用 keybd_event 函数。

注意 此函数已被取代。 请改用 SendInput。

语法

C++

```
void keybd_event(
    [in] BYTE      bVk,
    [in] BYTE      bScan,
    [in] DWORD     dwFlags,
    [in] ULONG_PTR dwExtraInfo
);
```

参数

[in] bVk

类型: BYTE

虚拟密钥代码。 代码必须是 1 到 254 范围内的值。 有关完整列表，请参阅 [虚拟密钥代码](#)。

[in] bScan

类型: BYTE

密钥的硬件扫描代码。

[in] dwFlags

类型: DWORD

控制函数操作的各个方面。 此参数可使用以下一个或多个值。

值	含义
---	----

KEYEVENTF_EXTENDEDKEY 0x0001	如果指定，则扫描代码前面有一个前缀字节，其值0xE0 (224)。
KEYEVENTF_KEYUP 0x0002	如果指定，则释放密钥。如果未指定，则键被按下。

[in] dwExtraInfo

类型: **ULONG_PTR**

与键笔划关联的附加值。

返回值

无

备注

应用程序可以模拟按下 PRINTSCRN 键以获取屏幕快照并将其保存到剪贴板。为此，请调用 **keybd_event**，并将 *bVk* 参数设置为 **VK_SNAPSHOT**。

示例

以下示例程序使用 **keybd_event** **VK_NUMLOCK** 虚拟键切换 NUM LOCK 指示灯。它采用一个布尔值，该值指示是应关闭灯 (**FALSE**)，还是在 (**TRUE**) 上关闭。相同的技术可用于 CAPS LOCK 密钥 (**VK_CAPITAL**) 和 SCROLL LOCK 键 (**VK_SCROLL**)。

```
#include <windows.h>

void SetNumLock( BOOL bState )
{
    BYTE keyState[256];

    GetKeyboardState((LPBYTE)&keyState);
    if( (bState && !(keyState[VK_NUMLOCK] & 1)) ||
        (!bState && (keyState[VK_NUMLOCK] & 1)) )
    {
        // Simulate a key press
        keybd_event( VK_NUMLOCK,
                    0x45,
                    KEYEVENTF_EXTENDEDKEY | 0,
                    0 );
    }
}
```

```

    // Simulate a key release
    keybd_event( VK_NUMLOCK,
                 0x45,
                 KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP,
                 0);
}

void main()
{
    SetNumLock( TRUE );
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [GetSystemMetrics](#)
- [MapVirtualKey](#)
- [SetKeyboardState](#)
- [键盘输入](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

LoadKeyboardLayoutA 函数 (winuser.h)

项目2024/03/04

将新的输入区域设置标识符（以前称为键盘布局）加载到系统中。

在Windows 8之前：一次可以加载多个输入区域设置标识符，但每个进程一次只能有一个处于活动状态。加载多个输入区域设置标识符可以在它们之间快速切换。

从Windows 8开始：为整个系统加载输入区域设置标识符。如果当前进程不拥有具有键盘焦点的窗口，则此函数不起作用。

语法

C++

```
HKL LoadKeyboardLayoutA(
    [in] LPCSTR pwszKLID,
    [in] UINT   Flags
);
```

参数

[in] pwszKLID

类型： LPCTSTR

要加载的输入区域设置标识符的名称。此名称是由语言 [标识符](#)（低字）的十六进制值和（高字）的设备标识符组成的字符串。例如，美国英语的语言标识符为 0x0409，因此主要美国英语布局名为“00000409”。美国英语布局（的变体（如 Dvorak 布局））命名为“00010409”、“00020409”等。

有关 Windows 提供的输入布局列表，请参阅 [Windows 的键盘标识符和输入法编辑器](#)。

[in] Flags

类型： UINT

指定如何加载输入区域设置标识符。此参数可使用以下一个或多个值。

 展开表

值	含义
---	----

KLF_ACTIVATE 0x00000001	<p>在 Windows 8 之前：如果尚未加载指定的输入区域设置标识符，函数将加载并激活当前线程的输入区域设置标识符。</p> <p>从 Windows 8 开始：如果尚未加载指定的输入区域设置标识符，函数将加载并激活系统的输入区域设置标识符。</p>
KLF_NOTESELLSHELL 0x00000080	<p>在 Windows 8 之前：防止 <code>ShellProc</code> 挂钩过程在加载新的输入区域设置标识符时接收 <code>HSHELL_LANGUAGE</code> 挂钩代码。当应用程序逐个加载多个输入区域设置标识符时，通常会使用此值。将此值应用于除最后一个输入区域设置标识符之外的所有输入区域设置标识符会延迟 shell 的处理，直到添加所有输入区域设置标识符。</p> <p>从 Windows 8 开始：在此方案中，为整个系统设置最后一个输入区域设置标识符。</p>
KLF_REORDER 0x00000008	<p>在 Windows 8 之前：将指定的输入区域设置标识符移动到输入区域设置标识符列表的头，使该区域设置标识符成为当前线程的活动区域设置标识符。即使未提供 <code>KLF_ACTIVATE</code>，此值也会对输入区域设置标识符列表重新排序。</p> <p>从 Windows 8 开始：将指定的输入区域设置标识符移动到输入区域设置标识符列表的头，使该区域设置标识符成为系统的活动区域设置标识符。即使未提供 <code>KLF_ACTIVATE</code>，此值也会对输入区域设置标识符列表重新排序。</p>
KLF_REPLACELANG 0x00000010	如果新的输入区域设置标识符与当前输入区域设置标识符具有相同的语言标识符，则新的输入区域设置标识符会将当前区域设置标识符替换为该语言的输入区域设置标识符。如果未提供此值，并且输入区域设置标识符具有相同的语言标识符，则不会替换当前输入区域设置标识符，并且函数返回 <code>NULL</code> 。
KLF_SUBSTITUTE_OK 0x00000002	将指定的输入区域设置标识符替换为用户首选的另一个区域设置。系统从此标志集开始，建议应用程序始终使用此标志。仅当注册表项 <code>HKEY_CURRENT_USER\Keyboard Layout\Substitutes</code> 显式定义替换区域设置时，才会发生替换。例如，如果键包含值为“00010409”的值名称“00000409”，则加载美国版式（“00000409”）会导致美联航 States-Dvorak 布局（“00010409”）改为加载。系统在启动时使用 <code>KLF_SUBSTITUTE_OK</code> ，建议所有应用程序在加载输入区域设置标识符时都使用此值，以确保选择用户的首选项。
KLF_SETFORPROCESS 0x00000100	<p>在 Windows 8 之前：此标志仅对 <code>KLF_ACTIVATE</code> 有效。激活整个进程的指定输入区域设置标识符，并将 <code>WM_INPUTLANGCHANGE</code> 消息发送到当前线程的“焦点”。</p>

或“活动”窗口。通常，`LoadKeyboardLayout` 仅激活当前线程的输入区域设置标识符。

从 Windows 8 开始：不使用此标志。如果当前进程拥有具有键盘焦点的窗口，`LoadKeyboardLayout` 始终会激活整个系统的输入区域设置标识符。

`KLF_UNLOADPREVIOUS`

不支持此标志。请改用 [UnloadKeyboardLayout](#) 函数。

返回值

类型： `HKL`

如果函数成功，则返回值为与 `pwszKLID` 中指定的名称对应的输入区域设置标识符。如果没有匹配的区域设置可用，则返回值是系统的默认语言。

如果函数失败，则返回值为 `NULL`。如果从应用程序目录加载布局库，则可能会发生这种情况。

要获得更多的错误信息，请调用 `GetLastError`。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(输入法)或任何形式的输入。

应用程序可以而且通常会加载语言的默认输入区域设置标识符或 IME，并且可以通过仅指定语言标识符的字符串版本来执行此操作。如果应用程序要加载特定的区域设置或 IME，它应读取注册表以确定要传递给 `LoadKeyboardLayout` 的特定输入区域设置标识符。在这种情况下，激活区域设置的默认输入区域设置标识符的请求将激活第一个匹配的区域设置标识符。应使用从 [GetKeyboardLayout](#) 或 `LoadKeyboardLayout` 返回的显式输入区域设置标识符激活特定的 IME。

在 Windows 8 之前：此函数仅影响当前进程或线程的布局。

从 Windows 8 开始：此函数会影响整个系统的布局。

① 备注

`winuser.h` 标头将 `LoadKeyboardLayout` 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

[ActivateKeyboardLayout](#)

概念性

[GetKeyboardLayoutName](#)

[键盘输入](#)

[MAKELANGID](#)

其他资源

引用

[UnloadKeyboardLayout](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

mapVirtualKeyA 函数 (winuser.h)

项目2024/02/23

将虚拟键代码转换为（映射到）扫描代码或字符值，或将扫描代码转换为虚拟键代码。

语法

C++

```
UINT MapVirtualKeyA(  
    [in] UINT uCode,  
    [in] UINT uMapType  
) ;
```

参数

[in] uCode

类型：UINT

虚拟密钥代码或扫描密钥的代码。此值的解释方式取决于 *uMapType* 参数的值。

[in] uMapType

类型：UINT

要执行的转换。此参数的值取决于 *uCode* 参数的值。

[] 展开表

值	含义
MAPVK_VK_TO_VSC 0	<i>uCode</i> 参数是一个虚拟键代码，并转换为扫描代码。如果它是不区分左键和右键的虚拟键代码，则返回左侧扫描代码。如果没有转换，则函数返回 0。
MAPVK_VSC_TO_VK 1	<i>uCode</i> 参数是一个扫描代码，被转换为不区分左键和右键的虚拟键代码。如果没有转换，则函数返回 0。 Windows Vista 及更高版本： <i>uCode</i> 值的高字节可以包含指定扩展扫描代码 0xe0 或 0xe1。
MAPVK_VK_TO_CHAR 2	<i>uCode</i> 参数是一个虚拟键代码，在返回值的低序字中转换为非移位字符值。死键（音调符号）通过设置返回值的顶位来指示。如果没有转换，则函数返回 0。请参阅“备注”。

值	含义
MAPVK_VSC_TO_VK_EX 3	<i>uCode</i> 参数是扫描代码，并转换为区分左键和右键的虚拟键代码。如果没有转换，则函数返回 0。 Windows Vista 及更高版本： <i>uCode</i> 值的高字节可以包含指定扩展扫描代码 0xe0 或 0xe1。
MAPVK_VK_TO_VSC_EX 4	Windows Vista 及更高版本： <i>uCode</i> 参数是一个虚拟键代码，并转换为扫描代码。如果它是不区分左键和右键的虚拟键代码，则返回左侧扫描代码。如果扫描代码是扩展扫描代码，则返回值的高字节将包含指定扩展扫描代码 0xe0 或 0xe1。如果没有转换，则函数返回 0。

返回值

类型： **UINT**

返回值为扫描代码、虚拟键代码或字符值，具体取决于 *uCode* 和 *uMapType* 的值。如果没有转换，则返回值为零。

注解

若要指定用于转换指定代码的键盘布局的句柄，请使用 **MapVirtualKeyEx** 函数。

应用程序可以使用 **MapVirtualKey** 将扫描代码转换为虚拟键代码常量 **VK_SHIFT**、**VK_CONTROL** 和 **VK_MENU**，反之亦然。这些转换不区分 SHIFT、CTRL 或 Alt 键的左侧和右侧实例。

应用程序可以通过调用将 *uCode* 设置为以下虚拟键代码常量之一的 **MapVirtualKey** 来获取对应于其中一个键的左侧或右侧实例的扫描代码：

- **VK_LSHIFT**
- **VK_RSHIFT**
- **VK_LCONTROL**
- **VK_RCONTROL**
- **VK_LMENU**
- **VK_RMENU**

这些左右区分常量仅通过 **GetKeyboardState**、**SetKeyboardState**、**GetAsyncKeyState**、**GetKeyState**、**MapVirtualKey** 和 **MapVirtualKeyEx** 函数对应用程序可用。有关虚拟密钥代码的完整列表，请参阅 [虚拟密钥代码](#)。

在 **MAPVK_VK_TO_CHAR** 模式下，[虚拟键代码](#) 为“A”。Z 键转换为大写的“A”。“Z' 字符，而不考虑当前键盘布局。如果要将虚拟键代码转换为相应的字符，请使用 [ToAscii](#) 函数。

① 备注

winuser.h 标头将 MapVirtualKey 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

 展开表

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [GetSystemMetrics](#)
- [MapVirtualKey](#)
- [SetKeyboardState](#)
- [键盘输入](#)
- [键盘输入概述](#)

反馈

此页面是否有帮助？

 是

 否

mapVirtualKeyExA 函数 (winuser.h)

项目2024/03/04

将虚拟键代码转换为（映射到）扫描代码或字符值，或将扫描代码转换为虚拟键代码。该函数使用输入语言和输入区域设置标识符转换代码。

语法

C++

```
UINT MapVirtualKeyExA(
    [in]             UINT uCode,
    [in]             UINT uMapType,
    [in, out, optional] HKL dwhkl
);
```

参数

[in] uCode

类型：UINT

虚拟密钥代码或扫描密钥的代码。此值的解释方式取决于 *uMapType* 参数的值。

[in] uMapType

类型：UINT

要执行的转换。此参数的值取决于 *uCode* 参数的值。

[] 展开表

值	含义
MAPVK_VK_TO_VSC 0	<i>uCode</i> 参数是一个虚拟键代码，并转换为扫描代码。如果它是不区分左键和右键的虚拟键代码，则返回左侧扫描代码。如果没有转换，则函数返回 0。
MAPVK_VSC_TO_VK 1	<i>uCode</i> 参数是扫描代码，并转换为不区分左键和右键的虚拟键代码。如果没有转换，则函数返回 0。 Windows Vista 及更高版本： <i>uCode</i> 值的高字节可以包含指定扩展扫描代码 0xe0 或 0xe1。

值	含义
MAPVK_VK_TO_CHAR 2	<i>uCode</i> 参数是一个虚拟键代码，在返回值的低序字中转换为未移位的字符值。（音调符号）的死键通过设置返回值的顶部位来指示。如果没有转换，则函数返回 0。请参阅“备注”。
MAPVK_VSC_TO_VK_EX 3	<i>uCode</i> 参数是扫描代码，并转换为区分左键和右键的虚拟键代码。如果没有转换，则函数返回 0。 Windows Vista 及更高版本： <i>uCode</i> 值的高字节可以包含指定扩展扫描代码 0xe0 或 0xe1。
MAPVK_VK_TO_VSC_EX 4	Windows Vista 及更高版本： <i>uCode</i> 参数是一个虚拟键代码，并转换为扫描代码。如果它是不区分左键和右键的虚拟键代码，则返回左侧扫描代码。如果扫描代码是扩展扫描代码，则返回值的高字节将包含指定扩展扫描代码 0xe0 或 0xe1。如果没有转换，则函数返回 0。

[in, out, optional] dwhk1

类型： HKL

输入用于转换指定代码的区域设置标识符。此参数可以是 [以前由 LoadKeyboardLayout 函数返回的任何输入区域设置标识符](#)。

返回值

类型： UINT

返回值为扫描代码、虚拟键代码或字符值，具体取决于 *uCode* 和 *uMapType* 的值。如果没有转换，则返回值为零。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器（输入法）或任何其他输入形式。

应用程序可以使用 `MapVirtualKeyEx` 将扫描代码转换为虚拟密钥代码常量 `VK_SHIFT`、`VK_CONTROL` 和 `VK_MENU`，反之亦然。这些转换不区分 SHIFT、CTRL 或 ALT 键的左实例和右键实例。

应用程序可以通过调用 *uCode* 设置为以下虚拟键代码常量之一的 `MapVirtualKeyEx` 来获取对应于其中一个键的左侧或右侧实例的扫描代码：

- `VK_LSHIFT`
- `VK_RSHIFT`
- `VK_LCONTROL`

- VK_RCONTROL
- VK_LMENU
- VK_RMENU

这些左右区分常量只能通过 [GetKeyboardState](#)、[SetKeyboardState](#)、[GetAsyncKeyState](#)、[GetKeyState](#)、[MapVirtualKey](#) 和 [MapVirtualKeyEx](#) 函数提供给应用程序。有关虚拟密钥代码的完整列表，请参阅 [虚拟密钥代码](#)。

在 [MAPVK_VK_TO_CHAR](#) 模式下，[虚拟密钥代码](#)为“A”。“Z 键转换为大写“A”。“Z' 字符，而不考虑当前键盘布局。如果要将虚拟密钥代码转换为相应的字符，请使用 [ToAscii](#) 函数。

① 备注

winuser.h 标头将 [MapVirtualKeyEx](#) 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名的使用与非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [GetSystemMetrics](#)

- [MapVirtualKey](#)
 - [SetKeyboardState](#)
 - [LoadKeyboardLayout](#)
 - [键盘输入](#)
 - [键盘输入概述](#)
-

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

oemKeyScan 函数 (winuser.h)

项目2024/03/04

将 OEMASCII 代码 0 到 0x0FF 映射到 OEM 扫描代码和偏移状态。该函数提供的信息允许程序通过模拟键盘输入将 OEM 文本发送到另一个程序。

语法

C++

```
DWORD OemKeyScan(  
    [in] WORD wOemChar  
);
```

参数

[in] wOemChar

类型： WORD

OEM 字符的 ASCII 值。

返回值

类型： DWORD

返回值的低序字包含 OEM 字符的扫描代码，高序字包含移位状态，可以是以下位的组合。

展开表

bit	说明
1	按下任一 SHIFT 键。
2	按下任一 CTRL 键。
4	按下任一 ALT 键。
8	按下汉卡库键。
16	由键盘布局驱动程序) 定义的保留 (。

如果使用当前键盘布局的单个击键无法生成字符，则返回值为 -1。

注解

此函数不为需要 Ctrl+ALT 或死键的字符提供翻译。 必须通过使用 ALT+ 键盘机制模拟输入来复制此函数未转换的字符。 NUMLOCK 密钥必须处于关闭状态。

此函数不提供使用当前键盘布局一次击键键入的字符的翻译，例如具有音调符号的字符需要死键。 可以使用 ALT+ 键盘机制模拟此函数未翻译的字符。 NUMLOCK 密钥必须打开。

此函数是使用 [VkKeyScan](#) 函数实现的。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[键盘输入](#)

引用

[VkKeyScan](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

registerHotKey 函数 (winuser.h)

项目2023/08/27

定义系统范围内的热键。

语法

C++

```
BOOL RegisterHotKey(
    [in, optional] HWND hWnd,
    [in]           int id,
    [in]           UINT fsModifiers,
    [in]           UINT vk
);
```

参数

[in, optional] hWnd

类型: HWND

窗口句柄，该窗口将接收由热键生成的 [WM_HOTKEY](#) 消息。如果此参数为 NULL，[WM_HOTKEY](#) 消息将发布到调用线程的消息队列，并且必须在消息循环中进行处理。

[in] id

类型: int

热键的标识符。如果 *hWnd* 参数为 NULL，则热键与当前线程关联，而不是与特定窗口相关联。如果已存在具有相同 *hWnd* 和 *id* 参数的热键，请参阅所执行的操作的备注。

[in] fsModifiers

类型: UINT

必须与 *vk* 参数指定的键组合按下才能生成 [WM_HOTKEY](#) 消息的键。*fsModifiers* 参数可以是以下值的组合。

值	含义
MOD_ALT 0x0001	必须按住任一 ALT 键。

MOD_CONTROL 0x0002	必须按住任一 CTRL 键。
MOD_NOREPEAT 0x4000	更改热键行为，使键盘自动重复不产生多个热键通知。 Windows Vista: 不支持此标志。
MOD_SHIFT 0x0004	必须按住任一 SHIFT 键。
MOD_WIN 0x0008	按住了任一 WINDOWS 键。这些键标有 Windows 徽标。 涉及 WINDOWS 键的键盘快捷方式保留供操作系统使用。

[in] vk

类型: **UINT**

热键的虚拟密钥代码。请参阅 [虚拟密钥代码](#)。

返回值

类型: **BOOL**

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 [GetLastError](#)。

注解

按下某个键时，系统会查找与所有热键的匹配项。找到匹配项后，系统会将 [WM_HOTKEY](#) 消息发布到与热键关联的窗口的消息队列。如果热键与窗口不关联，则 [WM_HOTKEY](#) 消息将发布到与热键关联的线程。

此函数无法将热键与其他线程创建的窗口相关联。

如果为热键指定的击键已由另一个热键注册，[RegisterHotKey](#) 将失败。

如果已存在具有相同 *hWnd* 和 *id* 参数的热键，它将与新的热键一起维护。应用程序必须显式调用 [UnregisterHotKey](#) 才能取消注册旧的热键。

Windows Server 2003: 如果已存在具有相同 *hWnd* 和 *id* 参数的热键，则由新的热键替换。

F12 键保留供调试器随时使用，因此不应将其注册为热键。即使未调试应用程序，也会保留 F12，以防内核模式调试器或实时调试器驻留。

应用程序必须在0x0000到0xBFFF的范围内指定 ID 值。共享 DLL 必须在0xC000到 globalAddAtom 函数) 返回的范围0xFFFF (指定值。为了避免与其他共享 DLL 定义的热键标识符冲突, DLL 应使用 GlobalAddAtom 函数获取热键标识符。

示例

以下示例演示如何将 RegisterHotKey 函数与 MOD_NOREPEAT 标志一起使用。在此示例中, 为main线程注册了热键“Alt+b”。按下热键时, 线程将收到 WM_HOTKEY 消息, 该消息将在 GetMessage 调用中获取。由于此示例使用 *fsModifiers* 的 MOD_NOREPEAT 值 MOD_ALT, 因此, 当释放“b”键, 然后在按下“Alt”键时再次按下时, 线程才会收到另一条 WM_HOTKEY 消息。

C++

```
#include "stdafx.h"

int _cdecl _tmain (
    int argc,
    TCHAR *argv[])
{
    if (RegisterHotKey(
        NULL,
        1,
        MOD_ALT | MOD_NOREPEAT,
        0x42)) //0x42 is 'b'
    {
        _tprintf(_T("Hotkey 'ALT+b' registered, using MOD_NOREPEAT
flag\n"));
    }

    MSG msg = {0};
    while (GetMessage(&msg, NULL, 0, 0) != 0)
    {
        if (msg.message == WM_HOTKEY)
        {
            _tprintf(_T("WM_HOTKEY received\n"));
        }
    }

    return 0;
}
```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GlobalAddAtom](#)

[键盘输入](#)

引用

[为当前应用注册热键 \(CSRegisterHotkey\)](#)

[为当前应用注册热键 \(CppRegisterHotkey\)](#)

[为当前应用注册热键 \(VBRegisterHotkey\)](#)

示例

[UnregisterHotKey](#)

[WM_HOTKEY](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

sendInput 函数 (winuser.h)

项目2023/08/28

合成键击、鼠标动作和按钮单击。

语法

C++

```
UINT SendInput(
    [in] UINT    cInputs,
    [in] LPINPUT pInputs,
    [in] int     cbSize
);
```

参数

[in] cInputs

类型: **UINT**

pInputs 数组中的结构数。

[in] pInputs

类型: **LPINPUT**

INPUT 结构的数组。 每个结构都表示要插入键盘或鼠标输入流的事件。

[in] cbSize

类型: **int**

INPUT 结构的大小 (以字节为单位)。 如果 *cbSize* 不是 **INPUT** 结构的大小，则函数将失败。

返回值

类型: **UINT**

函数返回成功插入键盘或鼠标输入流的事件数。 如果函数返回零，则表示输入已被另一个线程阻止。 要获得更多的错误信息，请调用 **GetLastError**。

此函数在 UIPI 阻止时失败。请注意，[GetLastError](#) 和返回值都不会指示失败是由 UIPI 阻止引起的。

注解

此函数受 UIPI 约束。仅允许应用程序将输入注入到完整性级别相等或更低级别的应用程序。

[SendInput](#) 函数将 [INPUT](#) 结构中的事件串行插入键盘或鼠标输入流。这些事件不会与用户(键盘或鼠标)插入的其他键盘或鼠标输入事件，或者通过调用 [keybd_event](#)、[mouse_event](#) 或对 [SendInput](#) 的其他调用插入。

此函数不会重置键盘的当前状态。调用函数时已按下的任何键都可能会干扰此函数生成的事件。若要避免此问题，请使用 [GetAsyncKeyState](#) 函数检查键盘的状态，并根据需要进行更正。

由于触摸键盘使用 `winnls.h` 中定义的代理宏将输入发送到系统，因此键盘事件挂钩上的侦听器必须解码源自触摸键盘的输入。有关详细信息，请参阅[代理项和补充字符](#)。

辅助功能应用程序可以使用 [SendInput](#) 注入与 shell 处理的应用程序启动快捷键对应的按键。此功能不保证适用于其他类型的应用程序。

示例

C++

```
//*****
// Sends Win + D to toggle to the desktop
//*****
void ShowDesktop()
{
    OutputString(L"Sending 'Win-D'\r\n");
    INPUT inputs[4] = {};
    ZeroMemory(inputs, sizeof(inputs));

    inputs[0].type = INPUT_KEYBOARD;
    inputs[0].ki.wVk = VK_LWIN;

    inputs[1].type = INPUT_KEYBOARD;
    inputs[1].ki.wVk = 'D';

    inputs[2].type = INPUT_KEYBOARD;
    inputs[2].ki.wVk = 'D';
    inputs[2].ki.dwFlags = KEYEVENTF_KEYUP;
```

```

inputs[3].type = INPUT_KEYBOARD;
inputs[3].ki.wVk = VK_LWIN;
inputs[3].ki.dwFlags = KEYEVENTF_KEYUP;

UINT uSent = SendInput(ARRAYSIZE(inputs), inputs, sizeof(INPUT));
if (uSent != ARRAYSIZE(inputs))
{
    OutputString(L"SendInput failed: 0x%x\n",
    HRESULT_FROM_WIN32(GetLastError()));
}
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetAsyncKeyState](#)

[INPUT](#)

[键盘输入](#)

引用

[代理项和增补字符](#)

[keybd_event](#)

[mouse_event](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SetActiveWindow 函数 (winuser.h)

项目2023/08/28

激活窗口。 窗口必须附加到调用线程的消息队列。

语法

C++

```
HWND SetActiveWindow(  
    [in] HWND hWnd  
) ;
```

参数

[in] hWnd

类型: HWND

要激活的顶级窗口的句柄。

返回值

类型: HWND

如果函数成功，则返回值是以前处于活动状态的窗口的句柄。

如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。

注解

SetActiveWindow 函数激活窗口，但如果应用程序在后台，则不会激活窗口。 当系统激活窗口时，如果窗口的应用程序位于前台，则窗口将进入 Z 顺序) 的前台 (顶部)。

如果由 hWnd 参数标识的窗口是由调用线程创建的，则调用线程的活动窗口状态将设置为 hWnd。 否则，调用线程的活动窗口状态设置为 NULL。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	ext-ms-win-ntuser-window-l1-1-4 (在 Windows 10 版本 10.0.14393 中引入)

请参阅

概念性

[GetActiveWindow](#)

[键盘输入](#)

引用

[SetForegroundWindow](#)

[WM_ACTIVATE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

SetFocus 函数 (winuser.h)

项目2023/08/28

将键盘焦点设置为指定的窗口。 窗口必须附加到调用线程的消息队列。

语法

C++

```
HWND SetFocus(  
    [in, optional] HWND hWnd  
) ;
```

参数

[in, optional] hWnd

类型: HWND

将接收键盘输入的窗口的句柄。 如果此参数为 NULL，则忽略击键。

返回值

类型: HWND

如果函数成功，则返回值是以前具有键盘焦点的窗口的句柄。 如果 hWnd 参数无效或窗口未附加到调用线程的消息队列，则返回值为 NULL。 若要获取扩展的错误信息，请调用 [GetLastError 函数](#)。

扩展错误ERROR_INVALID_PARAMETER (0x57) 意味着窗口处于禁用状态。

注解

此函数向失去键盘焦点的窗口发送 [WM_KILLFOCUS](#) 消息，向接收键盘焦点的窗口发送 [WM_SETFOCUS](#) 消息。 它还会激活接收焦点的窗口或接收焦点的窗口的父级。

如果窗口处于活动状态但没有焦点，则任何按下的键都生成 [WM_SYSCHAR](#)、[WM_SYSKEYDOWN](#)或 [WM_SYSKEYUP](#) 消息。 如果同时按下VK_MENU键，则会设置消息的 *lParam* 参数的第 30 位。 否则，生成的消息不会设置此位。

通过使用 [AttachThreadInput 函数](#)，线程可以将其输入处理附加到另一个线程。这允许线程调用 [SetFocus](#)，以将键盘焦点设置为附加到另一个线程的消息队列的窗口。

示例

有关示例，请参阅 [初始化对话框](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	ext-ms-win-ntuser-window-l1-1-4 (在 Windows 10 版本 10.0.14393 中引入)

另请参阅

[AttachThreadInput 函数](#)、[GetFocus 函数](#)、[WM_KILLFOCUS](#)、[WM_SETFOCUS](#)、[WM_SYSCHAR](#)、[WM_SYSKEYDOWN](#)、[WM_SYSKEYUP](#)、[键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SetKeyboardState 函数 (winuser.h)

项目2024/03/04

将键盘键状态数组复制到调用线程的键盘输入状态表中。这是由 [GetKeyboardState](#) 和 [GetKeyState](#) 函数访问的同一表。对此表所做的更改不会影响任何其他线程的键盘输入。

语法

C++

```
BOOL SetKeyboardState(  
    [in] LPBYTE lpKeyState  
>;
```

参数

[in] lpKeyState

类型： LPBYTE

指向包含键盘键状态的 256 字节数组的指针。

返回值

类型： BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 [GetLastError](#)。

注解

由于 `SetKeyboardState` 函数会更改调用线程的输入状态，而不是系统的全局输入状态，因此应用程序无法使用 `SetKeyboardState` 设置 NUM LOCK、CAPS LOCK 或 SCROLL LOCK (或日语 KANA) 键盘上的指示灯。可以使用 [SendInput](#) 进行设置或清除以模拟击键。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [GetSystemMetrics](#)
- [MapVirtualKey](#)
- [SetKeyboardState](#)
- [SendInput](#)
- [keybd_event](#)
- [键盘输入](#)

反馈

此页面是否有帮助?



[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

ToAscii 函数 (winuser.h)

项目2023/07/21

将指定的虚拟键代码和键盘状态转换为相应的一个或多个字符。该函数使用输入语言和由键盘布局句柄标识的物理键盘布局转换代码。

若要指定用于转换指定代码的键盘布局的句柄，请使用 ToAsciiEx 函数。

① 备注

此方法可能无法正常使用某些键盘布局，这些 **键盘布局** 可能会生成多个字符（即，只需按一次键即可）和/或补充 Unicode 字符。强烈建议使用可正确处理此类情况的 ToUnicode 或 ToUnicodeEx 方法。

语法

C++

```
int ToAscii(
    [in]          UINT      uVirtKey,
    [in]          UINT      uScanCode,
    [in, optional] const BYTE *lpKeyState,
    [out]         LPWORD    lpChar,
    [in]          UINT      uFlags
);
```

参数

[in] uVirtKey

类型: **UINT**

要转换的虚拟密钥代码。请参阅[虚拟键代码](#)。

[in] uScanCode

类型: **UINT**

要转换的密钥的硬件扫描代码。如果键处于打开状态（未按），则设置此值的高阶位。

[in, optional] lpKeyState

类型: `const BYTE*`

指向包含当前键盘状态的 256 字节数组的指针。 数组中 (字节) 的每个元素都包含一个键的状态。 如果设置了字节的高阶位，则 (按下) 键。

低位 (如果已设置) 表示键已打开。 在此函数中，仅 CAPS LOCK 键的切换位相关。 忽略 NUM LOCK 和 SCROLL LOCK 键的切换状态。

[out] `lpChar`

类型: `LPWORD`

指向缓冲区的指针，该缓冲区接收转换字符 (或打包到单个 WORD 值的两个字符，其中低序字节包含第一个字符，高序字节包含第二个字符)。

[in] `uFlags`

类型: `UINT`

如果菜单处于活动状态，此参数必须为 1，否则为 0。

返回值

类型: `int`

返回值是以下值之一。

返回值	说明
0	指定的虚拟键对键盘的当前状态没有转换。
1	已将一个字符复制到缓冲区。
2	已将两个字符复制到缓冲区。 当键盘布局中存储的死键 (重音符或音调) 不能使用指定的虚拟键组合形成单个字符时，通常会发生这种情况。

备注

提供给 `ToAscii` 函数的参数可能不足以转换虚拟键代码，因为以前的死键存储在键盘布局中。

通常，`ToAscii` 会基于虚拟密钥代码执行转换。但在某些情况下，`uScanCode` 参数的第 15 位可用于区分按键和释放键。扫描代码用于转换 Alt+ 数字组合键。

尽管 NUM LOCK 是影响键盘行为的切换键，但 ToAscii 会忽略切换设置，(*lpKeyState* (`VK_NUMLOCK`) 的低位)，因为 *uVirtKey* 参数单独足以区分光标移动键 (`VK_HOME`、`VK_INSERT` 等) (`VK_DECIMAL`, - `VK_NUMPAD0`-`VK_NUMPAD9`)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[键盘输入](#)

[OemKeyScan](#)

引用

[ToAsciiEx](#)

[ToUnicode](#)

[VkKeyScan](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

toAsciiEx 函数 (winuser.h)

项目2023/07/21

将指定的虚拟键代码和键盘状态转换为相应的一个或多个字符。该函数使用输入语言和由输入区域设置标识符标识的物理键盘布局转换代码。

① 备注

此方法可能无法正常使用某些 **键盘布局**，这些布局可能会生成多个字符 (即在一次按键时) 连字和/或补充 Unicode 字符。强烈建议使用正确处理此类情况的 ToUnicode 或 ToUnicodeEx 方法。

语法

C++

```
int ToAsciiEx(
    [in]           UINT      uVirtKey,
    [in]           UINT      uScanCode,
    [in, optional] const BYTE *lpKeyState,
    [out]          LPWORD    lpChar,
    [in]           UINT      uFlags,
    [in, optional] HKL      dwhkl
);
```

参数

[in] uVirtKey

类型: **UINT**

要转换的虚拟密钥代码。请参阅[虚拟键代码](#)。

[in] uScanCode

类型: **UINT**

要转换的密钥的硬件扫描代码。如果键处于启动 (未按下)，则设置此值的高位。

[in, optional] lpKeyState

类型: **const BYTE***

指向包含当前键盘状态的 256 字节数组的指针。 数组中 (字节) 的每个元素都包含一个键的状态。 如果设置了字节的高阶位，则 (按下) 键。

低位 (如果设置) 指示键已打开。 在此函数中，只有 CAPS LOCK 键的切换位是相关的。 忽略 NUM LOCK 和 SCOLL LOCK 键的切换状态。

[out] lpChar

类型: LPWORD

指向缓冲区的指针，该缓冲区接收已翻译字符 (或两个字符打包到单个 WORD 值中，其中低位字节包含第一个字符，高序字节包含第二个字符)。

[in] uFlags

类型: UINT

如果菜单处于活动状态，则此参数必须为 1，否则为零。

[in, optional] dwhkl

类型: HKL

用于转换代码的输入区域设置标识符。 此参数可以是 [以前由 LoadKeyboardLayout 函数返回的任何输入区域设置标识符](#)。

返回值

类型: int

返回值是以下值之一。

返回值	说明
0	指定的虚拟键没有键盘当前状态的转换。
1	已将一个字符复制到缓冲区。
2	已将两个字符复制到缓冲区。 当键盘布局中存储的死键字符 (重音符或音调符号) 不能使用指定的虚拟键组成一个字符时，通常会发生这种情况。

备注

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(输入法)或任何形式的输入。

提供给 `ToAsciiEx` 函数的参数可能不足以转换虚拟键代码，因为以前的死键存储在键盘布局中。

通常，`ToAsciiEx` 会根据虚拟键代码执行转换。但在某些情况下，`uScanCode` 参数的第 15 位可用于区分按键和释放键。扫描代码用于转换 `ALT+数字组合键`。

尽管 `NUM LOCK` 是影响键盘行为的切换键，但 `ToAsciiEx` 会忽略 `lpKeyState` (`VK_NUMLOCK`) 的低位(切换设置，因为仅使用 `uVirtKey` 参数就足以将光标移动键 (`VK_HOME`、`VK_INSERT` 等) 与数字键 (`VK_DECIMAL` 区分开来，- `VK_NUMPAD0`-`VK_NUMPAD9`)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

键盘输入

[LoadKeyboardLayout](#)

[MapVirtualKeyEx](#)

引用

[ToUnicodeEx](#)

[VkKeyScan](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

toUnicode 函数 (winuser.h)

项目2023/03/15

将指定的虚拟键代码和键盘状态转换为相应的 Unicode 字符。

语法

C++

```
int ToUnicode(
    [in]          UINT      wVirtKey,
    [in]          UINT      wScanCode,
    [in, optional] const BYTE *lpKeyState,
    [out]         LPWSTR   pwszBuff,
    [in]          int       cchBuff,
    [in]          UINT      wFlags
);
```

参数

[in] wVirtKey

类型: **UINT**

要转换的虚拟密钥代码。 请参阅 [虚拟密钥代码](#)。

[in] wScanCode

类型: **UINT**

要转换的密钥的硬件 [扫描代码](#)。 如果键已启动，则设置此值的高序位。

[in, optional] lpKeyState

类型: **const BYTE***

指向包含当前键盘状态的 256 字节数组的指针。 数组中 (字节) 的每个元素都包含一个键的状态。

如果设置了字节的高阶位，则键关闭。 低位（如果设置）指示键已打开。 在此函数中，只有 CAPS LOCK 键的切换位是相关的。 忽略 NUM LOCK 和 SCROLL LOCK 键的切换状态。 有关详细信息，请参阅 [GetKeyboardState](#)。

[out] pwszBuff

类型: LPWSTR

接收已翻译字符或字符作为 UTF-16 代码单元数组的缓冲区。即使变量名称表明该缓冲区以 null 结尾，也可以返回该缓冲区，而不会以 null 结尾。可以使用此方法的返回值来确定写入的字符数。

[in] cchBuff

类型: int

pwszBuff 参数指向的缓冲区的大小 (以字符为单位)。

[in] wFlags

类型: UINT

函数的行为。

如果设置了位 0，则菜单处于活动状态。在此模式下，**不处理 Alt+数字键盘** 组合键。

如果设置了位 2，则键盘状态不会 (Windows 10 版本 1607 及更新)

保留 (到 31 个) 的所有其他位。

返回值

类型: int

函数返回以下值之一。

返回值	说明
<i>value</i> < 0	指定的虚拟键是 (重音符或音调符号) 的 死键 字符。无论键盘布局如何，都会返回此值，即使已键入多个字符并存储在键盘状态中也是如此。如果可能，即使使用 Unicode 键盘布局，函数也已将死键字符的间距版本写入 <i>pwszBuff</i> 指定的缓冲区。例如，函数写入字符锐音符 (U+00B4)，而不是字符组合锐音符 (U+0301)。
0	指定的虚拟键没有键盘当前状态的转换。未向 <i>pwszBuff</i> 指定的缓冲区写入任何内容。
<i>value</i> > 0	一个或多个 UTF-16 代码单元已写入 <i>pwszBuff</i> 指定的缓冲区。返回的 <i>pwszBuff</i> 包含的字符数可能多于返回值指定的字符数。发生这种情况时，任何额外的字符都无效，应忽略。

注解

若要指定用于转换指定代码的键盘布局的句柄，请使用 [ToUnicodeEx](#) 函数。

某些键盘布局可能会返回多个字符和/或补充字符作为 *pwszBuff* 中的代理项对。如果键盘布局中存储的死键字符 (重音符或音调符号) 不能与指定的虚拟键组合成一个字符，则以前输入的死字符可以与当前字符组合。

提供给 [ToUnicodeEx](#) 函数的参数可能不足以转换虚拟键代码，因为以前的 [死键](#) 存储在键盘布局中。

通常，[ToUnicode](#) 基于虚拟键代码执行转换。但在某些情况下，*wScanCode* 参数的第 15 位可用于区分按键和释放键 (例如 alt+numpad 键输入)。

当 [ToUnicode](#) 转换虚拟键代码时，它还会更改内核模式键盘缓冲区的状态。此状态更改会影响死键、连字、[Alt+数字键盘](#) 键输入等。如果与 [TranslateMessage](#) (结合使用，这也可能导致不需要的副作用，这也会更改内核模式键盘缓冲区) 的状态。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

[概念性](#)

[键盘输入](#)

[引用](#)

[ToAscii](#)

[ToUnicodeEx](#)

VkKeyScan

toUnicodeEx 函数 (winuser.h)

项目2023/03/15

将指定的虚拟键代码和键盘状态转换为相应的 Unicode 字符。

语法

C++

```
int ToUnicodeEx(
    [in]           UINT      wVirtKey,
    [in]           UINT      wScanCode,
    [in]           const BYTE *lpKeyState,
    [out]          LPWSTR   pwszBuff,
    [in]           int       cchBuff,
    [in]           UINT      wFlags,
    [in, optional] HKL      dwhkl
);
```

参数

[in] wVirtKey

类型: **UINT**

要转换的虚拟密钥代码。 请参阅 [虚拟密钥代码](#)。

[in] wScanCode

类型: **UINT**

要转换的密钥的硬件 [扫描代码](#)。 如果键已打开，则设置此值的高阶位。

[in] lpKeyState

类型: **const BYTE***

指向包含当前键盘状态的 256 字节数组的指针。 数组中 (字节) 的每个元素都包含一个键的状态。

如果设置了字节的高阶位，则键关闭。 低位（如果已设置）表示键已打开。 在此函数中，仅 CAPS LOCK 键的切换位相关。 忽略 NUM LOCK 和 SCROLL LOCK 键的切换状态。 有关详细信息，请参阅 [GetKeyboardState](#)。

[out] pwszBuff

类型: LPWSTR

接收已翻译字符作为 UTF-16 代码单元数组的缓冲区。即使变量名称表明该缓冲区以 null 结尾，也可以返回该缓冲区，而不会以 null 结尾。可以使用此方法的返回值来确定写入的字符数。

[in] cchBuff

类型: int

pwszBuff 参数指向的缓冲区的大小 (以字符为单位)。

[in] wFlags

类型: UINT

函数的行为。

如果设置了位 0，则菜单处于活动状态。在此模式下，**不处理 Alt+数字键盘** 组合键。

如果设置了第 1 位，**ToUnicodeEx** 除了通常处理键生成事件外，还将转换标记为键中断事件的扫描代码。

如果设置了第 2 位，则键盘状态不会更改 (Windows 10 版本 1607 及更新)

保留 (到 31) 的所有其他位。

[in, optional] dwhkl

类型: HKL

用于转换指定代码的输入区域设置标识符。此参数可以是 [以前由 LoadKeyboardLayout](#) 函数返回的任何输入区域设置标识符。

返回值

类型: int

函数返回以下值之一。

返回值	说明
<i>value</i> < 0	指定的虚拟键是 (重音或音调) 的 死键 字符。无论键盘布局如何，都会返回此值，即使已键入多个字符并存储在键盘状态中也是如此。如果可能，即使使用 Unicode 键盘布

	局，函数也已将死键字符的间距版本写入 <i>pwszBuff</i> 指定的缓冲区。例如，函数写入字符 ACUTE ACCENT (U+00B4)，而不是将 (U+0301) 组合字符。
0	指定的虚拟键对键盘的当前状态没有转换。 <i>pwszBuff</i> 指定的缓冲区未写入任何内容。
<i>value</i> > 0	一个或多个 UTF-16 代码单元已写入 <i>pwszBuff</i> 指定的缓冲区。返回的 <i>pwszBuff</i> 可能包含超过返回值指定的字符数。发生这种情况时，任何额外的字符都无效，应忽略。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器 (IME) 或任何其他形式的输入。

某些键盘布局可能会返回多个字符和/或补充字符作为 *pwszBuff* 中的代理项对。如果键盘布局中存储的死键字符 (重音符或音调符号) 无法与指定的虚拟键组合在一起形成一个字符，则上一个输入的死字符可以与当前字符组合使用。

提供给 **ToUnicodeEx** 函数的参数可能不足以转换虚拟键代码，因为以前的 **死键** 存储在键盘布局中。

通常，**ToUnicodeEx** 会基于虚拟密钥代码执行转换。但在某些情况下，*wScanCode* 参数的第 15 位可用于区分按键和释放键 (例如 alt+numpad 键输入)。

当 **ToUnicodeEx** 转换虚拟键代码时，它还会更改内核模式键盘缓冲区的状态。此状态更改会影响死键、连字、**Alt+数字键盘** 键项等。如果与 **TranslateMessage** (结合使用，这也可能导致意外的副作用，这也会更改内核模式键盘缓冲区) 的状态。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

概念性

[键盘输入](#)

[LoadKeyboardLayout](#)

引用

[ToAsciiEx](#)

[VkKeyScan](#)

UnloadKeyboardLayout 函数 (winuser.h)

项目2023/08/28

卸载输入区域设置标识符（以前称为键盘布局）。

语法

C++

```
BOOL UnloadKeyboardLayout(  
    [in] HKL hkl  
);
```

参数

[in] hkl

类型： HKL

要卸载的输入区域设置标识符。

返回值

类型： BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。 函数可能由于以下原因而失败：

- 传递了无效的输入区域设置标识符。
- 输入区域设置标识符已预加载。
- 输入区域设置标识符正在使用中。

要获得更多的错误信息，请调用 GetLastError。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器（输入法）或任何其他形式的输入。

`UnloadKeyboardLayout` 无法卸载系统默认输入区域设置标识符（如果它是加载的唯一键盘布局）。必须先加载另一个输入区域设置标识符，然后才能卸载默认输入区域设置标识符。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

[ActivateKeyboardLayout](#)

概念性

[GetKeyboardLayoutName](#)

[键盘输入](#)

[LoadKeyboardLayout](#)

引用

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

unregisterHotKey 函数 (winuser.h)

项目2024/03/04

释放以前由调用线程注册的热键。

语法

C++

```
BOOL UnregisterHotKey(  
    [in, optional] HWND hWnd,  
    [in]           int   id  
) ;
```

参数

[in, optional] hWnd

类型: HWND

与要释放的热键关联的窗口的句柄。 如果热键与窗口不关联，则此参数应为 NULL。

[in] id

类型: int

要释放的热键的标识符。

返回值

类型: BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[键盘输入](#)

引用

[RegisterHotKey](#)

[WM_HOTKEY](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

vkKeyScanA 函数 (winuser.h)

项目2023/08/28

[此函数已被 [VkKeyScanEx](#) 函数取代。但是，如果不需要指定键盘布局，仍然可以使用 [VkKeyScan](#)。]

将字符转换为当前键盘的相应虚拟键代码和偏移状态。

语法

C++

```
SHORT VkKeyScanA(  
    [in] CHAR ch  
)
```

参数

[in] ch

类型: TCHAR

要转换为虚拟键代码的字符。

返回值

类型: SHORT

如果函数成功，则返回值的低序字节包含虚拟键代码，高序字节包含 shift 状态，可以是以下标志位的组合。

返回值	说明
1	按下任一 SHIFT 键。
2	按下任一 Ctrl 键。
4	按下任一 Alt 键。
8	按下了 Hankaku 键
16	键盘布局驱动程序) 定义的保留 (。

如果函数找不到转换为传递的字符代码的键，则低序字节和高阶字节都包含 -1。

备注

对于使用右 Alt 键作为移位键的键盘布局 (例如法语键盘布局)，移位状态由值 6 表示，因为右侧 Alt 键在内部转换为 CTRL+Alt。

将忽略通过VK_DIVIDE) VK_NUMPAD0 的数字键盘 (转换。此函数仅用于将字符转换为来自main键盘部分的击键。例如，字符“7”将转换为VK_7，而不是VK_NUMPAD7。

VkKeyScan 由使用 WM_KEYUP 和 WM_KEYDOWN 消息发送字符的应用程序使用。

① 备注

winuser.h 标头将 VkKeyScan 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyNameText](#)

- [GetKeyState](#)
 - [GetKeyboardState](#)
 - [键盘输入](#)
 - [SetKeyboardState](#)
 - [VkKeyScanEx](#)
 - [WM_KEYDOWN](#)
 - [WM_KEYUP](#)
 - [键盘输入](#)
-

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

VkKeyScanExA 函数 (winuser.h)

项目2024/03/04

将字符转换为相应的虚拟键代码和偏移状态。该函数使用输入语言和由输入区域设置标识符标识的物理键盘布局转换字符。

语法

C++

```
SHORT VkKeyScanExA(  
    [in] CHAR ch,  
    [in] HKL dwhkl  
);
```

参数

[in] ch

类型: TCHAR

要转换为虚拟键代码的字符。

[in] dwhkl

类型: HKL

用于转换字符的输入区域设置标识符。此参数可以是[以前由 LoadKeyboardLayout 函数返回的任何输入区域设置标识符](#)。

返回值

类型: SHORT

如果函数成功，则返回值的低序字节包含虚拟键代码，高阶字节包含移位状态，可以是以下标志位的组合。

[] 展开表

返回值	说明
1	按下任一 SHIFT 键。

2	按下任一 CTRL 键。
4	按下任一 ALT 键。
8	按下了 Hankaku 键
16	由键盘布局驱动程序) 定义的保留 (。
32	由键盘布局驱动程序) 定义的保留 (。

如果函数找不到转换为传递的字符代码的键，则低序字节和高阶字节都包含 -1。

注解

输入区域设置标识符是比键盘布局更广泛的概念，因为它还可以包含语音转文本转换器、输入法编辑器(输入法)或任何其他输入形式。

对于使用右 ALT 键作为移位键的键盘布局(例如，法语键盘布局)，移位状态由值 6 表示，因为右侧 ALT 键在内部转换为 CTRL+ALT。

将忽略数字键盘(VK_NUMPAD0 到 VK_DIVIDE)的转换。此函数仅用于将字符转换为 main 键盘部分的击键。例如，字符“7”将转换为 VK_7，而不是 VK_NUMPAD7。

VkKeyScanEx 由使用 [WM_KEYUP](#) 和 [WM_KEYDOWN](#) 消息发送字符的应用程序使用。

① 备注

winuser.h 标头将 VkKeyScanEx 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

[] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]

要求	值
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

另请参阅

- [GetAsyncKeyState](#)
- [GetKeyNameText](#)
- [GetKeyState](#)
- [GetKeyboardState](#)
- [LoadKeyboardLayout](#)
- [SetKeyboardState](#)
- [ToAsciiEx](#)
- [键盘输入](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

键盘输入消息

项目 · 2024/01/28

本节内容

- [WM_GETHOTKEY](#)
- [WM_SETHOTKEY](#)

反馈

此页面是否有帮助?

是

否

WM_GETHOTKEY消息

项目 • 2023/06/13

发送以确定与窗口关联的热键。

C++

```
#define WM_GETHOTKEY 0x0033
```

参数

wParam

未使用;必须为零。

lParam

未使用;必须为零。

返回值

返回值是热键的虚拟键代码和修饰符;如果没有与窗口关联的热键，则返回值为 **NULL**。虚拟键代码位于返回值的低字节中，修饰符位于高字节中。修饰符可以是来自 **CommCtrl.h** 的以下标志的组合。

返回代码/值	说明
HOTKEYF_ALT 0x04	Alt 键
HOTKEYF_CONTROL 0x02	CTRL 键
HOTKEYF_EXT 0x08	扩展键
HOTKEYF_SHIFT 0x01	SHIFT 键

注解

这些热键与 [RegisterHotKey](#) 函数设置的热键无关。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[RegisterHotKey](#)

[WM_SETHOTKEY](#)

概念性

[键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

WM_SETHOTKEY消息

项目 • 2023/06/13

发送到窗口以将热键与窗口相关联。当用户按下热键时，系统将激活窗口。

C++

```
#define WM_SETHOTKEY 0x0032
```

参数

wParam

低序字指定要与窗口关联的虚拟键代码。

高序字可以是来自 CommCtrl.h 的下一个或多个值。

将 *wParam* 设置为 NULL 会删除与窗口关联的热键。

值	含义
HOTKEYF_ALT 0x04	Alt 键
HOTKEYF_CONTROL 0x02	CTRL 键
HOTKEYF_EXT 0x08	扩展键
HOTKEYF_SHIFT 0x01	SHIFT 键

lParam

未使用此参数。

返回值

返回值为下列值之一。

返回值	说明
-1	函数不成功;热键无效。

返回值	说明
0	函数不成功;窗口无效。
1	函数成功，并且没有其他窗口具有相同的热键。
2	函数成功，但另一个窗口已具有相同的热键。

备注

热键不能与子窗口相关联。

`VK_ESCAPE`、`VK_SPACE`和`VK_TAB`是无效的热键。

当用户按下热键时，系统会生成一条 `WM_SYSCOMMAND` 消息，其中 `wParam` 等于 `SC_HOTKEY`，`lParam` 等于窗口的句柄。如果此消息传递给 `DefWindowProc`，则系统会将窗口的最后一个活动弹出窗口(如果它)存在，则窗口本身((如果没有弹出窗口)到前台)。

一个窗口只能有一个热键。如果窗口已有与其关联的热键，则新的热键将替换旧热键。如果多个窗口具有相同的热键，则由热键激活的窗口是随机的。

这些热键与 [RegisterHotKey](#) 设置的热键无关。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[RegisterHotKey](#)

[WM_GETHOTKEY](#)

[WM_SYSCOMMAND](#)

概念性

键盘输入

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

键盘输入通知

项目 · 2023/06/13

本节内容

- WM_ACTIVATE
- WM_APPCOMMAND
- WM_CHAR
- WM_DEADCHAR
- WM_HOTKEY
- WM_KEYDOWN
- WM_KEYUP
- WM_KILLFOCUS
- WM_SETFOCUS
- WM_SYSDEADCHAR
- WM_SYSKEYDOWN
- WM_SYSKEYUP
- WM_UNICHAR

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

WM_ACTIVATE消息

项目 • 2023/06/12

发送到正在激活的窗口和正在停用的窗口。如果窗口使用相同的输入队列，则消息将同步发送，首先发送到要停用的顶级窗口的窗口过程，然后发送到正在激活的顶级窗口的窗口过程。如果窗口使用不同的输入队列，则消息将异步发送，因此会立即激活窗口。

C++

```
#define WM_ACTIVATE 0x0006
```

参数

wParam

低序字指定是激活还是停用窗口。此参数的取值可为下列值之一：高序字指定激活或停用窗口的最小化状态。非零值表示窗口已最小化。

值	含义
WA_ACTIVE 1	例如，通过调用 SetActiveWindow 函数或使用键盘界面选择窗口) (鼠标单击以外的方法激活。
WA_CLICKACTIVE 2	通过鼠标单击激活。
WA_INACTIVE 0	关闭。

lParam

要激活或停用的窗口的句柄，具体取决于 *wParam* 参数的值。如果 *wParam* 的低序字 WA_INACTIVE，则 *lParam* 是激活窗口的句柄。如果 *wParam* 的低序字 WA_ACTIVE 或 WA_CLICKACTIVE，则 *lParam* 是要停用的窗口的句柄。此句柄可以为 NULL。

返回值

如果应用程序处理此消息，则它应返回零。

备注

如果窗口正在激活且未最小化，则 [DefWindowProc](#) 函数会将键盘焦点设置为窗口。如果窗口通过鼠标单击激活，它还会收到 [WM_MOUSEACTIVATE](#) 消息。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[DefWindowProc](#)

[SetActiveWindow](#)

[WM_MOUSEACTIVATE](#)

[WM_NCACTIVATE](#)

概念性

[键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

WM_APPCOMMAND 消息

项目 • 2023/07/01

通知窗口用户生成了应用程序命令事件，例如，使用鼠标单击应用程序命令按钮或在键盘上键入应用程序命令键。

C++

```
#define WM_APPCOMMAND 0x0319
```

参数

wParam

用户在其中单击按钮或按键的窗口的句柄。此窗口可以是接收消息的窗口的子窗口。有关处理此消息的详细信息，请参阅“注解”部分。

lParam

使用以下代码获取 *lParam* 参数中包含的信息。

```
cmd = GET_APPCOMMAND_LPARAM(lParam);
uDevice = GET_DEVICE_LPARAM(lParam);
dwKeys = GET_KEYSTATE_LPARAM(lParam);
```

应用程序命令为 *cmd*，可以是以下任一值。

值	含义
APPCOMMAND_BASS_BOOST 20	启用和禁用低音增强。
APPCOMMAND_BASS_DOWN 19	降低低音。
APPCOMMAND_BASS_UP 21	提高低音。
APPCOMMAND_BROWSER_BACKWARD 1	向后导航。

值	含义
APPCOMMAND_BROWSER_FAVORITES 6	打开收藏夹。
APPCOMMAND_BROWSER_FORWARD 2	向前导航。
APPCOMMAND_BROWSER_HOME 7	导航主页。
APPCOMMAND_BROWSER_REFRESH 3	刷新页面。
APPCOMMAND_BROWSER_SEARCH 5	打开搜索。
APPCOMMAND_BROWSER_STOP 4	停止下载。
APPCOMMAND_CLOSE 31	关闭窗口（而不是应用程序）。
APPCOMMAND_COPY 36	复制所选内容。
APPCOMMAND_CORRECTION_LIST 45	在语音输入过程中错误识别单词时，显示更正列表。
APPCOMMAND_CUT 37	剪切所选内容。
APPCOMMAND_DICTATE_OR_COMMAND_CONTROL_TOGGLE 43	在两种语音输入模式之间切换：听写和命令/控制（向应用程序提供命令或访问菜单）。
APPCOMMAND_FIND 28	打开“查找”对话框。
APPCOMMAND_FORWARD_MAIL 40	转发邮件。
APPCOMMAND_HELP 27	打开“帮助”对话框。
APPCOMMAND_LAUNCH_APP1 17	启动 App1。
APPCOMMAND_LAUNCH_APP2 18	启动 App2。

值	含义
APPCOMMAND_LAUNCH_MAIL 15	打开邮件。
APPCOMMAND_LAUNCH_MEDIA_SELECT 16	转到媒体选择模式。
APPCOMMAND_MEDIA_CHANNEL_DOWN 52	递减频道值，例如，电视或收音机调谐器。
APPCOMMAND_MEDIA_CHANNEL_UP 51	递增频道值，例如，电视或收音机调谐器。
APPCOMMAND_MEDIA_FAST_FORWARD 49	提高流播放速度。这可以通过多种方式实现，例如，使用固定速度或在一系列增加的速度之间切换。
APPCOMMAND_MEDIA_NEXTTRACK 11	转到下一曲目。
APPCOMMAND_MEDIA_PAUSE 47	暂停。如果已暂停，则不执行进一步操作。这是一个没有状态的直接PAUSE命令。如果存在离散的“播放”和“暂停”按钮，则应用程序应对此命令以及APPCOMMAND_MEDIA_PLAY_PAUSE执行操作。
APPCOMMAND_MEDIA_PLAY 46	在当前位置暂停播放。如果已暂停，它将恢复。这是一个没有状态的直接PLAY命令。如果存在离散的“播放”和“暂停”按钮，则应用程序应对此命令以及APPCOMMAND_MEDIA_PLAY_PAUSE执行操作。
APPCOMMAND_MEDIA_PLAY_PAUSE 14	播放或暂停播放。如果存在离散的“播放”和“暂停”按钮，则应用程序应对此命令以及APPCOMMAND_MEDIA_PLAY和APPCOMMAND_MEDIA_PAUSE执行操作。
APPCOMMAND_MEDIA_PREVIOUSTRACK 12	转到上一曲目。
APPCOMMAND_MEDIA_RECORD 48	开始录制当前流。

值	含义
APPCOMMAND_MEDIA_REWIND 50	以更高的速度在流中倒退。这可以通过多种方式实现，例如，使用固定速度或在一系列增加的速度之间切换。
APPCOMMAND_MEDIA_STOP 13	停止播放。
APPCOMMAND_MIC_ON_OFF_TOGGLE 44	切换麦克风。
APPCOMMAND_MICROPHONE_VOLUME_DOWN 25	降低麦克风音量。
APPCOMMAND_MICROPHONE_VOLUME_MUTE 24	将麦克风音量设为静音。
APPCOMMAND_MICROPHONE_VOLUME_UP 26	提高麦克风音量。
APPCOMMAND_NEW 29	创建新窗口。
APPCOMMAND_OPEN 30	打开窗口。
APPCOMMAND_PASTE 38	粘贴
APPCOMMAND_PRINT 33	打印当前文档。
APPCOMMAND_REDO 35	重做上一个操作。
APPCOMMAND_REPLY_TO_MAIL 39	答复邮件。
APPCOMMAND_SAVE 32	保存当前文档。
APPCOMMAND_SEND_MAIL 41	发送邮件。
APPCOMMAND SPELL_CHECK 42	启动拼写检查。
APPCOMMAND_TREBLE_DOWN 22	降低高音。

值	含义
APPCOMMAND_TREBLE_UP 23	提高高音。
APPCOMMAND_UNDO 34	撤消上一个操作。
APPCOMMAND_VOLUME_DOWN 9	降低音量。
APPCOMMAND_VOLUME_MUTE 8	设为静音。
APPCOMMAND_VOLUME_UP 10	提高静音。

uDevice 组件指示生成输入事件的输入设备，可以是以下任一值。

值	含义
FAPPCOMMAND_KEY 0	用户按下一个键。
FAPPCOMMAND_MOUSE 0x8000	用户单击了鼠标按钮。
FAPPCOMMAND_OEM 0x1000	未知的硬件源生成了事件。它可以是鼠标或键盘事件。

dwKeys 组件指示各种虚拟键是否按下，可以是以下一个或多个值。

值	含义
MK_CONTROL 0x0008	按下了 Ctrl 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 Shift 键。

值	含义
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

返回值

如果应用程序处理此消息，它应返回 TRUE。有关处理此返回值的详细信息，请参阅“注解”部分。

备注

[DefWindowProc](#) 在处理 [WM_XBUTTONUP](#) 或 [WM_NCXBUTTONUP](#) 消息时，或者当用户键入应用程序命令键时，会生成 WM_APPCOMMAND 消息。

如果子窗口不处理此消息，而是调用 [DefWindowProc](#)，则 DefWindowProc 会将消息发送到其父窗口。如果最上层窗口不处理此消息，而是调用 DefWindowProc，则 DefWindowProc 将调用挂钩代码等于 HSHELL_APPCOMMAND 的 shell 挂钩。

若要获取光标的坐标（如果消息是通过鼠标单击生成的），应用程序可以调用 [GetMessagePos](#)。应用程序可以通过检查 IParam 是否包含 FAPPCOMMAND_MOUSE 来测试消息是否由鼠标生成。

与其他窗口消息不同，如果应用程序处理此消息，则应从此消息返回 TRUE。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 [DefWindowProc](#) 来处理它。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[DefWindowProc](#)

[GET_APPCOMMAND_LPARAM](#)

[GET_DEVICE_LPARAM](#)

[GET_KEYSTATE_LPARAM](#)

[ShellProc](#)

[WM_XBUTTONUP](#)

[WM_NCXBUTTONUP](#)

概念性

[鼠标输入](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_CHAR消息

项目 • 2023/06/13

当 [TranslateMessage](#) 函数翻译WM_KEYDOWN消息时，使用键盘焦点发布到窗口。WM_CHAR消息包含按下的键的字符代码。

C++

```
#define WM_CHAR 0x0102
```

参数

wParam

键的字符代码。

lParam

重复计数、扫描代码、扩展键标志、上下文代码、以前的键状态标志和转换状态标志，如下表所示。

Bits	含义
0- 15	当前消息的重复计数。该值是由于用户按住键而自动重复击键的次数。如果击键的保持时间足够长，则发送多个消息。但是，重复计数不是累积的。
16- 23	扫描代码。该值取决于 OEM。
24	指示键是否为扩展键，例如显示在增强型 101 键或 102 键键盘上的右侧 Alt 和 Ctrl 键。如果是扩展键，则值为 1；否则为 0。
25- 28	保留;请勿使用。
29	上下文代码。如果在按住 ALT 键的同时按该键，则值为 1；否则值为 0。
30	上一个键状态。如果在发送消息之前键处于按下状态，则值为 1；如果键处于未按下状态，则值为 0。
31	转换状态。如果释放了键，则值为 1；如果按下了键，则值为 0。

有关详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

示例

C++

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        // ...

        case WM_CHAR:
            OnKeyPress(wParam);
            break;

        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;
}
```

GitHub 上的 [Windows 经典示例](#) 中的例子。

备注

如果使用 [RegisterClass](#) 函数的 Unicode 版本注册窗口类，则 **WM_CHAR** 消息在其 **wParam** 中使用 UTF-16 (16 位 Unicode 转换格式) 代码单元。否则，系统会在当前进程代码页中提供字符，可在 Windows 版本 1903 (2019 年 5 月更新) 及更新版本中将其设置为 UTF-8。有关详细信息，请参阅 [注册窗口类](#) 和 [在 Windows 应用中使用 UTF-8 代码页](#)。

从 Windows Vista 开始，**WM_CHAR** 消息可以将 [UTF-16 代理项对](#) 发送到 Unicode 窗口。如有必要，请使用 [IS_HIGH_SURROGATE](#)、[IS_LOW_SURROGATE](#) 和 [IS_SURROGATE_PAIR](#) 宏来检测此类情况。

按下的键和生成的字符消息之间不一定有一对一的对应关系，因此 *lParam* 参数的高阶字中的信息通常对应用程序没有用。高序字中的信息仅适用于发布 **WM_CHAR** 消息之前的最新 **WM_KEYDOWN** 消息。

对于增强型 101 键和 102 键键盘，扩展键是键盘 main 部分的右 Alt 键和右侧 CTRL 键；数字小键盘左侧群集中的 INS、DEL、HOME、END、PAGE UP、PAGE Down 和箭头键；数字小键盘中的除号 (/) 和 ENTER 键。其他一些键盘可能支持 *lParam* 参数中的扩展键位。

[WM_UNICHAR](#) 消息与 [WM_CHAR](#) 相同，只不过它使用 UTF-32。它旨在将 Unicode 字符发送或发布到 ANSI 窗口，并且可以处理 Unicode 补充平面字符。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

请参阅

- [TranslateMessage](#)
- [WM_KEYDOWN](#)
- [WM_UNICHAR](#)
- [键盘输入](#)
- [关于键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

WM_DEADCHAR消息

项目 • 2023/06/13

当 [TranslateMessage](#) 函数翻译WM_KEYUP消息时，将发布到具有键盘焦点的窗口。WM_DEADCHAR 指定由死键生成的字符代码。死键是生成字符的键，例如双点) 的 umlaut (，该字符与其他字符组合形成复合字符。例如，通过键入 umlaut 字符的死键，然后键入 O 键，生成 umlaut-O 字符 ()。

C++

<code>#define WM_DEADCHAR</code>	<code>0x0103</code>
----------------------------------	---------------------

参数

wParam

死键生成的字符代码。

lParam

重复计数、扫描代码、扩展键标志、上下文代码、以前的键状态标志和转换状态标志，如下表所示。

Bits	含义
0- 15	当前消息的重复计数。该值是由于用户按住键而自动重复击键的次数。如果击键的保持时间足够长，则发送多个消息。但是，重复计数不是累积的。
16- 23	扫描代码。该值取决于 OEM。
24	指示键是扩展键，例如在增强型 101 键或 102 键键盘上显示的右侧 Alt 键和 Ctrl 键。如果是扩展键，则值为 1；否则为 0。
25- 28	保留；请勿使用。
29	上下文代码。如果在按住 ALT 键的同时按该键，则值为 1；否则值为 0。
30	上一个键状态。如果在发送消息之前键处于按下状态，则值为 1；如果键处于未按下状态，则值为 0。
31	转换状态。如果释放了键，则值为 1；如果按下了键，则值为 0。

有关更多详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

备注

WM_DEADCHAR消息通常由应用程序用来向用户提供有关按下的每个键的反馈。例如，应用程序可以在当前字符位置显示音调，而不移动插入点。

由于按下的键和生成的字符消息之间不一定有一对一的对应关系，因此 *lParam* 参数的高字节中的信息通常对应用程序没有用。高字节中的信息仅适用于发布 WM_DEADCHAR 邮件之前的最新 WM_KEYDOWN 消息。

对于增强型 101 和 102 键键盘，扩展键是键盘 main 部分的右 ALT 和右 CTRL 键；数字键盘左侧群集中的 INS、DEL、HOME、END、PAGE UP、PAGE DOWN 和箭头键；数字键盘中的除号 (/) 和 Enter 键。其他一些键盘可能支持 *lParam* 参数中的扩展键位。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

请参阅

- [TranslateMessage](#)
- [WM_KEYDOWN](#)
- [WM_KEYUP](#)
- [WM_SYSDEADCHAR](#)
- [键盘输入](#)
- [关于键盘输入](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

WM_HOTKEY消息

项目 • 2023/06/13

当用户按下 [RegisterHotKey](#) 函数注册的热键时发布。 消息放置在与注册热密钥的线程关联的消息队列的顶部。

C++

```
#define WM_HOTKEY 0x0312
```

参数

wParam

生成消息的热键的标识符。 如果消息由系统定义的热键生成，则此参数将是以下值之一。

值	含义
IDHOT_SNAPDESKTOP -2	按下了“贴靠桌面”热键。
IDHOT_SNAPWINDOW -1	按下了“贴靠窗口”热键。

lParam

低序字指定要与由高序字指定的键组合按下的键，以生成 WM_HOTKEY 消息。 此单词可以是以下一个或多个值。 高序字指定热键的虚拟密钥代码。

值	含义
MOD_ALT 0x0001	其中一个 ALT 键已被按住。
MOD_CONTROL 0x0002	按住任一 CTRL 键。
MOD_SHIFT 0x0004	两个 SHIFT 键均已按下。
MOD_WIN 0x0008	其中一个 WINDOWS 密钥已被按住。 这些键标有 Windows 徽标。 涉及 Windows 密钥的热键保留供操作系统使用。

备注

WM_HOTKEY 与 [WM_GETHOTKEY](#) 和 [WM_SETHOTKEY](#) 热键无关。 WM_HOTKEY消息针对通用热键发送，而WM_SETHOTKEY和WM_GETHOTKEY消息与窗口激活热键相关。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[RegisterHotKey](#)

[WM_GETHOTKEY](#)

[WM_SETHOTKEY](#)

概念性

[键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

WM_KEYDOWN消息

项目 • 2023/06/13

按下非系统键时，使用键盘焦点发布到窗口。 非系统键是在未按下 Alt 键时按下的键。

C++

```
#define WM_KEYDOWN 0x0100
```

参数

wParam

非系统密钥的虚拟密钥代码。 请参阅 [虚拟密钥代码](#)。

lParam

重复计数、扫描代码、扩展键标志、上下文代码、以前的键状态标志和转换状态标志，如下所示。

Bits	含义
0-15	当前消息的重复计数。 该值是由于用户按住键而自动重复击键的次数。 如果击键的保持时间足够长，则发送多个消息。 但是，重复计数不是累积的。
16-23	扫描代码。 该值取决于 OEM。
24	指示键是扩展键，例如在增强型 101 键或 102 键键盘上显示的右侧 Alt 键和 Ctrl 键。 如果是扩展键，则值为 1；否则为 0。
25-28	保留；请勿使用。
29	上下文代码。 对于 WM_KEYDOWN 消息，该值始终为 0。
30	上一个键状态。 如果键在发送消息之前关闭，则值为 1；如果键已打开，则值为 0。
31	转换状态。 对于 WM_KEYDOWN 消息，该值始终为 0。

有关更多详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

示例

C++

```
LRESULT CALLBACK HostWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_KEYDOWN:
            if (wParam == VK_ESCAPE)
            {
                if (isFullScreen)
                {
                    GoPartialScreen();
                }
            }
            break;

        // ...
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

GitHub 上的 [Windows 经典示例](#) 中的例子。

备注

如果按下 F10 键，[DefWindowProc](#) 函数将设置内部标志。当 [DefWindowProc](#) 收到 [WM_KEYUP](#) 消息时，函数会检查内部标志是否已设置，如果是，则向顶级窗口发送 [WM_SYSCOMMAND](#) 消息。消息的 [WM_SYSCOMMAND](#) 参数设置为 [SC_KEYMENU](#)。

由于自动重现功能，在发布 [WM_KEYUP](#) 消息之前，可能会发布多个 [WM_KEYDOWN](#) 消息。可以使用上一个键状态 (位 30) 来确定 [WM_KEYDOWN](#) 消息是指示第一次向下转换还是重复向下转换。

对于增强型 101 和 102 键键盘，扩展键是键盘 main 部分中的右 ALT 和 CTRL 键；数字键盘左侧群集中的 INS、DEL、HOME、END、PAGE UP、PAGE DOWN 和箭头键；数字键盘中除号 (/) 和 Enter 键。其他键盘可能支持 *lParam* 参数中的扩展键位。

应用程序必须将 *wParam* 传递到 [TranslateMessage](#)，而无需对其进行更改。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

- [DefWindowProc](#)
- [TranslateMessage](#)
- [WM_CHAR](#)
- [WM_KEYUP](#)
- [WM_SYSCOMMAND](#)
- [键盘输入](#)
- [关于键盘输入](#)

反馈

此页面是否有帮助？



[在 Microsoft Q&A 获得帮助](#)

WM_KEYUP消息

项目 • 2023/06/12

释放非系统键时，使用键盘焦点发布到窗口。 非系统键是在未按下 Alt 键时按下的键，或在窗口具有键盘焦点时按下的键盘键。

C++

```
#define WM_KEYUP 0x0101
```

参数

wParam

非系统密钥的虚拟密钥代码。 请参阅 [虚拟密钥代码](#)。

lParam

重复计数、扫描代码、扩展键标志、上下文代码、以前的键状态标志和转换状态标志，如下表所示。

Bits	含义
0-15	当前消息的重复计数。 该值是由于用户按住键而自动重复击键的次数。 对于WM_KEYUP消息，重复计数始终为 1。
16-23	扫描代码。 该值取决于 OEM。
24	指示键是扩展键，例如在增强型 101 键或 102 键键盘上显示的右侧 Alt 键和 Ctrl 键。 如果是扩展键，则值为 1；否则为 0。
25-28	保留；请勿使用。
29	上下文代码。 对于 WM_KEYUP 消息，该值始终为 0。
30	上一个键状态。 对于 WM_KEYUP 消息，该值始终为 1。
31	转换状态。 对于 WM_KEYUP 消息，该值始终为 1。

有关更多详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

备注

如果释放了 F10 键或 ALT 键，[DefWindowProc](#) 函数会将 [WM_SYSCOMMAND](#) 消息发送到顶级窗口。消息的 *wParam* 参数设置为 SC_KEYMENU。

对于增强型 101 和 102 键键盘，扩展键是键盘 main 部分中的右 ALT 和 CTRL 键；数字键盘左侧群集中的 INS、DEL、HOME、END、PAGE UP、PAGE DOWN 和箭头键；数字键盘中除号 (/) 和 Enter 键。其他键盘可能支持 *lParam* 参数中的扩展键位。

应用程序必须将 *wParam* 传递到 [TranslateMessage](#)，而无需对其进行更改。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

- [DefWindowProc](#)
- [TranslateMessage](#)
- [WM_KEYDOWN](#)
- [WM_SYSCOMMAND](#)
- [键盘输入](#)
- [关于键盘输入](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

WM_KILLFOCUS消息

项目 • 2023/06/13

在失去键盘焦点之前立即发送到窗口。

C++

```
#define WM_KILLFOCUS 0x0008
```

参数

wParam

接收键盘焦点的窗口的句柄。 此参数可以为 NULL。

lParam

未使用此参数。

返回值

如果应用程序处理此消息，则应返回零。

备注

如果应用程序显示插入点，此时应销毁插入点。

处理此消息时，不要进行任何显示或激活窗口的函数调用。 这会导致线程产生控制，并可能导致应用程序停止响应消息。 有关详细信息，请参阅 [消息死锁](#)。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[SetFocus](#)

[WM_SETFOCUS](#)

概念性

[键盘输入](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_SETFOCUS消息

项目 • 2023/06/12

在获得键盘焦点后发送到窗口。

C++

```
#define WM_SETFOCUS 0x0007
```

参数

wParam

已失去键盘焦点的窗口的句柄。此参数可以为 NULL。

lParam

未使用此参数。

返回值

如果应用程序处理此消息，则应返回零。

备注

若要显示插入点，应用程序应在收到 WM_SETFOCUS 消息时调用相应的插入点函数。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

SetFocus

WM_KILLFOCUS

概念性

键盘输入

反馈

此页面是否有帮助?

是

否

在 Microsoft Q&A 获得帮助

WM_SYSDEADCHAR消息

项目 • 2023/06/13

当 [TranslateMessage](#) 函数翻译WM_SYSKEYDOWN消息时，使用键盘焦点发送到窗口。WM_SYSDEADCHAR 指定系统死键的字符代码，即按住 Alt 键时按下的死键。

C++

```
#define WM_SYSDEADCHAR 0x0107
```

参数

wParam

由系统死键生成的字符代码，即按住 Alt 键时按下的死键。

lParam

重复计数、扫描代码、扩展键标志、上下文代码、以前的键状态标志和转换状态标志，如下表所示。

Bits	含义
0- 15	当前消息的重复计数。该值是由于用户按住键而自动重复击键的次数。如果击键的保持时间足够长，则发送多个消息。但是，重复计数不是累积的。
16- 23	扫描代码。该值取决于 OEM。
24	指示键是扩展键，例如在增强型 101 键或 102 键键盘上显示的右侧 Alt 键和 Ctrl 键。如果是扩展键，则值为 1；否则为 0。
25- 28	保留；请勿使用。
29	上下文代码。如果在按住 ALT 键的同时按该键，则值为 1；否则值为 0。
30	上一个键状态。如果在发送消息之前键处于按下状态，则值为 1；如果键处于未按下状态，则值为 0。
31	转换状态。如果释放了键，则值为 1；如果按下了键，则值为 0。

有关更多详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

备注

对于增强型 101 和 102 键键盘，扩展键是键盘 main 部分中的右 ALT 和 CTRL 键；数字键盘左侧群集中的 INS、DEL、HOME、END、PAGE UP、PAGE DOWN 和箭头键；数字键盘中除号 (/) 和 Enter 键。其他键盘可能支持 *lParam* 参数中的扩展键位。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

- [TranslateMessage](#)
- [WM_DEADCHAR](#)
- [WM_SYSKEYDOWN](#)
- [键盘输入](#)
- [关于键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

WM_SYSKEYDOWN message

Article • 08/04/2022

Posted to the window with the keyboard focus when the user presses the F10 key (which activates the menu bar) or holds down the ALT key and then presses another key. It also occurs when no window currently has the keyboard focus; in this case, the WM_SYSKEYDOWN message is sent to the active window. The window that receives the message can distinguish between these two contexts by checking the context code in the *lParam* parameter.

C++

```
#define WM_SYSKEYDOWN 0x0104
```

Parameters

wParam

The virtual-key code of the key being pressed. See [Virtual-Key Codes](#).

lParam

The repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, as shown in the following table.

Bits	Meaning
0-15	The repeat count for the current message. The value is the number of times the keystroke is autorepeated as a result of the user holding down the key. If the keystroke is held long enough, multiple messages are sent. However, the repeat count is not cumulative.
16-23	The scan code. The value depends on the OEM.
24	Indicates whether the key is an extended key, such as the right-hand ALT and CTRL keys that appear on an enhanced 101- or 102-key keyboard. The value is 1 if it is an extended key; otherwise, it is 0.
25-28	Reserved; do not use.
29	The context code. The value is 1 if the ALT key is down while the key is pressed; it is 0 if the WM_SYSKEYDOWN message is posted to the active window because no window has the keyboard focus.

Bits	Meaning
30	The previous key state. The value is 1 if the key is down before the message is sent, or it is 0 if the key is up.
31	The transition state. The value is always 0 for a WM_SYSKEYDOWN message.

For more detail, see [Keystroke Message Flags](#).

Return value

An application should return zero if it processes this message.

Remarks

The [DefWindowProc](#) function examines the specified key and generates a **WM_SYSCOMMAND** message if the key is either TAB or ENTER.

When the context code is zero, the message can be passed to the [TranslateAccelerator](#) function, which will handle it as though it were a normal key message instead of a character-key message. This allows accelerator keys to be used with the active window even if the active window does not have the keyboard focus.

Because of automatic repeat, more than one **WM_SYSKEYDOWN** message may occur before a **WM_SYSKEYUP** message is sent. The previous key state (bit 30) can be used to determine whether the **WM_SYSKEYDOWN** message indicates the first down transition or a repeated down transition.

For enhanced 101- and 102-key keyboards, enhanced keys are the right ALT and CTRL keys on the main section of the keyboard; the INS, DEL, HOME, END, PAGE UP, PAGE DOWN, and arrow keys in the clusters to the left of the numeric keypad; and the divide (/) and ENTER keys in the numeric keypad. Other keyboards may support the extended-key bit in the *lParam* parameter.

This message is also sent whenever the user presses the F10 key without the ALT key.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	Winuser.h (include Windows.h)

See also

- [DefWindowProc](#)
- [TranslateAccelerator](#)
- [WM_SYSCOMMAND](#)
- [WM_SYSKEYUP](#)
- [Keyboard Input](#)
- [About Keyboard Input](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_SYSKEYUP消息

项目 • 2023/06/22

当用户释放按住 Alt 键时按下的键时，使用键盘焦点发布到窗口。当当前没有窗口具有键盘焦点时，也会发生这种情况;在这种情况下，WM_SYSKEYUP 消息将发送到活动窗口。接收消息的窗口可以通过检查 IParam 中的上下文代码来区分这两个上下文。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_SYSKEYUP 0x0105
```

参数

wParam

要释放的密钥的虚拟密钥代码。请参阅[虚拟键代码](#)。

lParam

重复次数、扫描代码、扩展键标志、上下文代码、上一个键状态标志和转换状态标志，如下表所示。

Bits	含义
0- 15	当前消息的重复次数。该值是由于用户按住键而自动重复击键的次数。对于 WM_SYSKEYUP 消息，重复计数始终为 1。
16- 23	扫描代码。此值取决于 OEM。
24	指定键是否为扩展键，例如出现在增强型 101 或 102 键键盘上的右侧 ALT 和 CTRL 键。如果它是扩展键，则值为 1;否则为零。
25- 28	保留；请勿使用。
29	上下文代码。如果在释放键时 ALT 键关闭，则值为 1;如果 WM_SYSKEYUP 消息发布到活动窗口，则为零，因为没有窗口具有键盘焦点。
30	上一个键状态。对于 WM_SYSKEYUP 消息，该值始终为 1。
31	转换状态。对于 WM_SYSKEYUP 消息，该值始终为 1。

有关详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

备注

如果释放了 F10 键或 ALT 键，[DefWindowProc](#) 函数会将 [WM_SYSCOMMAND](#) 消息发送到顶级窗口。消息的 *wParam* 参数设置为 SC_KEYMENU。

当上下文代码为零时，可以将消息传递给 [TranslateAccelerator](#) 函数，该函数将其视为普通的键消息进行处理，而不是将其视为字符键消息。这样，即使活动窗口没有键盘焦点，也能对活动窗口使用加速键。

对于增强型 101 和 102 键键盘，扩展键是键盘主区域上的右侧 ALT 和 CTRL 键；数字键盘左侧键群中的 INS、DEL、HOME、END、PAGE UP、PAGE DOWN 和箭头键；以及数字键盘中的除号 (/) 和 ENTER 键。其他一些键盘可能支持 IParam 参数中的扩展键位。

对于非美国增强型 102 键键盘，右侧 ALT 键作为 CTRL+Alt 键处理。下表显示了当用户按下并释放此键时生成的消息序列。

Message	虚拟密钥代码
WM_KEYDOWN	VK_CONTROL
WM_KEYDOWN	VK_MENU
WM_KEYUP	VK_CONTROL
WM_SYSKEYUP	VK_MENU

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

请参阅

- [DefWindowProc](#)
 - [TranslateAccelerator](#)
 - [WM_SYSCOMMAND](#)
 - [WM_SYSKEYDOWN](#)
 - [键盘输入](#)
 - [关于键盘输入](#)
-

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_UNICHAR消息

项目 • 2023/06/13

应用程序可以使用WM_UNICHAR消息将输入发布到其他窗口。此消息包含按下的键的字符代码。(通过发送 *wParam* 设置为 UNICODE_NOCHAR.) 的消息来测试目标应用是否可以处理WM_UNICHAR消息

C++

```
#define WM_UNICHAR 0x0109
```

参数

wParam

键的字符代码。

如果 *wParam* UNICODE_NOCHAR 并且应用程序处理此消息，则返回 TRUE。

[DefWindowProc](#) 函数将在默认) (返回 FALSE。

如果未UNICODE_NOCHAR*wParam*，则返回 FALSE。 Unicode [DefWindowProc](#) 发布具有相同参数 的WM_CHAR 消息，ANSI DefWindowProc 函数发布一条 或两条 具有相应 ANSI 字符的WM_CHAR消息 ()。

lParam

重复计数、扫描代码、扩展键标志、上下文代码、以前的键状态标志和转换状态标志，如下表所示。

Bits	含义
0- 15	当前消息的重复计数。该值是由于用户按住键而自动重复击键的次数。如果击键的保持时间足够长，则发送多个消息。但是，重复计数不是累积的。
16- 23	扫描代码。该值取决于 OEM。
24	指示键是否为扩展键，例如显示在增强型 101 键或 102 键键盘上的右侧 Alt 和 Ctrl 键。如果是扩展键，则值为 1；否则为 0。
25- 28	保留;请勿使用。
29	上下文代码。如果在按住 ALT 键的同时按该键，则值为 1；否则值为 0。

Bits	含义
30	上一个键状态。如果在发送消息之前键处于按下状态，则值为 1；如果键处于未按下状态，则值为 0。
31	转换状态。如果释放了键，则值为 1；如果按下了键，则值为 0。

有关详细信息，请参阅 [击键消息标志](#)。

返回值

如果应用程序处理此消息，则应返回零。

备注

WM_UNICHAR消息类似于[WM_CHAR](#)，但它使用 Unicode 转换格式 (UTF) -32，而 **WM_CHAR** 使用 UTF-16。

此消息旨在将 Unicode 字符发送或发布到 ANSI 窗口，并且可以处理 Unicode 补充平面字符。

由于按下的键与生成的字符消息之间不一定有一对一的对应关系，因此 *IParam* 参数的高序字中的信息通常对应用程序没有用处。高序字中的信息仅适用于发布 **WM_UNICHAR** 消息之前的最新 **WM_KEYDOWN** 消息。

对于增强型 101 键和 102 键键盘，扩展键是键盘 main 部分的右 Alt 键和右侧 CTRL 键；数字小键盘左侧群集中的 INS、DEL、HOME、END、PAGE UP、PAGE Down 和箭头键；数字小键盘中的除号 (/) 和 ENTER 键。其他一些键盘可能支持 *IParam* 参数中的扩展键位。

要求

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

- [TranslateMessage](#)

- WM_CHAR
 - WM_KEYDOWN
 - 键盘输入
 - 关于键盘输入
-

反馈

此页面是否有帮助?



[在 Microsoft Q&A 获得帮助](#)

键盘输入结构

项目 · 2023/06/12

本节内容

- [HARDWAREINPUT](#)
 - [输入](#)
 - [KEYBDINPUT](#)
 - [LASTINPUTINFO](#)
 - [MOUSEINPUT](#)
-

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

HARDWAREINPUT 结构 (winuser.h)

项目2023/08/28

包含有关由键盘或鼠标以外的输入设备生成的模拟消息的信息。

语法

C++

```
typedef struct tagHARDWAREINPUT {
    DWORD uMsg;
    WORD wParamL;
    WORD wParamH;
} HARDWAREINPUT, *PHARDWAREINPUT, *LPHARDWAREINPUT;
```

成员

uMsg

类型: DWORD

输入硬件生成的消息。

wParamL

类型: WORD

uMsg 的 *lParam* 参数的低序字。

wParamH

类型: WORD

uMsg 的 *lParam* 参数的高序字。

要求

最低受支持的客户端

Windows XP [仅限桌面应用]

最低受支持的服务器

Windows Server 2003 [仅限桌面应用]

标头

winuser.h (包括 Windows.h)

请参阅

概念性

[INPUT](#)

[键盘输入](#)

引用

[SendInput](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

输入结构 (winuser.h)

项目2024/03/04

由 [SendInput](#) 用于存储用于合成输入事件（例如击键、鼠标移动和鼠标单击）的信息。

语法

C++

```
typedef struct tagINPUT {
    DWORD type;
    union {
        MOUSEINPUT     mi;
        KEYBDINPUT    ki;
        HARDWAREINPUT hi;
    } DUMMYUNIONNAME;
} INPUT, *PINPUT, *LPIINPUT;
```

成员

type

类型: DWORD

输入事件的类型。此成员可以是以下值之一。

[展开表](#)

值	含义
INPUT_MOUSE 0	事件是鼠标事件。使用联合的 mi 结构。
INPUT_KEYBOARD 1	事件是键盘事件。使用联合的 ki 结构。
INPUT_HARDWARE 2	事件是硬件事件。使用联合的 hi 结构。

DUMMYUNIONNAME

DUMMYUNIONNAME.mi

类型: [MOUSEINPUT](#)

有关模拟鼠标事件的信息。

DUMMYUNIONNAME.ki

类型: [KEYBDINPUT](#)

有关模拟键盘事件的信息。

DUMMYUNIONNAME.hi

类型: [HARDWAREINPUT](#)

有关模拟硬件事件的信息。

注解

`INPUT_KEYBOARD` 支持非键板输入方法，例如手写识别或语音识别，就像它是使用 `KEYEVENTF_UNICODE` 标志输入文本一样。有关详细信息，请参阅 [KEYBDINPUT](#) 的备注部分。

要求

[] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetMessageExtraInfo](#)

[HARDWAREINPUT](#)

[KEYBDINPUT](#)

[键盘输入](#)

[MOUSEINPUT](#)

引用

[SendInput](#)

[keybd_event](#)

[mouse_event](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

KEYBDINPUT 结构 (winuser.h)

项目2023/05/20

包含有关模拟键盘事件的信息。

语法

C++

```
typedef struct tagKEYBDINPUT {
    WORD      wVk;
    WORD      wScan;
    DWORD     dwFlags;
    DWORD     time;
    ULONG_PTR dwExtraInfo;
} KEYBDINPUT, *PKEYBDINPUT, *LPKEYBDINPUT;
```

成员

wVk

类型: WORD

虚拟密钥代码。代码必须是 1 到 254 范围内的值。如果 dwFlags 成员指定 KEYEVENTF_UNICODE，则 wVk 必须为 0。

wScan

类型: WORD

密钥的硬件扫描代码。如果 dwFlags 指定 KEYEVENTF_UNICODE，则 wScan 将指定一个 Unicode 字符，该字符将发送到前台应用程序。

dwFlags

类型: DWORD

指定击键的各个方面。此成员可以是以下值的某些组合。

值	含义
KEYEVENTF_EXTENDEDKEY 0x0001	如果指定，则 wScan 扫描代码由两个字节组成的序列组成，其中第一个字节的值为 0xE0。有关详细信息，请参阅 扩展键标志 。

KEYEVENTF_KEYUP 0x0002	如果指定，则释放密钥。如果未指定，则按键。
KEYEVENTF_SCANCODE 0x0008	如果指定，wScan 将标识密钥，并忽略 wVk。
KEYEVENTF_UNICODE 0x0004	如果指定，系统会合成 VK_PACKET 击键。wVk 参数必须为零。此标志只能与 KEYEVENTF_KEYUP 标志组合使用。有关详细信息，请参见“备注”部分。

time

类型: DWORD

事件的时间戳（以毫秒为单位）。如果此参数为零，则系统将提供自己的时间戳。

dwExtraInfo

类型: ULONG_PTR

与击键关联的附加值。使用 [GetMessageExtraInfo](#) 函数获取此信息。

注解

INPUT_KEYBOARD 支持非键板输入方法（如手写识别或语音识别），就像它是使用 KEYEVENTF_UNICODE 标志输入文本一样。如果指定了 KEYEVENTF_UNICODE，SendInput 会将 WM_KEYDOWN 或 WM_KEYUP 消息发送到 wParam 等于 VK_PACKET 的前景线程的消息队列。GetMessage 或 PeekMessage 获取此消息后，将消息传递给 TranslateMessage 会发布 WM_CHAR 消息，其中包含 wScan 最初指定的 Unicode 字符。如果此 Unicode 字符被发布到 ANSI 窗口，则会自动转换为相应的 ANSI 值。

设置 KEYEVENTF_SCANCODE 标志以根据扫描代码定义键盘输入。无论当前使用的是哪种键盘，这都可用于模拟物理击键。如果扫描代码是扩展密钥，还可以传递 KEYEVENTF_EXTENDEDKEY 标志。键的虚拟键值可能会根据当前键盘布局或按下的其他键而更改，但扫描代码将始终相同。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

另请参阅

- [GetMessageExtraInfo](#)
- [INPUT](#)
- [SendInput](#)
- [键盘输入](#)

LASTINPUTINFO 结构 (winuser.h)

项目2023/08/28

包含最后一个输入的时间。

语法

C++

```
typedef struct tagLASTINPUTINFO {
    UINT cbSize;
    DWORD dwTime;
} LASTINPUTINFO, *PLASTINPUTINFO;
```

成员

`cbSize`

类型: `UINT`

结构大小 (以字节为单位)。此成员必须设置为 `sizeof(LASTINPUTINFO)`。

`dwTime`

类型: `DWORD`

收到最后一个输入事件时的时钟周期计数。

注解

此函数可用于输入空闲检测。有关时钟周期计数的详细信息，请参阅 [GetTickCount](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetLastInputInfo](#)

[GetTickCount](#)

[键盘输入](#)

引用

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

MOUSEINPUT 结构 (winuser.h)

项目2023/08/28

包含有关模拟鼠标事件的信息。

语法

C++

```
typedef struct tagMOUSEINPUT {
    LONG      dx;
    LONG      dy;
    DWORD     mouseData;
    DWORD     dwFlags;
    DWORD     time;
    ULONG_PTR dwExtraInfo;
} MOUSEINPUT, *PMOUSEINPUT, *LPMOUSEINPUT;
```

成员

dx

类型: LONG

鼠标的绝对位置, 或自上次生成鼠标事件以来的运动量, 具体取决于 dwFlags 成员的值。 绝对数据指定为鼠标的 x 坐标; 相对数据指定为移动的像素数。

dy

类型: LONG

鼠标的绝对位置, 或自上次生成鼠标事件以来的运动量, 具体取决于 dwFlags 成员的值。 绝对数据指定为鼠标的 y 坐标; 相对数据指定为移动的像素数。

mouseData

类型: DWORD

如果 dwFlags 包含 MOUSEEVENTF_WHEEL, 则 mouseData 指定滚轮移动量。 正值表示滚轮向前旋转 (远离用户); 负值表示滚轮向后旋转 (朝向用户)。 一键滚轮定义为 WHEEL_DELTA, 即 120。

Windows Vista: 如果 *dwFlags* 包含 **MOUSEEVENTF_HWHEEL**，则 *dwData* 指定滚轮移动量。正值表示滚轮向右旋转；负值表示滚轮向左旋转。一键滚轮定义为 **WHEEL_DELTA**，即 120。

如果 *dwFlags* 不包含 **MOUSEEVENTF_WHEEL**、**MOUSEEVENTF_XDOWN** 或 **MOUSEEVENTF_XUP**，则 *mouseData* 应为零。

如果 *dwFlags* 包含 **MOUSEEVENTF_XDOWN** 或 **MOUSEEVENTF_XUP**，则 *mouseData* 指定按下或释放的 X 按钮。此值可以是以下标志的任意组合。

Value	含义
XBUTTON1 0x0001	设置是否按下或释放第一个 X 按钮。
XBUTTON2 0x0002	设置是否按下或释放第二个 X 按钮。

dwFlags

类型: **DWORD**

一组位标志，用于指定鼠标运动和按钮单击的各个方面。此成员中的位可以是以下值的任意合理组合。

指定鼠标按钮状态的位标志设置为指示状态的更改，而不是正在进行的条件。例如，如果按下并按住鼠标左键，则会在首次按下左键时设置 **MOUSEEVENTF_LEFTDOWN**，但不设置为后续动作。同样，**仅在** 首次释放按钮时设置**MOUSEEVENTF_LEFTUP**。

不能在 *dwFlags* 参数中同时指定 **MOUSEEVENTF_WHEEL** 标志和 **MOUSEEVENTF_XDOWN** 或 **MOUSEEVENTF_XUP** 标志，因为它们都需要使用 *mouseData* 字段。

Value	含义
MOUSEEVENTF_MOVE 0x0001	发生了移动。
MOUSEEVENTF_LEFTDOWN 0x0002	按下了左侧按钮。
MOUSEEVENTF_LEFTUP 0x0004	左按钮已释放。
MOUSEEVENTF_RIGHTDOWN 0x0008	按下了向右按钮。

Value	含义
MOUSEEVENTF_RIGHTUP 0x0010	右侧按钮已松开。
MOUSEEVENTF_MIDDLEDOWN 0x0020	按下中间按钮。
MOUSEEVENTF_MIDDLEUP 0x0040	中间按钮已释放。
MOUSEEVENTF_XDOWN 0x0080	按下了 X 按钮。
MOUSEEVENTF_XUP 0x0100	已释放 X 按钮。
MOUSEEVENTF_WHEEL 0x0800	如果鼠标有滚轮，则滚轮已移动。 移动量在 mouseData 中指定。
MOUSEEVENTF_HWHEEL 0x1000	如果鼠标有滚轮，则方向盘是水平移动的。 移动量在 mouseData 中指定。 Windows XP/2000： 不支持此值。
MOUSEEVENTF_MOVE_NOCOALESCE 0x2000	不会合并 WM_MOUSEMOVE 消息。 默认行为是 合并 WM_MOUSEMOVE 消息。 Windows XP/2000： 不支持此值。
MOUSEEVENTF_VIRTUALDESK 0x4000	将坐标映射到整个桌面。 必须与 MOUSEEVENTF_ABSOLUTE 一起使用。
MOUSEEVENTF_ABSOLUTE 0x8000	dx 和 dy 成员包含规范化的绝对坐标。 如果未设置标志， dx 和 dy 包含相对数据 (自上次报告的位置) 更改。 无论哪种类型的鼠标或其他指针设备 (如果有) 连接到系统，都可以设置或不设置此标志。 有关相对鼠标运动的详细信息，请参阅以下“备注”部分。

time

类型: DWORD

事件的时间戳 (以毫秒为单位)。 如果此参数为 0，则系统将提供自己的时间戳。

dwExtraInfo

类型: ULONG_PTR

与鼠标事件关联的附加值。 应用程序调用 [GetMessageExtraInfo](#) 以获取此额外信息。

备注

如果鼠标已移动（由`MOUSEEVENTF_MOVE`指示），`dx` 和 `dy` 将指定有关该移动的信息。信息指定为绝对或相对整数值。

如果指定了`MOUSEEVENTF_ABSOLUTE`值，则`dx` 和 `dy` 包含介于 0 和 65,535 之间的规范化绝对坐标。事件过程将这些坐标映射到显示图面。坐标 (0, 0) 映射到显示图面的左上角;坐标 (65535, 65535) 映射到右下角。在多监视器系统中，坐标映射到主监视器。

如果指定了`MOUSEEVENTF_VIRTUALDESK`，则坐标将映射到整个虚拟桌面。

如果未指定`MOUSEEVENTF_ABSOLUTE`值，`dx` 和 `dy` 指定相对于上一个鼠标事件（上一个报告位置）的移动。正值表示鼠标向右移动（或向下移动）;负值表示鼠标向左移动（或向上移动）。

相对鼠标运动受鼠标速度和双鼠标阈值的影响。用户使用控制面板的“**鼠标属性**”工作表的“**指针速度**”滑块设置这三个值。可以使用 [SystemParametersInfo](#) 函数获取和设置这些值。

系统会对指定的相对鼠标移动应用两个测试。如果沿 x 或 y 轴的指定距离大于第一个鼠标阈值，并且鼠标速度不为零，则系统会将距离加倍。如果沿 x 或 y 轴的指定距离大于第二个鼠标阈值，并且鼠标速度等于 2，系统会将应用第一个阈值测试得出的距离加倍。因此，系统可以沿 x 或 y 轴将指定的相对鼠标移动乘以最多四倍。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetMessageExtraInfo](#)

[INPUT](#)

[键盘输入](#)

引用

[SendInput](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

键盘输入常量

项目 · 2024/02/24

- 虚拟键代码

反馈

此页面是否有帮助?

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

虚拟键代码

项目 · 2023/09/22

下表显示了系统使用的虚拟键代码的符号常量名称、十六进制值和鼠标或键盘等效项。代码按数字顺序列出。

常数	Value	说明
VK_LBUTTON	0x01	鼠标左键
VK_RBUTTON	0x02	鼠标右键
VK_CANCEL	0x03	控制中断处理
VK_MBUTTON	0x04	鼠标中键
VK_XBUTTON1	0x05	X1 鼠标按钮
VK_XBUTTON2	0x06	X2 鼠标按钮
-	0x07	保留
VK_BACK	0x08	BACKSPACE 键
VK_TAB	0x09	Tab 键
-	0x0A-0B	预留
VK_CLEAR	0x0C	CLEAR 键
VK_RETURN	0x0D	Enter 键
-	0x0E-0F	未分配
VK_SHIFT	0x10	SHIFT 键
VK_CONTROL	0x11	CTRL 键
VK_MENU	0x12	Alt 键
VK_PAUSE	0x13	PAUSE 键
VK_CAPITAL	0x14	CAPS LOCK 键
VK_KANA	0x15	IME Kana 模式
VK_HANGUL	0x15	IME Hanguel 模式

常数	Value	说明
VKIME_ON	0x16	IME 打开
VK_JUNJA	0x17	IME Junja 模式
VKFINAL	0x18	IME 最终模式
VKHANJA	0x19	IME Hanja 模式
VKKANJI	0x19	IME Kanji 模式
VKIME_OFF	0x1A	IME 关闭
VK_ESCAPE	0x1B	ESC 键
VK_CONVERT	0x1C	IME 转换
VK_NONCONVERT	0x1D	IME 不转换
VK_ACCEPT	0x1E	IME 接受
VK_MODECHANGE	0x1F	IME 模式更改请求
VK_SPACE	0x20	空格键
VKPRIOR	0x21	PAGE UP 键
VKNEXT	0x22	PAGE DOWN 键
VKEND	0x23	END 键
VKHOME	0x24	HOME 键
VKLEFT	0x25	LEFT ARROW 键
VKUP	0x26	UP ARROW 键
VKRIGHT	0x27	RIGHT ARROW 键
VKDOWN	0x28	DOWN ARROW 键
VKSELECT	0x29	SELECT 键
VKPRINT	0x2A	PRINT 键
VKEXECUTE	0x2B	EXECUTE 键
VK_SNAPSHOT	0x2C	PRINT SCREEN 键
VK_INSERT	0x2D	INS 键
VK_DELETE	0x2E	DEL 键

常数	Value	说明
VK_HELP	0x2F	HELP 键
	0x30	0 键
	0x31	1 个键
	0x32	2 键
	0x33	3 键
	0x34	4 键
	0x35	5 键
	0x36	6 键
	0x37	7 键
	0x38	8 键
	0x39	9 键
-	0x3A- 40	Undefined
	0x41	A 键
	0x42	B 键
	0x43	C 键
	0x44	D 键
	0x45	E 键
	0x46	F 键
	0x47	G 键
	0x48	H 键
	0x49	I 键
	0x4A	J 键
	0x4B	K 键
	0x4C	L 键
	0x4D	M 键
	0x4E	N 键

常数	Value	说明
	0x4F	O 键
	0x50	P 键
	0x51	Q 键
	0x52	R 键
	0x53	S 键
	0x54	T 键
	0x55	U 键
	0x56	V 键
	0x57	W 键
	0x58	X 键
	0x59	Y 键
	0x5A	Z 键
VK_LWIN	0x5B	左 Windows 键
VK_RWIN	0x5C	右侧 Windows 键
VK_APPS	0x5D	应用程序密钥
-	0x5E	预留
VK_SLEEP	0x5F	计算机休眠键
VK_NUMPAD0	0x60	数字键盘 0 键
VK_NUMPAD1	0x61	数字键盘 1 键
VK_NUMPAD2	0x62	数字键盘 2 键
VK_NUMPAD3	0x63	数字键盘 3 键
VK_NUMPAD4	0x64	数字键盘 4 键
VK_NUMPAD5	0x65	数字键盘 5 键
VK_NUMPAD6	0x66	数字键盘 6 键
VK_NUMPAD7	0x67	数字键盘 7 键
VK_NUMPAD8	0x68	数字键盘 8 键

常数	Value	说明
VK_NUMPAD9	0x69	数字键盘 9 键
VK_MULTIPLY	0x6A	乘号键
VK_ADD	0x6B	加号键
VK_SEPARATOR	0x6C	分隔符键
VK_SUBTRACT	0x6D	减号键
VK_DECIMAL	0x6E	句点键
VK_DIVIDE	0x6F	除号键
VK_F1	0x70	F1 键
VK_F2	0x71	F2 键
VK_F3	0x72	F3 键
VK_F4	0x73	F4 键
VK_F5	0x74	F5 键
VK_F6	0x75	F6 键
VK_F7	0x76	F7 键
VK_F8	0x77	F8 键
VK_F9	0x78	F9 键
VK_F10	0x79	F10 键
VK_F11	0x7A	F11 键
VK_F12	0x7B	F12 键
VK_F13	0x7C	F13 键
VK_F14	0x7D	F14 键
VK_F15	0x7E	F15 键
VK_F16	0x7F	F16 键
VK_F17	0x80	F17 键
VK_F18	0x81	F18 键
VK_F19	0x82	F19 键

常数	Value	说明
VK_F20	0x83	F20 键
VK_F21	0x84	F21 键
VK_F22	0x85	F22 键
VK_F23	0x86	F23 键
VK_F24	0x87	F24 键
-	0x88-8F	保留
VK_NUMLOCK	0x90	NUM LOCK 键
VK_SCROLL	0x91	SCROLL LOCK 键
-	0x92-96	OEM 特有
-	0x97-9F	未分配
VK_LSHIFT	0xA0	左 SHIFT 键
VK_RSHIFT	0xA1	右 SHIFT 键
VK_LCONTROL	0xA2	左 Ctrl 键
VK_RCONTROL	0xA3	右 Ctrl 键
VK_LMENU	0xA4	左 ALT 键
VK_RMENU	0xA5	右 ALT 键
VK_BROWSER_BACK	0xA6	浏览器后退键
VK_BROWSER_FORWARD	0xA7	浏览器前进键
VK_BROWSER_REFRESH	0xA8	浏览器刷新键
VK_BROWSER_STOP	0xA9	浏览器停止键
VK_BROWSER_SEARCH	0xAA	浏览器搜索键
VK_BROWSER_FAVORITES	0xAB	浏览器收藏键
VK_BROWSER_HOME	0xAC	浏览器“开始”和“主页”键
VK_VOLUME_MUTE	0xAD	静音键

常数	Value	说明
VK_VOLUME_DOWN	0xAE	音量减小键
VK_VOLUME_UP	0xAF	音量增加键
VK_MEDIA_NEXT_TRACK	0xB0	下一曲目键
VK_MEDIA_PREV_TRACK	0xB1	上一曲目键
VK_MEDIA_STOP	0xB2	停止媒体键
VK_MEDIA_PLAY_PAUSE	0xB3	播放/暂停媒体键
VK_LAUNCH_MAIL	0xB4	启动邮件键
VK_LAUNCH_MEDIA_SELECT	0xB5	选择媒体键
VK_LAUNCH_APP1	0xB6	启动应用程序 1 键
VK_LAUNCH_APP2	0xB7	启动应用程序 2 键
-	0xB8- B9	预留
VK_OEM_1	0xBA	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 ;
VK_OEM_PLUS	0xBB	对于任何国家/地区，键 +
VK_OEM_COMMMA	0xBC	对于任何国家/地区，键 ,
VK_OEM_MINUS	0xBD	对于任何国家/地区，键 -
VK_OEM_PERIOD	0xBE	对于任何国家/地区，键 .
VK_OEM_2	0xBF	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 /?
VK_OEM_3	0xC0	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 `~
-	0xC1- DA	保留
VK_OEM_4	0xDB	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 [{
VK_OEM_5	0xDC	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 \\

常数	Value	说明
VK_OEM_6	0xDD	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 [] }
VK_OEM_7	0xDE	用于杂项字符；它可能因键盘而异。对于美国标准键盘，键 ' "
VK_OEM_8	0xDF	用于杂项字符；它可能因键盘而异。
-	0xE0	预留
-	0xE1	OEM 特有
VK_OEM_102	0xE2	美国标准键盘上的 <> 键，或非美国 102 键键盘上的 \\ 键
-	0xE3-E4	OEM 特有
VK_PROCESSKEY	0xE5	IME PROCESS 键
-	0xE6	OEM 特有
VK_PACKET	0xE7	用于将 Unicode 字符当作键击传递。 <code>VK_PACKET</code> 键是用于非键盘输入法的 32 位虚拟键值的低位字。有关更多信息，请参阅 KEYBDINPUT 、 SendInput 、 WM_KEYDOWN 和 WM_KEYUP 中的注释
-	0xE8	未分配
-	0xE9-F5	OEM 特有
VK_ATTN	0xF6	Attn 键
VK_CRSEL	0xF7	CrSel 键
VK_EXSEL	0xF8	ExSel 键
VK_EREOF	0xF9	Erase EOF 键
VK_PLAY	0xFA	Play 键
VK_ZOOM	0xFB	Zoom 键
VK_NONAME	0xFC	预留
VK_PA1	0xFD	PA1 键
VK_OEM_CLEAR	0xFE	Clear 键

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h

反馈

此页面是否有帮助？

是

否

鼠标输入

项目 • 2023/06/12

本部分介绍系统如何向应用程序提供鼠标输入，以及应用程序如何接收和处理该输入。

在本节中

主题	说明
关于鼠标输入	本主题讨论鼠标输入。
使用鼠标输入	本部分介绍与鼠标输入关联的任务。
鼠标输入参考	

函数

名称	说明
_TrackMouseEvent	当鼠标指针离开窗口或将鼠标悬停在窗口上指定时间量时发布消息。此函数调用 TrackMouseEvent (如果存在)，否则会模拟它。
BlockInput	阻止键盘和鼠标输入事件到达应用程序。
DragDetect	捕获鼠标并跟踪其移动，直到用户释放左键、按 ESC 键或将鼠标移动到围绕指定点的拖动矩形外部。拖动矩形的宽度和高度由 getSystemMetrics 函数返回的SM_CXDRAG和SM_CYDRAG值指定。
EnableMouseInPointer	使鼠标充当指针设备。
EnableWindow	启用或禁用对指定窗口或控件的鼠标和键盘输入。禁用输入时，窗口不会接收鼠标单击和按键等输入。启用输入后，窗口将接收所有输入。
GetCapture	检索窗口的句柄 ((如果有任何捕获了鼠标的))。一次只有一个窗口可以捕获鼠标;无论光标是否在其边框内，此窗口都会收到鼠标输入。
GetDoubleClickTime	检索鼠标的当前双击时间。双击是鼠标按钮的一系列两次单击，第二次单击发生在第一次之后的指定时间内。双击时间是双击的第一次和第二次单击之间可能发生的最大毫秒数。
GetMouseMovePointsEx	检索最多 64 个以前鼠标或笔坐标的记录。
IsWindowEnabled	确定是否为鼠标和键盘输入启用指定的窗口。

名称	说明
ReleaseCapture	从当前线程的窗口中释放鼠标捕获，并还原正常的鼠标输入处理。捕获鼠标的窗口接收所有鼠标输入，而不考虑光标的位置，但光标热点位于另一个线程的窗口中时单击鼠标按钮除外。
SendInput	合成击键、鼠标动作和按钮单击。
SetCapture	将鼠标捕获设置为属于当前线程的指定窗口。当鼠标悬停在捕获窗口上时，或者在鼠标悬停在捕获窗口上且按钮仍处于关闭状态时按下鼠标按钮时， SetCapture 捕获鼠标输入。一次只能有一个窗口可以捕获鼠标。 如果鼠标光标位于由另一个线程创建的窗口上，则仅当鼠标按钮按下时，系统才会将鼠标输入定向到指定的窗口。
SetDoubleClickTime	设置鼠标的双击时间。双击是鼠标按钮的一系列两次单击，第二次单击发生在第一次之后的指定时间内。双击时间是双击的第一次和第二次单击之间可能发生的最大毫秒数。
SwapMouseButton	反转或还原鼠标左右按钮的含义。
TrackMouseEvent	当鼠标指针离开窗口或将鼠标悬停在窗口上指定时间量时发布消息。

以下函数已过时。

函数	说明
mouse_event	合成鼠标运动和按钮单击。

通知

名称	说明
WM_APPCOMMAND	通知窗口用户生成了应用程序命令事件，例如，通过使用鼠标单击应用程序命令按钮或在键盘上键入应用程序命令键。
WM_CAPTURECHANGED	发送到丢失鼠标捕获的窗口。
WM_LBUTTONDOWNDBLCLK	当用户在光标位于窗口的工作区时双击鼠标左键时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_LBUTTONDOWN	当用户在光标位于窗口的工作区时按下鼠标左键时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_LBUTTONUP	当用户释放鼠标左键时光标位于窗口的工作区时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。

名称	说明
WM_MBUTTONDOWNDBLCLK	当用户在光标位于窗口的工作区时双击鼠标中间按钮时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_MBUTTONDOWN	当用户在光标位于窗口的工作区时按下鼠标中间按钮时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_MBUTTONUP	当用户释放鼠标中键时光标位于窗口的工作区时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_MOUSEACTIVATE	当光标处于非活动窗口且用户按下鼠标按钮时发送。仅当子窗口将其传递给 DefWindowProc 函数时，父窗口才会收到此消息。
WM_MOUSEOVER	当光标悬停在窗口的工作区上之前调用 TrackMouseEvent 中指定的时间段时，发布到窗口。
WM_MOUSEWHEEL	当鼠标的水平滚轮倾斜或旋转时，发送到焦点窗口。 DefWindowProc 函数将消息传播到窗口的父级。不应有消息的内部转发，因为 DefWindowProc 会将其传播到父链上，直到找到处理它的窗口。
WM_MOUSELEAVE	当光标离开之前调用 TrackMouseEvent 中指定的窗口工作区时，发布到窗口。
WM_MOUSEMOVE	光标移动时发布到窗口。如果未捕获鼠标，消息将发布到包含光标的窗口。否则，消息将发布到捕获了鼠标的窗口。
WM_MOUSEWHEEL	当鼠标滚轮旋转时，发送到焦点窗口。 DefWindowProc 函数将消息传播到窗口的父级。不应有消息的内部转发，因为 DefWindowProc 会将其传播到父链上，直到找到处理它的窗口。
WM_NCHITTEST	发送到窗口以确定窗口的哪个部分对应于特定的屏幕坐标。例如，当光标移动、按下或释放鼠标按钮或响应对 WindowFromPoint 等函数的调用时，可能会发生这种情况。如果未捕获鼠标，则会将消息发送到光标下方的窗口。否则，消息将发送到已捕获鼠标的窗口。
WM_NCLBUTTONDOWNDBLCLK	当用户在光标位于窗口的非工作区内时双击鼠标左键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCLBUTTONDOWN	当用户在光标位于窗口的非工作区内时按下鼠标左键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCLBUTTONUP	当用户释放鼠标左键时，光标位于窗口的非工作区内时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

名称	说明
WM_NCMBUTTONDOWNDBLCLK	当用户在光标位于窗口的非工作区内时双击鼠标中间按钮时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCMBUTTONDOWN	当用户按下鼠标中键时光标位于窗口的非工作区内时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCMBUTTONUP	当用户释放鼠标中键时光标位于窗口的非工作区内时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCMOUSEHOVER	在之前调用 TrackMouseEvent 中指定的时间段内光标悬停在窗口的非工作区上时，发布到窗口。
WM_NCMOUSELEAVE	当光标离开之前调用 TrackMouseEvent 中指定的窗口的非工作区时，将发布到窗口。
WM_NCMOUSEMOVE	当光标在窗口的非工作区内移动时发布到窗口。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCRBUTTONDOWNDBLCLK	当用户在光标位于窗口的非工作区内时双击鼠标右键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCRBUTTONDOWN	当用户在光标位于窗口的非工作区内时按下鼠标右键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCRBUTTONUP	当用户释放鼠标右键时光标位于窗口的非工作区内时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCXBUTTONDOWNDBLCLK	当用户在光标位于窗口的非工作区时双击第一个或第二个 X 按钮时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCXBUTTONDOWNS	当用户按下第一个或第二个 X 按钮时，光标位于窗口的非工作区时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_NCXBUTTONUP	当用户释放第一个或第二个 X 按钮时，光标位于窗口的非工作区时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。
WM_RBUTTONDOWNDBLCLK	当用户在光标位于窗口的工作区时双击鼠标右键时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。

名称	说明
WM_RBUTTONDOWN	当用户光标位于窗口的工作区时按下鼠标右键时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_RBUTTONUP	当用户释放鼠标右键时光标位于窗口的工作区时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_XBUTTONDOWNDBLCLK	当用户在光标位于窗口的工作区时双击第一个或第二个 X 按钮时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_XBUTTONDOWN	当用户在光标位于窗口工作区时按下第一个或第二个 X 按钮时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。
WM_XBUTTONUP	当用户释放第一个或第二个 X 按钮时，光标位于窗口的工作区时发布。如果未捕获鼠标，消息将发布到光标下方的窗口中。否则，消息将发布到捕获了鼠标的窗口。

结构

名称	说明
HARDWAREINPUT	包含有关由键盘或鼠标以外的输入设备生成的模拟消息的信息。
输入	包含用于合成输入事件（如击键、鼠标移动和鼠标单击）的信息。
LASTINPUTINFO	包含最后一次输入的时间。
MOUSEINPUT	包含有关模拟鼠标事件的信息。
MOUSEMOVEPOINT	包含有关屏幕坐标中鼠标位置的信息。
TRACKMOUSEEVENT	由 TrackMouseEvent 函数用于跟踪鼠标指针何时离开窗口或将鼠标悬停在窗口上指定时间量。

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获取帮助

鼠标输入概述

项目 • 2023/07/01

鼠标是应用程序的重要（但可选）用户输入设备。一个编写良好的应用程序应该包括一个鼠标接口，但它不应仅依赖鼠标来获取用户输入。该应用程序还应提供完整的键盘支持。

应用程序以发送或发布到其窗口的消息形式接收鼠标输入。

本部分涵盖了以下主题：

- [鼠标光标](#)
- [鼠标捕获](#)
- [鼠标单击锁定](#)
- [鼠标配置](#)
- [XBUTTON](#)
- [鼠标消息](#)
 - [工作区鼠标消息](#)
 - [非工作区鼠标消息](#)
 - [WM_NCHITTEST 消息](#)
- [鼠标声音波形](#)
- [鼠标消失](#)
- [鼠标滚轮](#)
- [窗口激活](#)

鼠标光标

当用户移动鼠标时，系统会在屏幕上移动一个称为鼠标光标的位图。鼠标光标包含一个称为热点的单像素点，系统跟踪该点并将其识别为光标的位置。当鼠标事件发生时，包含热点的窗口通常会接收到事件生成的鼠标消息。窗口不需要处于活动状态或具有键盘焦点来接收鼠标消息。

系统维护一个控制鼠标速度的变量，即光标在用户移动鼠标时移动的距离。可以使用带有 SPI_GETMOUSE 或 SPI_SETMOUSE 标志的 [SystemParametersInfo](#) 函数来检索或设置鼠标速度。有关鼠标光标的详细信息，请参阅[光标](#)。

鼠标捕获

当鼠标事件发生时，系统通常会向包含光标热点的窗口发布鼠标消息。应用程序可以通过使用 [SetCapture](#) 函数将鼠标消息路由到特定窗口来更改此行为。该窗口会收到所有鼠

标消息，直到应用程序调用 [ReleaseCapture](#) 函数或指定另一个捕获窗口，或者直到用户单击由另一个线程创建的窗口。

当鼠标捕获发生变化时，系统会向失去鼠标捕获的窗口发送 [WM_CAPTURECHANGED](#) 消息。消息的 IParam 参数指定获取鼠标捕获的窗口的句柄。

只有前台窗口可以捕获鼠标输入。当后台窗口尝试捕获鼠标输入时，它只接收当光标热点位于窗口可见部分内时发生的鼠标事件的消息。

如果窗口必须接收所有鼠标输入，即使光标移出窗口，捕获鼠标输入也很有用。例如，应用程序通常在按下鼠标按钮事件后跟踪光标位置，直到鼠标按钮弹起事件发生。如果应用程序未捕获鼠标输入，并且用户在窗口外松开鼠标按键，则窗口不会收到按钮弹起消息。

线程可以使用 [GetCapture](#) 函数来确定其某个窗口是否捕获了鼠标。如果线程的某个窗口捕获了鼠标，[GetCapture](#) 将检索窗口的句柄。

鼠标单击锁定

“鼠标单击锁定”辅助功能使用户能够在单击后锁定主鼠标按钮。对于应用程序，该按钮仍似乎是被按下的状态。要解锁按钮，应用程序可以发送任何鼠标消息，或者用户可以单击任何鼠标按钮。用户可通过此功能更轻松地进行复杂的鼠标组合。例如，具有某些身体限制的用户可以更轻松地突出显示文本、拖动对象或打开菜单。有关详细信息，请参阅以下标志和 [SystemParametersInfo](#) 中的备注：

- SPI_GETMOUSECLICKLOCK
- SPI_SETMOUSECLICKLOCK
- SPI_GETMOUSECLICKLOCKTIME
- SPI_SETMOUSECLICKLOCKTIME

鼠标配置

虽然鼠标是应用程序的重要输入设备，但并非每个用户都有鼠标。应用程序可以通过将 [SM_MOUSEPRESENT](#) 值传递给 [GetSystemMetrics](#) 函数来确定系统是否包含鼠标。

Windows 支持具有最多三个按钮的鼠标。在三键鼠标上，按钮被指定为左键、中键和右键。与鼠标按钮相关的消息和命名常量使用字母 L、M 和 R 来标识按钮。单键鼠标上的按钮被认为是左键。尽管 Windows 支持具有多个按钮的鼠标，但大多数应用程序主要使用左键，而很少使用其他按钮（如果有的话）。

应用程序还可以支持鼠标滚轮。可以按下或旋转鼠标滚轮。按下鼠标滚轮时，它充当中间（第三个）按钮，向应用程序发送正常的中键消息。旋转鼠标滚轮时，会向应用程序

发送一条滚轮消息。有关详细信息，请参阅[鼠标滚轮部分](#)。

应用程序可以支持应用程序命令按钮。这些按钮称为 X 按钮，用于更轻松地访问 Internet 浏览器、电子邮件和媒体服务。当按下 X 按钮时，系统会向应用程序发送一条 [WM_APPCOMMAND](#) 消息。有关详细信息，请参阅 [WM_APPCOMMAND](#) 消息中的说明。

应用程序可以通过将 [SM_CMOUSEBUTTONS](#) 值传递给 [GetSystemMetrics](#) 函数来确定鼠标按钮的数量。要为惯用左手的用户配置鼠标，应用程序可以使用 [SwapMouseButton](#) 函数来反转鼠标左键和右键的含义。将 [SPI_SETMOUSEBUTTONSWAP](#) 值传递给 [SystemParametersInfo](#) 函数是另一种反转按钮含义的方法。但是请注意，鼠标是共享资源，因此反转按钮的含义会影响所有应用程序。

XBUTTON

Windows 支持具有五个按钮的鼠标。除了左键、中键、右键之外，还有 XBUTTON1 和 XBUTTON2，它们在使用浏览器时提供向后和向前导航。

窗口管理器通过 [WM_XBUTTON*](#) 和 [WM_NCXBUTTON*](#) 消息支持 XBUTTON1 和 XBUTTON2。这些消息中 [WPARAM](#) 的 [HIWORD](#) 包含一个标志，指示按下了哪个 X 按钮。由于这些鼠标消息也适用于常量 [WM_MOUSEFIRST](#) 和 [WM_MOUSELAST](#)，因此应用程序可以使用 [GetMessage](#) 或 [PeekMessage](#) 筛选所有鼠标消息。

以下项支持 XBUTTON1 和 XBUTTON2：

- [WM_APPCOMMAND](#)
- [WM_NCXBUTTONDBLCLK](#)
- [WM_NCXBUTTONDOWNG](#)
- [WM_NCXBUTTONUP](#)
- [WM_XBUTTONDBLCLK](#)
- [WM_XBUTTONDOWNG](#)
- [WM_XBUTTONUP](#)
- [MOUSEHOOKSTRUCTEX](#)

修改了以下 API 以支持这些按钮：

- [mouse_event](#)
- [ShellProc](#)
- [MSLLHOOKSTRUCT](#)
- [MOUSEINPUT](#)
- [WM_PARENTNOTIFY](#)

组件应用程序中的子窗口不太可能直接执行 XBUTTON1 和 XBUTTON2 的命令。因此，当单击 X 按钮时，DefWindowProc 会向窗口发送 WM_APPCOMMAND 消息。DefWindowProc 还会将 WM_APPCOMMAND 消息发送到其父窗口。这类似于通过右键单击调用上下文菜单的方式 - DefWindowProc 将 WM_CONTEXTMENU 消息发送到菜单，并将其发送到其父级。此外，如果 DefWindowProc 收到顶级窗口的 WM_APPCOMMAND 消息，它会调用代码为 HSHELL_APPCOMMAND 的 shell 挂钩。

支持具有用于浏览器功能、媒体功能、应用程序启动和电源管理的额外键的键盘。有关详细信息，请参阅[用于浏览和其他功能的键盘键](#)。

鼠标消息

当用户移动鼠标或者按下或松开鼠标按钮时，鼠标会生成一个输入事件。系统将鼠标输入事件转换为消息，并将其发布到相应线程的消息队列中。当鼠标消息的发布速度快于线程处理速度时，系统会丢弃除最新鼠标消息以外的所有消息。

当光标位于窗口边框内发生鼠标事件时或当窗口已捕获鼠标时，窗口会收到鼠标消息。鼠标消息分为两组：工作区消息和非工作区消息。通常，应用程序处理工作区消息并忽略非工作区消息。

本部分涵盖了以下主题：

- [工作区鼠标消息](#)
- [非工作区鼠标消息](#)
- [WM_NCHITTEST 消息](#)

工作区鼠标消息

当窗口的工作区内发生鼠标事件时，窗口会收到工作区鼠标消息。当用户在工作区内移动光标时，系统会将 WM_MOUSEMOVE 消息发布到窗口。当光标位于工作区内时，如果用户按下或松开鼠标按钮，它将发布以下消息之一。

消息	含义
WM_LBUTTONDOWNDBLCLK	双击鼠标左键。
WM_LBUTTONDOWN	鼠标左按钮曾按下。
WM_LBUTTONUP	松开鼠标左键。
WM_MBUTTONDOWNDBLCLK	双击鼠标中键。
WM_MBUTTONDOWN	鼠标中按钮曾按下。
WM_MBUTTONUP	松开鼠标中键。

消息	含义
WM_RBUTTONDOWNDBLCLK	双击鼠标右键。
WM_RBUTTONDOWN	鼠标右按钮曾按下。
WM_RBUTTONUP	松开鼠标右键。
WM_XBUTTONDOWNDBLCLK	双击鼠标 X 按钮。
WM_XBUTTONDOWN	按下鼠标 X 按钮。
WM_XBUTTONUP	松开鼠标 X 按钮。

此外，应用程序可以调用 [TrackMouseEvent](#) 函数，让系统发送另外两条消息。当光标悬停在工作区上一段时间后，它会发布 [WM_MOUSEHOVER](#) 消息。当光标离开工作区时，它会发布 [WM_MOUSELEAVE](#) 消息。

消息参数

工作区鼠标消息的 IParam 参数指示光标热点的位置。低序字表示热点的 x 坐标，高序字表示 y 坐标。坐标在工作区坐标中指定。在工作区坐标系中，屏幕上的所有点都是相对于工作区左上角的坐标 (0,0) 指定的。

wParam 参数包含指示其他鼠标按钮以及 CTRL 和 SHIFT 键在鼠标事件发生时的状态的标志。当鼠标消息处理取决于另一鼠标按钮或 CTRL 或 SHIFT 键的状态时，可以检查这些标志。wParam 参数可以是以下值的组合。

值	说明
MK_CONTROL	按下了 Ctrl 键。
MK_LBUTTON	按下了鼠标左键。
MK_MBUTTON	按下了鼠标中键。
MK_RBUTTON	按下了鼠标右键。
MK_SHIFT	按下了 Shift 键。
MK_XBUTTON1	按下了第一个 X 按钮。
MK_XBUTTON2	按下了第二个 X 按钮。

双击消息

当用户快速连续两次单击鼠标按钮时，系统会生成双击消息。当用户单击某个按钮时，系统将建立一个围绕光标热点居中的矩形。它还标记了单击发生的时间。当用户第二次单击同一按钮时，系统会确定热点是否仍在矩形内，并计算自第一次单击后经过的时间。如果热点仍在矩形范围内，并且经过的时间没有超过双击超时值，则系统生成双击消息。

应用程序可以分别使用 [GetDoubleClickTime](#) 和 [SetDoubleClickTime](#) 函数获取和设置双击超时值。或者，应用程序可以结合使用 [SystemParametersInfo](#) 函数和 SPI_SETDOUBLECLICKTIME 标志来设置双击超时值。它还可以通过将 SPI_SETDOUBLECLKWIDTH 和 SPI_SETDOUBLECLKHEIGHT 标志传递给 [SystemParametersInfo](#) 来设置系统用来检测双击的矩形的大小。但是请注意，设置双击超时值和矩形会影响所有应用程序。

默认情况下，应用程序定义的窗口不会接收双击消息。由于生成双击消息涉及系统开销，因此这些消息仅针对属于具有 CS_DBLCLKS 类样式的类的窗口生成。应用程序必须在注册窗口类时设置此样式。有关详细信息，请参阅[窗口类](#)。

双击消息始终是四条消息系列中的第三条消息。前两条消息是第一次单击生成的按钮按下和按钮弹起消息。第二次单击会生成双击消息，然后是另一个按钮弹起消息。例如，双击鼠标左键会生成以下消息序列：

1. [WM_LBUTTONDOWN](#)
2. [WM_LBUTTONUP](#)
3. [WM_LBUTTONDOWNDBLCLK](#)
4. [WM_LBUTTONUP](#)

因为窗口始终在接收到双击消息之前接收到按钮按下消息，所以应用程序通常使用双击消息来扩展在按钮按下消息期间开始的任务。例如，当用户单击 Microsoft 画图调色板中的一种颜色时，画图会在调色板旁边显示所选颜色。当用户双击一种颜色时，画图会显示该颜色并打开“编辑颜色”对话框。

非工作区鼠标消息

当鼠标事件发生在窗口的任何部分（工作区除外）时，窗口会收到非工作区鼠标消息。窗口的非工作区由边框、菜单栏、标题栏、滚动条、窗口菜单、最小化按钮和最大化按钮组成。

系统生成非工作区消息，主要供其自身使用。例如，当光标热点移动到窗口的边框时，系统使用非工作区消息将光标更改为双向箭头。窗口必须将非工作区鼠标消息传递到 [DefWindowProc](#) 函数，才能利用内置鼠标接口。

每个工作区鼠标消息都有对应的非工作区鼠标消息。这些消息的名称相似，只不过非工作区消息的命名常量包含字母 NC。例如，在非工作区移动光标会生成 WM_NCMOUSEMOVE 消息，当光标位于非工作区时按下鼠标左键会生成 WM_NCLBUTTONDOWN 消息。

非工作区鼠标消息的 IParam 参数是一个包含光标热点的 x 和 y 坐标的结构。与工作区鼠标消息的坐标不同，坐标以屏幕坐标而不是工作区坐标指定。在屏幕坐标系中，屏幕上的所有点都是相对于屏幕左上角坐标 (0,0) 指定。

wParam 参数包含一个命中测试值，该值指示鼠标事件在非工作区中发生的位置。以下部分解释了命中测试值的用途。

WM_NCHITTEST 消息

每当发生鼠标事件时，系统都会向包含光标热点的窗口或捕获鼠标的窗口发送 WM_NCHITTEST 消息。系统使用此消息来确定是发送工作区鼠标消息还是非工作区鼠标消息。必须接收鼠标移动和鼠标按钮消息的应用程序必须将 WM_NCHITTEST 消息传递给 DefWindowProc 函数。

WM_NCHITTEST 消息的 IParam 参数包含光标热点的屏幕坐标。DefWindowProc 函数检查坐标并返回指示热点位置的命中测试值。命中测试值可以是以下值之一。

值	热点位置
HTBORDER	在没有大小调整边框的窗口边框中。
HTBOTTOM	在窗口的下水平边框中。
HTBOTTOMLEFT	在窗口边框的左下角。
HTBOTTOMRIGHT	在窗口边框的右下角。
HTCAPTION	在标题栏中。
HTCLIENT	在工作区中。
HTCLOSE	在“关闭”按钮中。
HTERROR	在屏幕背景上或窗口之间的分割线上（与 HTNOWHERE 相同，但 DefWindowProc 函数会生成系统蜂鸣音以指示错误）。
HTGROWBOX	在大小框中（与 HTSIZE 相同）。
HTHELP	在“帮助”按钮中。
HTHSCROLL	在水平滚动条中。
HTLEFT	在窗口的左边框中。

值	热点位置
HTMENU	在菜单中。
HTMAXBUTTON	在“最大化”按钮中。
HTMINBUTTON	在“最小化”按钮中。
HTNOWHERE	在屏幕背景上，或在窗口之间的分隔线上。
HTREDUCE	在“最小化”按钮中。
HTRIGHT	在窗口的右边框中。
HTSIZE	在大小框中（与 HTGROWBOX 相同）。
HTSYSMENU	在子窗口的“系统”菜单或“关闭”按钮中。
HTTOP	在窗口的上水平边框中。
HTTOPLEFT	在窗口边框的左上角。
HTTOPRIGHT	在窗口边框的右上角。
HTTRANSPARENT	在当前被同一线程中的另一窗口覆盖的窗口中。
HTVSCROLL	在垂直滚动条中。
HTZOOM	在“最大化”按钮中。

如果光标位于窗口的工作区，[DefWindowProc](#) 会将 HTCLIENT 命中测试值返回给窗口过程。当窗口过程将这段代码返回给系统时，系统将光标热点的屏幕坐标转换为工作区坐标，然后发布相应的工作区鼠标消息。

当光标热点位于窗口的非工作区时，[DefWindowProc](#) 函数返回其他命中测试值之一。当窗口过程返回这些命中测试值之一时，系统会发布一条非工作区鼠标消息，将命中测试值放在消息的 wParam 参数中，将光标坐标放置在 lParam 参数中。

鼠标声音波形

当用户按下和松开 CTRL 键时，鼠标声音波形辅助功能会在指针周围短暂显示几个同心圆。此功能可帮助用户在杂乱或分辨率设置过高的屏幕上、在质量较差的显示器上或为视力受损的用户定位鼠标指针。有关详细信息，请参阅 [SystemParametersInfo](#) 中的以下标志：

SPI_GETMOUSESONAR

SPI_SETMOUSESONAR

鼠标消失

鼠标消失辅助功能会在用户键入时隐藏指针。当用户移动鼠标时，鼠标指针会再次出现。此功能可防止指针遮挡正在键入的文本，例如，在电子邮件或其他文档中。有关详细信息，请参阅 [SystemParametersInfo](#) 中的以下标志：

SPI_GETMOUSEVANISH

SPI_SETMOUSEVANISH

鼠标滚轮

鼠标滚轮结合了滚轮和鼠标按钮的功能。滚轮具有离散的均匀间距的凹槽。旋转滚轮，遇到每个凹槽时，都会向应用程序发送一条滚轮消息。滚轮按钮也可以用作普通的 Windows 中（第三个）键。按下并松开鼠标滚轮会发送标准的 [WM_MBUTTONDOWN](#) 和 [WM_MBUTTONUP](#) 消息。双击第三个按钮会发送标准的 [WM_MBUTTONDOWNDBLCLK](#) 消息。

通过 [WM_MOUSEWHEEL](#) 消息支持鼠标滚轮。

旋转鼠标会将 [WM_MOUSEWHEEL](#) 消息发送到焦点窗口。[DefWindowProc](#) 函数将消息传播到窗口的父级。不应有消息的内部转发，因为 [DefWindowProc](#) 将它向上传播到父链，直到找到处理窗口。

确定滚动行数

应用程序应使用 [SystemParametersInfo](#) 函数检索每次滚动操作（滚轮凹槽）的文档滚动的行数。要检索行数，应用程序会进行以下调用：

```
SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, pulScrollLines, 0)
```

变量“pulScrollLines”指向一个无符号整数值，当鼠标滚轮在没有修改键的情况下旋转时，该值接收建议的滚动行数：

- 如果此数字为 0，则不应发生滚动。
- 如果此数字为 WHEEL_PAGESCROLL，滚轮应解释为在滚动条的向下翻页或向上翻页区域中单击一次。

- 如果要滚动的行数大于可查看的行数，则滚动操作也应解释为向下翻页或向上翻页操作。

滚动行数的默认值为 3。如果用户通过使用“控制面板”中的“鼠标属性”表更改滚动行数，操作系统会将 WM_SETTINGCHANGE 消息广播到指定 SPI_SETWHEELSCROLLLINES 的所有顶级窗口。当应用程序收到 WM_SETTINGCHANGE 消息时，它可以通过调用以下项获取新的滚动行数：

```
SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, pulScrollLines, 0)
```

滚动控件

下表列出了具有滚动功能的控件（包括用户设置的滚动行）。

控制	滚动
编辑控件	纵向和横向。
列表框控件	纵向和横向。
组合框	当没有下拉时，每次滚动都会检索下一项或上一项。下拉时，每次滚动都会将消息转发到相应滚动的列表框。
CMD（命令行）	垂直。
树视图	纵向和横向。
“列表视图”	纵向和横向。
向上/向下滚动	一次一项。
跟踪条滚动	一次一项。
Microsoft Rich Edit 1.0	垂直。请注意，Exchange 客户端有自己的列表视图和树视图控件版本，不支持滚轮。
Microsoft Rich Edit 2.0	垂直。

检测带滚轮的鼠标

要确定是否连接了带滚轮的鼠标，请使用 SM_MOUSEWHEELPRESENT 调用 [GetSystemMetrics](#)。返回值“TRUE”表示鼠标已连接。

以下示例来自多行编辑控件的窗口过程：

```
BOOL ScrollLines(
    PWNDDATA pwndData,    //scrolls the window indicated
    int cLinesToScroll); //number of times

short gcWheelDelta; //wheel delta from roll
PWNDDATA pWndData; //pointer to structure containing info about the window
UINT gucWheelScrollLines=0;//number of lines to scroll on a wheel rotation

gucWheelScrollLines = SystemParametersInfo(SPI_GETWHEELSCROLLLINES,
                                           0,
                                           pulScrollLines,
                                           0);

case WM_MOUSEWHEEL:
/*
 * Do not handle zoom and datazoom.
 */
if (wParam & (MK_SHIFT | MK_CONTROL)) {
    goto PassToDefaultWindowProc;
}

gcWheelDelta -= (short) HIWORD(wParam);
if (abs(gcWheelDelta) >= WHEEL_DELTA && gucWheelScrollLines > 0)
{
    int cLineScroll;

    /*
     * Limit a roll of one (1) WHEEL_DELTA to
     * scroll one (1) page.
     */
    cLineScroll = (int) min(
        (UINT) pWndData->ichLinesOnScreen - 1,
        gucWheelScrollLines);

    if (cLineScroll == 0) {
        cLineScroll++;
    }

    cLineScroll *= (gcWheelDelta / WHEEL_DELTA);
    assert(cLineScroll != 0);

    gcWheelDelta = gcWheelDelta % WHEEL_DELTA;
    return ScrollLines(pWndData, cLineScroll);
}

break;
```

窗口激活

当用户单击非活动顶级窗口或非活动顶级窗口的子窗口时，系统会将 [WM_MOUSEACTIVATE](#) 消息（以及其他信息）发送到顶级窗口或子窗口。系统会在将 [WM_NCHITTEST](#) 消息发布到窗口之后（但在发布按钮按下消息之前）发送此消息。当 [WM_MOUSEACTIVATE](#) 传递给 [DefWindowProc](#) 函数时，系统会激活顶级窗口，然后将按钮按下消息发送给顶级窗口或子窗口。

通过处理 [WM_MOUSEACTIVATE](#)，窗口可以控制顶级窗口是否因鼠标单击而成为活动窗口，以及被单击的窗口是否接收后续按钮按下消息。它通过在处理 [WM_MOUSEACTIVATE](#) 后返回以下值之一来实现此操作。

值	含义
MA_ACTIVATE	激活窗口，并且不丢弃鼠标消息。
MA_NOACTIVATE	不激活窗口，并且不丢弃鼠标消息。
MA_ACTIVATEANDEAT	激活窗口，并丢弃鼠标消息。
MA_NOACTIVATEANDEAT	不激活窗口，但丢弃鼠标消息。

请参阅

[利用高分辨率鼠标移动](#)

反馈

此页面是否有帮助？

[是](#)

[否](#)

[在 Microsoft Q&A 获取帮助](#)

使用鼠标输入

项目 · 2023/06/13

本部分介绍与鼠标输入关联的任务。

- 跟踪鼠标光标
- 使用鼠标绘制线条
- 处理双击消息
- 选择文本行
- 在包含嵌入对象的文档中使用鼠标滚轮
- 检索鼠标滚轮滚动行数

跟踪鼠标光标

应用程序通常执行涉及跟踪鼠标光标位置的任务。例如，大多数绘图应用程序在绘制操作期间跟踪鼠标光标的位置，从而允许用户通过拖动鼠标在窗口的工作区中绘图。Word 处理应用程序还可以跟踪光标，使用户能够通过单击并拖动鼠标来选择单词或文本块。

跟踪游标通常涉及处理 `WM_LBUTTONDOWN`、`WM_MOUSEMOVE` 和 `WM_LBUTTONUP` 消息。窗口通过检查 `WM_LBUTTONDOWN` 消息的 *lParam* 参数中提供的游标位置来确定何时开始跟踪 **光标**。例如，仅当光标位于文本行上时发生 `WM_LBUTTONDOWN` 消息时，字处理应用程序才会开始跟踪游标，但如果它已超过文档的末尾，则不会开始跟踪光标。

窗口通过处理在鼠标移动时发布到窗口的 `WM_MOUSEMOVE` 消息流来跟踪光标的位 置。处理 `WM_MOUSEMOVE` 消息通常涉及工作区中的重复绘制或绘图操作。例如，绘 图应用程序可能会在鼠标移动时重复重新绘制线条。窗口使用 `WM_LBUTTONUP` 消息作 为停止跟踪光标的信号。

此外，应用程序可以调用 `TrackMouseEvent` 函数，让系统发送对跟踪游标有用的其他消息。当光标悬停在工作区上一定时间段时，系统会发布 `WM_MOUSEOVER` 消息。当 光标离开工作区时，它会发布 `WM_MOUSELEAVE` 消息。`WM_NCMOUSEOVER` 和 `WM_NCMOUSELEAVE` 消息是非工作区的相应消息。

使用鼠标绘制线条

本部分中的示例演示如何跟踪鼠标光标。它包含窗口过程的某些部分，使用户能够通过 拖动鼠标在窗口的工作区中绘制线条。

当窗口过程收到 `WM_LBUTTONDOWN` 消息时，它会捕获鼠标并保存光标的坐标，并使 用坐标作为线条的起点。它还使用 `ClipCursor` 函数在线条绘制操作期间将光标限制在工

作区。

在第一[个WM_MOUSEMOVE](#)消息期间，窗口过程将绘制一条从起始点到光标当前位置的线条。在后续WM_MOUSEMOVE消息期间，窗口过程通过用倒笔颜色在上一行上绘制来擦除上一行。然后，它从起始点到光标的 new 位置绘制一条新线。

[WM_LBUTTONDOWN](#)消息表示绘图操作结束。窗口过程释放鼠标捕获，并从工作区中释放鼠标。

```
LRESULT APIENTRY MainWndProc(HWND hwndMain, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;                      // handle to device context
    RECT rcClient;                // client area rectangle
    POINT ptClientUL;             // client upper left corner
    POINT ptClientLR;             // client lower right corner
    static POINTS ptsBegin;        // beginning point
    static POINTS ptsEnd;          // new endpoint
    static POINTS ptsPrevEnd;      // previous endpoint
    static BOOL fPrevLine = FALSE; // previous line flag

    switch (uMsg)
    {
        case WM_LBUTTONDOWN:
            // Capture mouse input.

            SetCapture(hwndMain);

            // Retrieve the screen coordinates of the client area,
            // and convert them into client coordinates.

            GetClientRect(hwndMain, &rcClient);
            ptClientUL.x = rcClient.left;
            ptClientUL.y = rcClient.top;

            // Add one to the right and bottom sides, because the
            // coordinates retrieved by GetClientRect do not
            // include the far left and lowermost pixels.

            ptClientLR.x = rcClient.right + 1;
            ptClientLR.y = rcClient.bottom + 1;
            ClientToScreen(hwndMain, &ptClientUL);
            ClientToScreen(hwndMain, &ptClientLR);

            // Copy the client coordinates of the client area
            // to the rcClient structure. Confine the mouse cursor
            // to the client area by passing the rcClient structure
            // to the ClipCursor function.

            SetRect(&rcClient, ptClientUL.x, ptClientUL.y,
```

```

        ptClientLR.x, ptClientLR.y);
ClipCursor(&rcClient);

// Convert the cursor coordinates into a POINTS
// structure, which defines the beginning point of the
// line drawn during a WM_MOUSEMOVE message.

ptsBegin = MAKEPOINTS(lParam);
return 0;

case WM_MOUSEMOVE:

    // When moving the mouse, the user must hold down
    // the left mouse button to draw lines.

    if (wParam & MK_LBUTTON)
    {

        // Retrieve a device context (DC) for the client area.

        hdc = GetDC(hwndMain);

        // The following function ensures that pixels of
        // the previously drawn line are set to white and
        // those of the new line are set to black.

        SetROP2(hdc, R2_NOTXORPEN);

        // If a line was drawn during an earlier WM_MOUSEMOVE
        // message, draw over it. This erases the line by
        // setting the color of its pixels to white.

        if (fPrevLine)
        {
            MoveToEx(hdc, ptsBegin.x, ptsBegin.y,
                      (LPPOINT) NULL);
            LineTo(hdc, ptsPrevEnd.x, ptsPrevEnd.y);
        }

        // Convert the current cursor coordinates to a
        // POINTS structure, and then draw a new line.

        ptsEnd = MAKEPOINTS(lParam);
        MoveToEx(hdc, ptsBegin.x, ptsBegin.y, (LPPOINT) NULL);
        LineTo(hdc, ptsEnd.x, ptsEnd.y);

        // Set the previous line flag, save the ending
        // point of the new line, and then release the DC.

        fPrevLine = TRUE;
        ptsPrevEnd = ptsEnd;
        ReleaseDC(hwndMain, hdc);
    }
    break;
}

```

```
case WM_LBUTTONDOWN:
    // The user has finished drawing the line. Reset the
    // previous line flag, release the mouse cursor, and
    // release the mouse capture.

    fPrevLine = FALSE;
    ClipCursor(NULL);
    ReleaseCapture();
    return 0;

case WM_DESTROY:
    PostQuitMessage(0);
    break;

// Process other messages.
```

处理双击消息

若要接收双击消息，窗口必须属于具有 [CS_DBCLKS](#) 类样式的窗口类。 注册窗口类时设置此样式，如以下示例所示。

```
BOOL InitApplication(HINSTANCE hInstance)
{
    WNDCLASS wc;

    wc.style = CS_DBCLKS | CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = (WNDPROC) MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_IBEAM);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = "MainMenu";
    wc.lpszClassName = "MainWClass";

    return RegisterClass(&wc);
}
```

双击消息前面始终带有按钮关闭消息。 出于此原因，应用程序通常使用双击消息来扩展它在按钮关闭消息期间开始的任务。

选择文本行

本部分中的示例取自一个简单的字处理应用程序。它包含的代码使用户能够通过单击文本行上的任意位置来设置插入点的位置，并通过双击该行上的任意位置选择(突出显示)文本行。

```
LRESULT APIENTRY MainWndProc(HWND hwndMain, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;                      // handle to device context
    TEXTMETRIC tm;                // font size data
    int i, j;                     // loop counters
    int cCR = 0;                  // count of carriage returns
    char ch;                      // character from input buffer
    static int nBegLine;           // beginning of selected line
    static int nCurrentLine = 0;   // currently selected line
    static int nLastLine = 0;       // last text line
    static int nCaretPosX = 0;     // x-coordinate of caret
    static int cch = 0;            // number of characters entered
    static int nCharWidth = 0;     // exact width of a character
    static char szHilite[128];    // text string to highlight
    static DWORD dwCharX;          // average width of characters
    static DWORD dwLineHeight;     // line height
    static POINTS ptsCursor;       // coordinates of mouse cursor
    static COLORREF crPrevText;   // previous text color
    static COLORREF crPrevBk;     // previous background color
    static PTCHAR pchInputBuf;    // pointer to input buffer
    static BOOL fTextSelected = FALSE; // text-selection flag
    size_t * pcch;
    HRESULT hResult;

    switch (uMsg)
    {
        case WM_CREATE:

            // Get the metrics of the current font.

            hdc = GetDC(hwndMain);
            GetTextMetrics(hdc, &tm);
            ReleaseDC(hwndMain, hdc);

            // Save the average character width and height.

            dwCharX = tm.tmAveCharWidth;
            dwLineHeight = tm.tmHeight;

            // Allocate a buffer to store keyboard input.

            pchInputBuf = (LPSTR) GlobalAlloc(GPTR,
                BUFSIZE * sizeof(TCHAR));

            return 0;
    }
}
```

```

case WM_CHAR:
    switch (wParam)
    {
        case 0x08: // backspace
        case 0x0A: // linefeed
        case 0x1B: // escape
            MessageBeep( (UINT) -1);
            return 0;

        case 0x09: // tab

            // Convert tabs to four consecutive spaces.

            for (i = 0; i < 4; i++)
                SendMessage(hwndMain, WM_CHAR, 0x20, 0);
            return 0;

        case 0x0D: // carriage return

            // Record the carriage return, and position the
            // caret at the beginning of the new line.

            pchInputBuf[cch++] = 0x0D;
            nCaretPosX = 0;
            nCurrentLine += 1;
            break;

        default: / displayable character

            ch = (char) wParam;
            HideCaret(hwndMain);

            // Retrieve the character's width, and display the
            // character.

            hdc = GetDC(hwndMain);
            GetCharWidth32(hdc, (UINT) wParam, (UINT) wParam,
                           &nCharWidth);
            TextOut(hdc, nCaretPosX,
                    nCurrentLine * dwLineHeight, &ch, 1);
            ReleaseDC(hwndMain, hdc);

            // Store the character in the buffer.

            pchInputBuf[cch++] = ch;

            // Calculate the new horizontal position of the
            // caret. If the new position exceeds the maximum,
            // insert a carriage return and reposition the
            // caret at the beginning of the next line.

            nCaretPosX += nCharWidth;
            if ((DWORD) nCaretPosX > dwMaxCharX)
            {
                nCaretPosX = 0;

```

```

        pchInputBuf[cch++] = 0x0D;
        ++nCurrentLine;
    }

    ShowCaret(hwndMain);

    break;
}
SetCaretPos(nCaretPosX, nCurrentLine * dwLineHeight);
nLastLine = max(nLastLine, nCurrentLine);
break;

// Process other messages.

case WM_LBUTTONDOWN:

    // If a line of text is currently highlighted, redraw
    // the text to remove the highlighting.

    if (fTextSelected)
    {
        hdc = GetDC(hwndMain);
        SetTextColor(hdc, crPrevText);
        SetBkColor(hdc, crPrevBk);
        hResult = StringCchLength(szHilite, 128/sizeof(TCHAR),
pcch);
        if (FAILED(hResult))
        {
            // TODO: write error handler
            TextOut(hdc, 0, nCurrentLine * dwLineHeight,
szHilite, *pcch);
            ReleaseDC(hwndMain, hdc);
            ShowCaret(hwndMain);
            fTextSelected = FALSE;
        }
    }

    // Save the current mouse-cursor coordinates.

    ptsCursor = MAKEPOINTS(lParam);

    // Determine which line the cursor is on, and save
    // the line number. Do not allow line numbers greater
    // than the number of the last line of text. The
    // line number is later multiplied by the average height
    // of the current font. The result is used to set the
    // y-coordinate of the caret.

    nCurrentLine = min((int)(ptsCursor.y / dwLineHeight),
nLastLine);

    // Parse the text input buffer to find the first
    // character in the selected line of text. Each
    // line ends with a carriage return, so it is possible
    // to count the carriage returns to find the selected

```

```

// line.

cCR = 0;
nBegLine = 0;
if (nCurrentLine != 0)
{
    for (i = 0; (i < cch) &&
        (cCR < nCurrentLine); i++)
    {
        if (pchInputBuf[i] == 0x0D)
            ++cCR;
    }
    nBegLine = i;
}

// Starting at the beginning of the selected line,
// measure the width of each character, summing the
// width with each character measured. Stop when the
// sum is greater than the x-coordinate of the cursor.
// The sum is used to set the x-coordinate of the caret.

hdc = GetDC(hwndMain);
nCaretPosX = 0;
for (i = nBegLine;
    (pchInputBuf[i] != 0x0D) && (i < cch); i++)
{
    ch = pchInputBuf[i];
    GetCharWidth32(hdc, (int) ch, (int) ch, &nCharWidth);
    if ((nCaretPosX + nCharWidth) > ptsCursor.x) break;
    else nCaretPosX += nCharWidth;
}
ReleaseDC(hwndMain, hdc);

// Set the caret to the user-selected position.

SetCaretPos(nCaretPosX, nCurrentLine * dwLineHeight);
break;

case WM_LBUTTONDOWN:
    // Copy the selected line of text to a buffer.

    for (i = nBegLine, j = 0; (pchInputBuf[i] != 0x0D) &&
        (i < cch); i++)
    {
        szHilite[j++] = pchInputBuf[i];
    }
    szHilite[j] = '\0';

    // Hide the caret, invert the background and foreground
    // colors, and then redraw the selected line.

    HideCaret(hwndMain);
    hdc = GetDC(hwndMain);
    crPrevText = SetTextColor(hdc, RGB(255, 255, 255));

```

```

        crPrevBk = SetBkColor(hdc, RGB(0, 0, 0));
        hResult = StringCchLength(szHilite, 128/sizeof(TCHAR), pcch);
        if (FAILED(hResult))
        {
        // TODO: write error handler
        }
        TextOut(hdc, 0, nCurrentLine * dwLineHeight, szHilite, *pcch);
        SetTextColor(hdc, crPrevText);
        SetBkColor(hdc, crPrevBk);
        ReleaseDC(hwndMain, hdc);

        fTextSelected = TRUE;
        break;

        // Process other messages.

    default:
        return DefWindowProc(hwndMain, uMsg, wParam, lParam);
    }
    return NULL;
}

```

在包含嵌入对象的文档中使用鼠标滚轮

此示例假定 Microsoft Word 文档包含各种嵌入对象：

- Microsoft Excel 电子表格
- 滚动以响应滚轮的嵌入列表框控件
- 不响应滚轮的嵌入文本框控件

[MSH_MOUSEWHEEL](#)消息始终发送到 Microsoft Word 中的main窗口。即使嵌入的电子表格处于活动状态，也是如此。下表说明了如何根据焦点处理MSH_MOUSEWHEEL消息。

重点	处理方式如下：
在于	
Word 文档	Word滚动文档窗口。
嵌入式 Excel 电子 表格	Word将邮件发布到 Excel。必须决定嵌入式应用程序是否响应消息。
嵌入式控件	由应用程序将消息发送到具有焦点的嵌入式控件，并检查返回代码以查看控件是否处理了它。如果控件未处理它，则应用程序应滚动文档窗口。例如，如果用户单击列表框，然后滚动滚轮，则该控件将滚动以响应滚轮旋转。如果用户单击文本框，然后旋转滚轮，则整个文档将滚动。

以下示例演示应用程序如何处理两个滚轮消息。

```
/**************************************************************************
* this code deals with MSH_MOUSEWHEEL
*************************************************************************/
#include "zmouse.h"

//
// Mouse Wheel rotation stuff, only define if we are
// on a version of the OS that does not support
// WM_MOUSEWHEEL messages.
//
#ifndef WM_MOUSEWHEEL
#define WM_MOUSEWHEEL WM_MOUSELAST+1
    // Message ID for IntelliMouse wheel
#endif

UINT uMSH_MOUSEWHEEL = 0;    // Value returned from
                            // RegisterWindowMessage()

/**************************************************************************

INT WINAPI WinMain(
    HINSTANCE hInst,
    HINSTANCE hPrevInst,
    LPSTR lpCmdLine,
    INT nCmdShow)
{
    MSG msg;
    BOOL bRet;

    if (!InitInstance(hInst, nCmdShow))
        return FALSE;

    //
    // The new IntelliMouse uses a Registered message to transmit
    // wheel rotation info. So register for it!

    uMSH_MOUSEWHEEL =
        RegisterWindowMessage(MSH_MOUSEWHEEL);
    if ( !uMSH_MOUSEWHEEL )
    {
        MessageBox(NULL,
            RegisterWindowMessage Failed!,
            "Error",MB_OK);
        return msg.wParam;
    }

    while (( bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
    {
```

```

        if (bRet == -1)
        {
            // handle the error and possibly exit
        }
        else
        {
            if (!TranslateAccelerator(ghwndApp,
                                      ghaccelTable,
                                      &msg))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }
    }

    return msg.wParam;
}

/*****************
* this code deals with WM_MOUSEWHEEL
*****************/
LONG APIENTRY MainWndProc(
    HWND hwnd,
    UINT msg,
    WPARAM wParam,
    LPARAM lParam)
{
    static int nZoom = 0;

    switch (msg)
    {
        //
        // Handle Mouse Wheel messages generated
        // by the operating systems that have built-in
        // support for the WM_MOUSEWHEEL message.
        //

        case WM_MOUSEWHEEL:
            ((short) HIWORD(wParam)< 0) ? nZoom-- : nZoom++;
            //
            // Do other wheel stuff...
            //
            break;

        default:
            //
            // uMSH_MOUSEWHEEL is a message registered by
            // the mswheel dll on versions of Windows that
            // do not support the new message in the OS.

            if( msg == uMSH_MOUSEWHEEL )

```

```

    {
        ((int)wParam < 0) ? nZoom-- : nZoom++;

        //
        // Do other wheel stuff...
        //
        break;
    }

    return DefWindowProc(hwnd,
                         msg,
                         wParam,
                         lParam);
}

return 0L;
}

```

检索鼠标滚轮滚动行数

以下代码允许应用程序使用 [SystemParametersInfo](#) 函数检索滚动行数。

```

#ifndef SPI_GETWHEELSCROLLLINES
#define SPI_GETWHEELSCROLLLINES    104
#endif

#include "zmouse.h"

/****************************************************************************
 * FUNCTION: GetNumScrollLines
 * Purpose : An OS independent method to retrieve the
 *            number of wheel scroll lines
 * Params   : none
 * Returns  : UINT: Number of scroll lines where WHEEL_PAGESCROLL
 *            indicates to scroll a page at a time.
 *****/
UINT GetNumScrollLines(void)
{
    HWND hdlMsWheel;
    UINT ucNumLines=3; // 3 is the default
    OSVERSIONINFO osversion;
    UINT uiMsh_MsgScrollLines;

    memset(&osversion, 0, sizeof(osversion));
    osversion.dwOSVersionInfoSize =sizeof(osversion);
    GetVersionEx(&osversion);

    if ((osversion.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS) ||
        (osversion.dwPlatformId == VER_PLATFORM_WIN32_NT) &&

```

```
(osversion.dwMajorVersion < 4) )    )
{
    hdlMsWheel = FindWindow(MSH_WHEELMODULE_CLASS,
                           MSH_WHEELMODULE_TITLE);
    if (hdlMsWheel)
    {
        uiMsh_MsgScrollLines = RegisterWindowMessage
                               (MSH_SCROLL_LINES);
        if (uiMsh_MsgScrollLines)
            ucNumLines = (int)SendMessage(hdlMsWheel,
                                         uiMsh_MsgScrollLines,
                                         0,
                                         0);
    }
}
else if ( (osversion.dwPlatformId ==
           VER_PLATFORM_WIN32_NT) &&
          (osversion.dwMajorVersion >= 4) )
{
    SystemParametersInfo(SPI_GETWHEELSCROLLLINES,
                         0,
                         &ucNumLines, 0);
}
return(ucNumLines);
}
```

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

鼠标输入参考

项目 • 2023/06/13

本节中包含的主题提供鼠标输入的参考规范。

在本节中

主题	说明
鼠标输入函数	
鼠标输入宏	
鼠标输入通知	
鼠标输入结构	

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

鼠标输入函数

项目 · 2023/08/19

在本节中

主题	说明
_TrackMouseEvent	当在指定时间内鼠标指针离开窗口或将鼠标悬停在窗口上时，发布消息。此函数调用 TrackMouseEvent （如果存在），否则会模拟它。
DragDetect	捕获鼠标并跟踪其移动，直到用户释放左键、按 ESC 键或将鼠标移动到围绕指定点的拖动矩形外部。拖动矩形的宽度和高度由 GetSystemMetrics 函数返回的 SM_CXDRAG 和 SM_CYDRAG 值指定。
GetCapture	检索任何捕获鼠标的窗口句柄（如果有）。一次只能有一个窗口捕获鼠标；无论光标是否在其边框内，此窗口都会收到鼠标输入。
GetDoubleClickTime	检索鼠标的当前双击时间。双击是鼠标按钮的一系列两次单击，第二次单击在第一次单击之后的指定时间内发生。双击时间是双击的第一次单击和第二次单击之间可能发生的最大毫秒数。最大双击时间为 5000 毫秒。
GetMouseMovePointsEx	检索最多 64 个鼠标或笔的先前坐标的历史记录。
mouse_event	mouse_event 函数合成鼠标运动和按钮单击。 注意： 此函数已被取代。请改用 SendInput 。
ReleaseCapture	从当前线程中的窗口释放鼠标捕获，并还原正常鼠标输入处理。捕获鼠标的窗口接收所有鼠标输入，而不考虑光标的位置，但当光标热点位于另一个线程的窗口中时单击鼠标按钮除外。
SetCapture	将鼠标捕获设置为属于当前线程的指定窗口。
SetDoubleClickTime	设置鼠标的双击时间。双击是鼠标按钮的一系列两次单击，第二次单击在第一次单击之后的指定时间内发生。双击时间是双击的第一次单击和第二次单击之间可能发生的最大毫秒数。
SwapMouseButton	反转或还原鼠标左键和右键的含义。
TrackMouseEvent	当在指定时间内鼠标指针离开窗口或将鼠标悬停在窗口上时，发布消息。 注意： _TrackMouseEvent 函数调用 TrackMouseEvent （如果存在），否则 _TrackMouseEvent 模拟 TrackMouseEvent 。

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

_TrackMouseEvent函数 (commctrl.h)

项目2023/08/22

当在指定时间内鼠标指针离开窗口或将鼠标悬停在窗口上时，发布消息。此函数调用 TrackMouseEvent（如果存在），否则会模拟它。

语法

C++

```
BOOL _TrackMouseEvent(  
    [in, out] LPTRACKMOUSEEVENT lpEventTrack  
>;
```

参数

[in, out] lpEventTrack

类型： LPTRACKMOUSEEVENT

指向包含跟踪信息的 TRACKMOUSEEVENT 结构的指针。

返回值

类型： BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	commctrl.h

Library	Comctl32.lib
DLL	Comctl32.dll

请参阅

概念性

[鼠标输入](#)

其他资源

引用

[SystemParametersInfo](#)

[TRACKMOUSEEVENT](#)

[TrackMouseEvent](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

DragDetect 函数 (winuser.h)

项目2024/03/04

捕获鼠标并跟踪其移动，直到用户释放左键、按 ESC 键或将鼠标移动到围绕指定点的拖动矩形外部。拖动矩形的宽度和高度由 GetSystemMetrics 函数返回的 SM_CXDRAG 和 SM_CYDRAG 值指定。

语法

C++

```
BOOL DragDetect(
    [in] HWND hwnd,
    [in] POINT pt
);
```

参数

[in] hwnd

类型: **HWND**

接收鼠标输入的窗口的句柄。

[in] pt

类型: **POINT**

鼠标的初始位置，以屏幕坐标表示。该函数使用此点来确定拖动矩形的坐标。

返回值

类型: **BOOL**

如果用户在按住左键的同时将鼠标移到了拖动矩形之外，则返回值不为零。

如果用户在按住左键的同时未将鼠标移到了拖动矩形之外，则返回值为零。

注解

拖动矩形的系统指标是可配置的，允许更大或更小的拖动矩形。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetSystemMetrics](#)

[鼠标输入](#)

[点](#)

[引用](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

EnterReaderModeHelper 函数

项目 • 2023/04/20

启用鼠标中间按钮以切换可滚动窗口上是否存在滚动指南针光标。

语法

C++

```
LONG EnterReaderModeHelper(  
    HANDLE hwnd  
) ;
```

参数

hwnd

为其启用了滚动指南针光标的窗口的句柄。

返回值

如果传递的窗口句柄无效，则计算结果为 FALSE 的值;否则为 TRUE。

注解

此函数未在 SDK 标头中定义，必须由调用方声明。此函数从 user32.dll 导出。

要求

要求	值
最低受支持的客户端	Windows 10
最低受支持的服务器	Windows 10
DLL	user32.dll

另请参阅

GetCapture 函数 (winuser.h)

项目2024/03/04

检索任何捕获鼠标的窗口句柄（如果有）。一次只能有一个窗口捕获鼠标；无论光标是否在其边框内，此窗口都会收到鼠标输入。

语法

C++

```
HWND GetCapture();
```

返回值

类型：HWND

返回值是与当前线程关联的捕获窗口的句柄。如果线程中没有窗口捕获鼠标，则返回值为 NULL。

注解

NULL 返回值表示当前线程尚未捕获鼠标。但是，另一个线程或进程可能捕获了鼠标。

若要获取另一个线程上的捕获窗口的句柄，请使用 [GetGUIThreadInfo](#) 函数。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib

要求	值
DLL	User32.dll
API 集	在 Windows 8) 中引入的 ext-ms-win-ntuser-mouse-l1-1-0 (

请参阅

概念性

[GetGUIThreadInfo](#)

[鼠标输入](#)

引用

[ReleaseCapture](#)

[SetCapture](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

GetDoubleClickTime 函数 (winuser.h)

项目2024/03/04

检索鼠标的当前双击时间。 双击是鼠标按钮的一系列两次单击，第二次单击在第一次单击之后的指定时间内发生。 双击时间是双击的第一次单击和第二次单击之间可能发生的最大毫秒数。 最大双击时间为 5000 毫秒。

语法

C++

```
UINT GetDoubleClickTime();
```

返回值

类型: **UINT**

返回值指定当前的双击时间（以毫秒为单位）。 最大返回值为 5000 毫秒。

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	在 Windows 8 中引入 ext-ms-win-ntuser-mouse-l1-1-0 (

请参阅

概念性

[鼠标输入](#)

引用

[SetDoubleClickTime](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

getMouseMovePointsEx 函数 (winuser.h)

项目2024/03/04

检索最多 64 个鼠标或笔的先前坐标的历史记录。

语法

C++

```
int GetMouseMovePointsEx(
    [in]  UINT          cbSize,
    [in]  LPMOUSEMOVEPOINT lppt,
    [out] LPMOUSEMOVEPOINT lpptBuf,
    [in]  int           nBufPoints,
    [in]  DWORD         resolution
);
```

参数

[in] cbSize

类型: **UINT**

MOUSEMOVEPOINT 结构的大小 (以字节为单位)。

[in] lppt

类型: **LPMOUSEMOVEPOINT**

指向 **MOUSEMOVEPOINT** 结构的指针, 该结构包含有效鼠标坐标 (屏幕坐标)。它还可能包含时间戳。

GetMouseMovePointsEx 函数搜索鼠标坐标历史记录中的点。如果函数找到该点, 它将返回提供点之前的最后一个 *nBufPoint*, 并包括提供的点。

如果应用程序提供时间戳, **GetMouseMovePointsEx** 函数将使用它来区分在不同时间记录的两个等点。

应用程序应使用从 **WM_MOUSEMOVE** 消息接收的鼠标坐标调用此函数, 并将其转换为屏幕坐标。

[out] lppBuf

类型: LPMOUSEMOVEPOINT

指向将接收点的缓冲区的指针。 它的大小至少应为 $cbSize * nBufPoints$ 。

[in] nBufPoints

类型: int

要检索的点数。

[in] resolution

类型: DWORD

所需的分辨率。 此参数的取值可为下列值之一:

 展开表

值	含义
GMMP_USE_DISPLAY_POINTS 1	使用显示分辨率检索点。
GMMP_USE_HIGH_RESOLUTION_POINTS 2	检索高分辨率点。 x 坐标和 y 坐标中的点范围为 0 到 65,535 (0xFFFF)。 这是绝对坐标指向设备 (如绘图平板电脑) 提供的分辨率。

返回值

类型: int

如果函数成功，则返回值为缓冲区中的点数。 否则，该函数返回 -1。 对于扩展的错误信息，应用程序可以调用 [GetLastError](#)。

注解

系统保留最后 64 个鼠标坐标及其时间戳。 如果应用程序向 [GetMouseMovePointsEx](#) 提供鼠标坐标，并且坐标存在于系统的鼠标坐标历史记录中，则函数将从系统的历史记录中检索指定数量的坐标。 还可以提供时间戳，用于区分历史记录中的相同点。

[GetMouseMovePointsEx](#) 函数将返回最终不仅调度到调用线程的点，还会返回其他线程的点。

在以下情况下，**GetMouseMovePointsEx** 可能会失败或返回错误值：

- 如果在 [MOUSEMOVEPOINT](#) 结构中传递负坐标。
- 如果 **GetMouseMovePointsEx** 检索具有负值的坐标。

如果存在多个监视器，则可能会出现这些情况。 若要更正此问题，请首先调用 [GetSystemMetrics](#) 以获取以下值：

- **SM_XVIRTUALSCREEN**,
- **SM_YVIRTUALSCREEN**,
- **SM_CXVIRTUALSCREEN** 和
- **SM_CYVIRTUALSCREEN**。

然后，对于从 **GetMouseMovePointsEx** 返回的每个点，执行以下转换：

```
int nVirtualWidth = GetSystemMetrics(SM_CXVIRTUALSCREEN) ;
int nVirtualHeight = GetSystemMetrics(SM_CYVIRTUALSCREEN) ;
int nVirtualLeft = GetSystemMetrics(SM_XVIRTUALSCREEN) ;
int nVirtualTop = GetSystemMetrics(SM_YVIRTUALSCREEN) ;
int cpt = 0 ;
int mode = GMMP_USE_DISPLAY_POINTS ;

MOUSEMOVEPOINT mp_in ;
MOUSEMOVEPOINT mp_out[64] ;

ZeroMemory(&mp_in, sizeof(mp_in)) ;
mp_in.x = pt.x & 0x0000FFFF ;//Ensure that this number will pass through.
mp_in.y = pt.y & 0x0000FFFF ;
cpt = GetMouseMovePointsEx(&mp_in, &mp_out, 64, mode) ;

for (int i = 0; i < cpt; i++)
{
    switch(mode)
    {
        case GMMP_USE_DISPLAY_POINTS:
            if (mp_out[i].x > 32767)
                mp_out[i].x -= 65536 ;
            if (mp_out[i].y > 32767)
                mp_out[i].y -= 65536 ;
            break ;
        case GMMP_USE_HIGH_RESOLUTION_POINTS:
            mp_out[i].x = ((mp_out[i].x * (nVirtualWidth - 1)) - (nVirtualLeft * 65536)) / nVirtualWidth ;
            mp_out[i].y = ((mp_out[i].y * (nVirtualHeight - 1)) - (nVirtualTop * 65536)) / nVirtualHeight ;
            break ;
    }
}
```

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[MOUSEMOVEPOINT](#)

[鼠标输入](#)

引用

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

mouse_event 函数 (winuser.h)

项目2023/08/27

mouse_event 函数合成鼠标运动和按钮单击。

注意 此函数已被取代。 请改用 SendInput。

语法

C++

```
void mouse_event(
    [in] DWORD      dwFlags,
    [in] DWORD      dx,
    [in] DWORD      dy,
    [in] DWORD      dwData,
    [in] ULONG_PTR  dwExtraInfo
);
```

参数

[in] dwFlags

类型: DWORD

控制鼠标运动和按钮单击的各个方面。 此参数可以是以下值的某些组合。

值	含义
MOUSEEVENTF_ABSOLUTE 0x8000	<i>dx</i> 和 <i>dy</i> 参数包含规范化的绝对坐标。 如果未设置，则这些参数包含相对数据：自上次报告的位置以来的位置变化。 无论哪种类型的鼠标或类似鼠标的设备（如果有）连接到系统，都可以设置或不设置此标志。 有关相对鼠标运动的详细信息，请参阅以下“备注”部分。
MOUSEEVENTF_LEFTDOWN 0x0002	左按钮按下。
MOUSEEVENTF_LEFTUP 0x0004	左按钮已向上。
MOUSEEVENTF_MIDDLEDOWN	中间按钮已关闭。

0x0020	
MOUSEEVENTF_MIDDLEUP 0x0040	中间按钮已向上。
MOUSEEVENTF_MOVE 0x0001	发生了移动。
MOUSEEVENTF_RIGHTDOWN 0x0008	右按钮已关闭。
MOUSEEVENTF_RIGHTUP 0x0010	右侧按钮已向上。
MOUSEEVENTF_WHEEL 0x0800	如果鼠标有滚轮，则滚轮已移动。 移动量在 <i>dwData</i> 中指定
MOUSEEVENTF_XDOWN 0x0080	按下了 X 按钮。
MOUSEEVENTF_XUP 0x0100	已释放 X 按钮。
MOUSEEVENTF_WHEEL 0x0800	滚轮按钮已旋转。
MOUSEEVENTF_HWHEEL 0x01000	滚轮按钮倾斜。

指定鼠标按钮状态的值设置为指示状态的更改，而不是正在进行的条件。例如，如果按下并按住鼠标左键，则 **MOUSEEVENTF_LEFTDOWN** 是在首次按下左按钮时设置的，但不是在后续动作时设置的。同样，仅在首次释放按钮时设置 **MOUSEEVENTF_LEFTUP**。

不能在 *dwFlags* 参数中同时指定 **MOUSEEVENTF_WHEEL** 和 **MOUSEEVENTF_XDOWN** 或 **MOUSEEVENTF_XUP**，因为它们都需要使用 *dwData* 字段。

[in] dx

类型: DWORD

鼠标沿 x 轴的绝对位置或其自上次生成鼠标事件以来的运动量，具体取决于 **MOUSEEVENTF_ABSOLUTE** 的设置。绝对数据指定为鼠标的实际 x 坐标;相对数据指定为移动的米奇数。麦克风是鼠标必须移动才能报告其移动量。

[in] dy

类型: DWORD

鼠标沿 y 轴的绝对位置或其自上次生成鼠标事件以来的运动量，具体取决于 **MOUSEEVENTF_ABSOLUTE** 的设置。 绝对数据指定为鼠标的实际 y 坐标；相对数据指定为移动的米奇数。

[in] dwData

类型：DWORD

如果 *dwFlags* 包含 **MOUSEEVENTF_WHEEL**，则 *dwData* 指定滚轮移动量。 正值表示滚轮向前旋转（远离用户）；负值表示滚轮向后旋转（朝向用户）。 一键式单击定义为 **WHEEL_DELTA**，即 120。

如果 *dwFlags* 包含 **MOUSEEVENTF_HWHEEL**，则 *dwData* 指定滚轮移动量。 正值表示方向盘向右倾斜；负值表示方向盘向左倾斜。

如果 *dwFlags* 包含 **MOUSEEVENTF_XDOWN** 或 **MOUSEEVENTF_XUP**，则 *dwData* 指定按下或释放了哪些 X 按钮。 此值可以是以下标志的任意组合。

如果 *dwFlags* 未 **MOUSEEVENTF_WHEEL**、**MOUSEEVENTF_XDOWN** 或 **MOUSEEVENTF_XUP**，则 *dwData* 应为零。

值	含义
XBUTTON1 0x0001	设置是否按下或释放第一个 X 按钮。
XBUTTON2 0x0002	设置是否按下或释放第二个 X 按钮。

[in] dwExtraInfo

类型：ULONG_PTR

与鼠标事件关联的附加值。 应用程序调用 [GetMessageExtraInfo](#) 以获取此额外信息。

返回值

无

备注

如果鼠标已移动（由 **设置MOUSEEVENTF_MOVE** 指示），*dx* 和 *dy* 将保留有关该动作的信息。 信息指定为绝对或相对整数值。

如果指定了 `MOUSEEVENTF_ABSOLUTE` 值，则 `dx` 和 `dy` 包含介于 0 和 65,535 之间的规范化绝对坐标。事件过程将这些坐标映射到显示图面。坐标 (显示图面左上角的 0, 0) 地图，(65535, 65535) 贴图到右下角。

如果未指定 `MOUSEEVENTF_ABSOLUTE` 值，则 `dx` 和 `dy` 指定从上次生成鼠标事件时起的相对运动 (上次报告的位置)。正值表示鼠标向右移动 (或向下移动)；负值表示鼠标向左移动 (或向上移动)。

相对鼠标运动取决于鼠标速度和加速级别的设置。最终用户在 控制面板 中使用鼠标应用程序设置这些值。应用程序使用 `SystemParametersInfo` 函数获取并设置这些值。

应用加速时，系统会对指定的相对鼠标运动应用两个测试。如果沿 `x` 或 `y` 轴的指定距离大于第一个鼠标阈值，并且鼠标加速级别不为零，则操作系统会将距离加倍。如果沿 `x` 轴或 `y` 轴的指定距离大于第二个鼠标阈值，并且鼠标加速级别等于 2，则操作系统会将应用第一个阈值测试产生的距离加倍。因此，操作系统可以沿 `x` 轴或 `y` 轴将相对指定的鼠标运动乘以最多四倍。

应用加速后，系统会按所需的鼠标速度缩放结果值。鼠标速度的范围从 1 (最慢) 到 20 (最快)，表示指针根据鼠标移动的距离移动多少。默认值为 10，这不会导致对鼠标运动进行其他修改。

`mouse_event` 函数用于由需要执行此操作的应用程序合成鼠标事件。应用程序也使用它，这些应用程序需要从鼠标获取比其位置和按钮状态更多的信息。例如，如果平板电脑制造商想要将基于笔的信息传递给自己的应用程序，它可以编写一个 DLL，该 DLL 直接与平板电脑硬件通信，获取额外信息，并将其保存在队列中。然后，DLL 使用标准按钮和 `x/y` 位置数据调用 `mouse_event`，并在 `dwExtraInfo` 参数中调用一些指向排队的额外信息的指针或索引。当应用程序需要额外信息时，它会使用存储在 `dwExtraInfo` 中的指针或索引调用 DLL，DLL 将返回额外的信息。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetMessageExtraInfo](#)

鼠标输入

其他资源

引用

[SystemParametersInfo](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

releaseCapture 函数 (winuser.h)

项目2023/08/27

从当前线程中的窗口释放鼠标捕获，并还原正常鼠标输入处理。捕获鼠标的窗口接收所有鼠标输入，而不考虑光标的位置，但当光标热点位于另一个线程的窗口中时单击鼠标按钮除外。

语法

C++

```
BOOL ReleaseCapture();
```

返回值

类型: BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 GetLastError。

注解

应用程序在调用 SetCapture 函数后调用此函数。

示例

有关示例，请参阅 [使用鼠标绘制线条](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

Library	User32.lib
DLL	User32.dll
API 集	windows 8) 中引入的 ext-ms-win-ntuser-mouse-l1-1-0 (

请参阅

概念性

[GetCapture](#)

[鼠标输入](#)

引用

[SetCapture](#)

[WM_CAPTURECHANGED](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

setCapture 函数 (winuser.h)

项目2023/08/28

将鼠标捕获设置为属于当前线程的指定窗口。当鼠标悬停在捕获窗口上，或者在鼠标悬停在捕获窗口上且按钮仍然向下的情况下按下鼠标按钮时，SetCapture 将捕获鼠标输入。一次只会有一个窗口捕获鼠标。

如果鼠标光标位于另一个线程创建的窗口上，则仅当鼠标按钮按下时，系统才会将鼠标输入定向到指定的窗口。

语法

C++

```
HWND SetCapture(  
    [in] HWND hWnd  
);
```

参数

[in] hWnd

类型: HWND

当前线程中要捕获鼠标的窗口的句柄。

返回值

类型: HWND

返回值是之前捕获过鼠标的窗口的句柄。如果没有此类窗口，则返回值为 NULL。

注解

只有前台窗口可以捕获鼠标。当后台窗口尝试执行此操作时，窗口仅接收当光标热点位于窗口的可见部分时发生的鼠标事件的消息。此外，即使前台窗口已捕获鼠标，用户仍可以单击另一个窗口，将其带到前台。

当窗口不再需要所有鼠标输入时，创建窗口的线程应调用 [ReleaseCapture](#) 函数来释放鼠标。

此函数不能用于捕获用于另一个进程的鼠标输入。

捕获鼠标时，菜单热键和其他键盘快捷键不起作用。

示例

有关示例，请参阅 [使用鼠标绘制线条](#)。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	windows 8 中引入的 ext-ms-win-ntuser-mouse-l1-1-0 ()

请参阅

概念性

[GetCapture](#)

[鼠标输入](#)

引用

[ReleaseCapture](#)

[WM_CAPTURECHANGED](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

SetDoubleClickTime 函数 (winuser.h)

项目2024/03/04

设置鼠标的双击时间。 双击是鼠标按钮的一系列两次单击，第二次单击在第一次单击之后的指定时间内发生。 双击时间是双击的第一次单击和第二次单击之间可能发生的最大毫秒数。

语法

C++

```
BOOL SetDoubleClickTime(  
    [in] UINT unnamedParam1  
) ;
```

参数

[in] unnamedParam1

类型：UINT

双击的第一次和第二次单击之间可能发生的毫秒数。 如果此参数设置为 0，则系统使用默认的双击时间 500 毫秒。 如果此参数值大于 5000 毫秒，则系统将该值设置为 5000 毫秒。

返回值

类型：BOOL

如果该函数成功，则返回值为非零值。

如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。

注解

SetDoubleClickTime 函数更改系统中所有窗口的双击时间。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-mouse-l1-1-1 ()

请参阅

概念性

[GetDoubleClickTime](#)

[鼠标输入](#)

引用

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

swapMouseButton 函数 (winuser.h)

项目2023/08/28

反转或还原鼠标左键和右键的含义。

语法

C++

```
BOOL SwapMouseButton(  
    [in] BOOL fSwap  
) ;
```

参数

[in] fSwap

类型: BOOL

如果此参数为 TRUE，则左侧按钮生成右按钮消息，右侧按钮生成左按钮消息。如果此参数为 FALSE，则按钮将还原到其原始含义。

返回值

类型: BOOL

如果在调用 函数之前，鼠标按钮的含义被反转，则返回值为非零值。

如果未反转鼠标按钮的含义，则返回值为零。

注解

为方便左手使用鼠标的用户提供按钮交换。 SwapMouseButton 函数通常仅由控制面板调用。虽然应用程序可以自由调用函数，但鼠标是一个共享资源，并且反转其按钮的含义会影响所有应用程序。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[鼠标输入](#)

引用

[SetDoubleClickTime](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TrackMouseEvent 函数 (winuser.h)

项目2023/08/28

当在指定时间内鼠标指针离开窗口或将鼠标悬停在窗口上时，发布消息。

注意_TrackMouseEvent函数调用 TrackMouseEvent (如果存在) ，否则 _TrackMouseEvent模拟 TrackMouseEvent。

语法

C++

```
BOOL TrackMouseEvent(  
    [in, out] LPTRACKMOUSEEVENT lpEventTrack  
>;
```

参数

[in, out] lpEventTrack

类型: LPTRACKMOUSEEVENT

指向包含跟踪信息的 TRACKMOUSEEVENT 结构的指针。

返回值

类型: BOOL

如果函数成功，则返回值为非零。

如果函数失败，则返回值为零。要获得更多的错误信息，请调用 GetLastError。

注解

当鼠标指针在指定矩形内停留一段时间时，它被视为悬停。调用 SystemParametersInfo。和 使用 SPI_GETMOUSEHOVERWIDTH、SPI_GETMOUSEHOVERHEIGHT 和 SPI_GETMOUSEHOVERTIME 的值来检索矩形的大小和时间。

函数可以发布以下消息。

消息	说明
WM_NCMOUSEHOVER	与 WM_MOUSEOVER 的含义相同，只不过这适用于窗口的非工作区。
WM_NCMOUSELEAVE	与 WM_MOUSELEAVE 的含义相同，只是对于窗口的非工作区。
WM_MOUSEOVER	鼠标悬停在窗口的工作区上，在之前调用 TrackMouseEvent 中指定的时间段内。生成了此消息时，悬停跟踪将停止。如果应用程序需要进一步跟踪鼠标悬停行为，则必须再次调用 TrackMouseEvent 。
WM_MOUSELEAVE	鼠标离开之前对 TrackMouseEvent 的调用中指定的窗口的工作区。生成此消息时，会取消 TrackMouseEvent 请求的所有跟踪。如果鼠标需要进一步跟踪鼠标悬停行为，则当鼠标重新进入窗口时，应用程序必须调用 TrackMouseEvent 。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	windows 8) 中引入的 ext-ms-win-ntuser-mouse-l1-1-0 (

请参阅

[概念性](#)

[鼠标输入](#)

[其他资源](#)

[引用](#)

[SystemParametersInfo](#)

[TRACKMOUSEEVENT](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

鼠标输入宏

项目 · 2023/06/13

本节内容

- [GET_APPCOMMAND_LPARAM](#)
- [GET_DEVICE_LPARAM](#)
- [GET_FLAGS_LPARAM](#)
- [GET_KEYSTATE_LPARAM](#)
- [GET_KEYSTATE_WPARAM](#)
- [GET_MOUSEORKEY_LPARAM](#)
- [GET_NCHITTEST_WPARAM](#)
- [GET_WHEEL_DELTA_WPARAM](#)
- [GET_XBUTTON_WPARAM](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GET_APPCOMMAND_LPARAM宏 (winuser.h)

项目2024/03/04

从指定的 LPARAM 值检索应用程序命令。

语法

C++

```
void GET_APPCOMMAND_LPARAM(  
    LPARAM  
);
```

参数

lParam

要转换的值。

返回值

无

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

另请参阅

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获得帮助](#)

GET_DEVICE_LPARAM 宏 (winuser.h)

项目2023/07/01

从指定的 LPARAM 值检索输入设备类型。

语法

C++

```
void GET_DEVICE_LPARAM(  
    lParam  
) ;
```

参数

lParam

要转换的值。

返回值

返回值是表示输入设备类型的高位字的位。 可以是下列值之一。

返回代码/值	说明
FAPPCOMMAND_KEY 0	用户按下一个键。
FAPPCOMMAND_MOUSE 0x8000	用户单击了鼠标按钮。
FAPPCOMMAND_OEM 0x1000	未知的硬件源生成了事件。 它可以是鼠标或键盘事件。

返回值

无

备注

此宏与 [GET_MOUSEORKEY_LPARAM宏](#)相同。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

另请参阅

[GET_MOUSEORKEY_LPARAM宏、鼠标输入](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GET_FLAGS_LPARAM 宏 (winuser.h)

项目2024/03/04

从指定的 LPARAM 值检索某些虚拟密钥的状态。

语法

C++

```
void GET_FLAGS_LPARAM(  
    lParam  
);
```

参数

lParam

要转换的值。

返回值

无

备注

此宏与 [GET_KEYSTATE_LPARAM](#) 宏相同。

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GET_KEYSTATE_LPARAM](#)

鼠标输入

引用

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

GET_KEYSTATE_LPARAM宏 (winuser.h)

项目2023/08/27

从指定的 LPARAM 值检索某些虚拟密钥的状态。

语法

C++

```
void GET_KEYSTATE_LPARAM(  
    LPARAM  
);
```

参数

lParam

要转换的值。

返回值

无

备注

此宏与 [GET_FLAGS_LPARAM](#) 宏相同。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GET_FLAGS_LPARAM](#)

鼠标输入

引用

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GET_KEYSTATE_WPARAM 宏 (winuser.h)

项目2023/08/27

从指定的 WPARAM 值检索某些虚拟密钥的状态。

语法

C++

```
void GET_KEYSTATE_WPARAM(  
    wParam  
) ;
```

参数

wParam

要转换的值。

返回值

无

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

另请参阅

[鼠标输入概述](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GET_NCHITTEST_WPARAM宏 (winuser.h)

项目2024/03/04

从指定的 WPARAM 值检索命中测试值。

语法

C++

```
void GET_NCHITTEST_WPARAM(  
    wParam  
) ;
```

参数

wParam

要转换的值。

返回值

无

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

另请参阅

[鼠标输入概述](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获得帮助](#)

GET_WHEEL_DELTA_WPARAM 宏 (winuser.h)

项目2024/03/04

从指定的 WPARAM 值中检索 wheel-delta 值。

语法

C++

```
void GET_WHEEL_DELTA_WPARAM(  
    wParam  
);
```

参数

wParam

要转换的值。

返回值

无

要求

展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

另请参阅

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获得帮助](#)

GET_XBUTTON_WPARAM宏 (winuser.h)

项目2023/08/27

从指定的 WPARAM 值检索某些按钮的状态。

语法

C++

```
void GET_XBUTTON_WPARAM(  
    wParam  
) ;
```

参数

wParam

要转换的值。

返回值

无

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

另请参阅

[鼠标输入概述](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

鼠标输入通知

项目 · 2023/06/12

本节内容

- WM_CAPTURECHANGED
- WM_LBUTTONDOWNDBLCLK
- WM_LBUTTONDOWN
- WM_LBUTTONUP
- WM_MBUTTONDOWNDBLCLK
- WM_MBUTTONDOWN
- WM_MBUTTONUP
- WM_MOUSEACTIVATE
- WM_MOUSEHOVER
- WM_MOUSEWHEEL
- WM_MOUSELEAVE
- WM_MOUSEMOVE
- WM_MOUSEWHEEL
- WM_NCHITTEST
- WM_NCLBUTTONDOWNDBLCLK
- WM_NCLBUTTONDOWN
- WM_NCLBUTTONUP
- WM_NCMBUTTONDOWNDBLCLK
- WM_NCMBUTTONDOWN
- WM_NCMBUTTONUP
- WM_NCMOUSEHOVER
- WM_NCMOUSELEAVE
- WM_NCMOUSEMOVE
- WM_NCRBUTTONDOWNDBLCLK
- WM_NCRBUTTONDOWN
- WM_NCRBUTTONUP
- WM_NCXBUTTONDOWNDBLCLK
- WM_NCXBUTTONDOWN
- WM_NCXBUTTONUP
- WM_RBUTTONDOWNDBLCLK
- WM_RBUTTONDOWN
- WM_RBUTTONUP
- WM_XBUTTONDOWNDBLCLK
- WM_XBUTTONDOWN

- WM_XBUTTONUP
-

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_CAPTURECHANGED消息

项目 • 2023/06/22

发送到丢失鼠标捕获的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_CAPTURECHANGED 0x0215
```

参数

wParam

未使用此参数。

lParam

获取鼠标捕获的窗口的句柄。

返回值

如果应用程序处理此消息，则应返回零。

注解

窗口接收此消息，即使它调用 [ReleaseCapture](#) 本身。 应用程序不应尝试设置鼠标捕获以响应此消息。

收到此消息时，窗口应根据需要重绘自身，以反映新的鼠标捕获状态。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[ReleaseCapture](#)

[SetCapture](#)

概念性

[鼠标输入](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_LBUTTONDOWNDBLCLK 消息

项目 • 2024/01/10

当光标位于窗口工作区中并且用户双击鼠标左键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_LBUTTONDOWNDBLCLK 0x0203
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

只有具有 CS_DBLCLKS 样式的窗口才能接收 WM_LBUTTONDOWNDBLCLK 消息，每当用户按下、松开和再次按下鼠标左键时，系统都会在系统的双击时间限制内生成消息。双击鼠标左键实际上会生成一个由四条消息组成的序列：[WM_LBUTTONDOWN](#)、[WM_LBUTTONUP](#)、[WM_LBUTTONDOWNDBLCLK](#) 和 [WM_LBUTTONUP](#)。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]

要求	值
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetDoubleClickTime](#)

[SetCapture](#)

[SetDoubleClickTime](#)

[WM_LBUTTONDOWN](#)

[WM_LBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助？

是

否

WM_LBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户按下鼠标左键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_LBUTTONDOWN 0x0201
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

示例

C++

```
LRESULT CALLBACK WndProc(_In_ HWND hWnd, _In_ UINT msg, _In_ WPARAM wParam,
_In_ LPARAM lParam)
{
    POINT pt;

    switch (msg)
    {

        case WM_LBUTTONDOWN:
        {
            pt.x = GET_X_LPARAM(lParam);
            pt.y = GET_Y_LPARAM(lParam);
        }
        break;

        default:
            return DefWindowProc(hWnd, msg, wParam, lParam);
    }
    return 0;
}
```

有关更多示例，请参阅 GitHub 上的 [Windows 经典示例](#)。

备注

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐

标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

若要检测是否已按下 Alt 键，请检查是否是 [GetKeyState](#)，其中 **VK_MENU < 0**。请记住，不要使用 [GetAsyncKeyState](#)。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetKeyState](#)

[SetCapture](#)

[WM_LBUTTONDOWNDBLCLK](#)

[WM_LBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_LBUTTONDOWN 消息

项目 • 2024/01/11

当光标位于窗口工作区中并且用户释放鼠标左键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_LBUTTONDOWN 0x0202
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于 lParam 值的低位短值；y 坐标位于高位短值（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则你可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[WM_LBUTTONDOWNDBLCLK](#)

[WM_LBUTTONDOWN](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_MBUTTONDOWNDBLCLK 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户双击鼠标中键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MBUTTONDOWNDBLCLK 0x0209
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

只有具有 CS_DBLCLKS 样式的窗口才能接收 WM_MBUTTONDOWNDBLCLK 消息，每当用户按下、松开和再次按下鼠标中键时，系统都会在系统的双击时间限制内生成消息。双击鼠标中键实际上会生成四条消息：[WM_MBUTTONDOWN](#)、[WM_MBUTTONUP](#)、[WM_MBUTTONDOWNDBLCLK](#) 和 [WM_MBUTTONUP](#)。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]

要求	值
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetDoubleClickTime](#)

[SetCapture](#)

[SetDoubleClickTime](#)

[WM_MBUTTONDOWN](#)

[WM_MBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助？

是

否

WM_MBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户按下鼠标中键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MBUTTONDOWN 0x0207
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 **MAKEPOINTS** 宏从返回值中获取一个 **POINTS** 结构。还可以使用 **GET_X_LPARAM** 或 **GET_Y_LPARAM** 宏提取 x 或 y 坐标。

① 重要

请勿使用 **LOWORD** 或 **HIGHWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIGHWORD** 会将坐标视为无符号数量。

若要检测是否已按下 Alt 键，请检查是否是 **GetKeyState**，其中 **VK_MENU <= 0**。请记住，不要使用 **GetAsyncKeyState**。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetKeyState](#)

[SetCapture](#)

[WM_MBUTTONDOWNDBLCLK](#)

[WM_MBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_MBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户释放鼠标中键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MBUTTONDOWN 0x0208
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

请注意，当快捷菜单存在（显示）时，坐标是相对于屏幕的，而不是相对于工作区的。因为 [TrackPopupMenu](#) 是一个异步调用，并且 WM_MBUTTONDOWN 通知无指示坐标派生的特殊标志，所以应用程序无法判断 IParam 中包含的 x,y 坐标是相对于屏幕还是工作区。

返回值

如果应用程序处理此消息，它应返回零。

备注

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[WM_MBUTTONDOWNDBLCLK](#)

[WM_MBUTTONDOWN](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_MOUSEACTIVATE 消息

项目 • 2024/01/06

当光标处于非活动窗口中并且用户按下鼠标按钮时发送。仅当子窗口将此消息传递给 [DefWindowProc](#) 函数时，父窗口才会收到此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MOUSEACTIVATE 0x0021
```

参数

wParam

要激活的窗口的顶级父窗口的句柄。

lParam

低序字指定 [DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息后返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

高序字指定当用户按下鼠标按钮时生成的鼠标消息的标识符。鼠标消息将被丢弃或发布到窗口，具体取决于返回值。

返回值

返回值指定是否应当激活窗口，以及是否应当丢弃鼠标消息的标识符。必须是以下值之一。

[展开表](#)

返回代码/值	说明
MA_ACTIVATE 1	激活窗口，并且不丢弃鼠标消息。
MA_ACTIVATEANDEAT 2	激活窗口，并丢弃鼠标消息。
MA_NOACTIVATE 3	不激活窗口，并且不丢弃鼠标消息。

返回代码/值	说明
MA_NOACTIVATEANDEAT	不激活窗口，但丢弃鼠标消息。
4	

注解

在进行任何处理之前，[DefWindowProc](#) 函数会将消息传递给子窗口的父窗口。父窗口确定是否激活子窗口。如果它激活子窗口，父窗口应返回 MA_NOACTIVATE 或 MA_NOACTIVATEANDEAT，以阻止系统进一步处理消息。

要求

[\[+\] 展开表](#)

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[DefWindowProc](#)

[HIWORD](#)

[LOWORD](#)

[WM_NCHITTEST](#)

[Conceptual](#)

[鼠标输入](#)

反馈

此页面是否有帮助?

是

否

WM_MOUSEHOVER 消息

项目 • 2024/01/06

当光标悬停在窗口的工作区上的时间达到上次 [TrackMouseEvent](#) 调用中指定的时间段时，发布到一个窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MOUSEOVER 0x02A1
```

参数

wParam

指示各种虚拟键是否已按下。 此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。 坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。 坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

当生成了 WM_MOUSEHOVER 时，悬停跟踪将停止。 如果应用程序需要进一步跟踪鼠标悬停行为，则必须再次调用 [TrackMouseEvent](#)。

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。 如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。 还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。 具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

要求

[] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[TrackMouseEvent](#)

[TRACKMOUSEEVENT](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_MOUSEWHEEL 消息

项目 • 2024/01/06

在鼠标的水平滚轮发生倾斜或旋转时发送给活动窗口。DefWindowProc 函数将消息传播到窗口的父级。不应在内部转发消息，因为 DefWindowProc 将它向上传播到父链，直到找到处理它的窗口。

窗口通过其 WindowProc 函数接收此消息。

C++

```
#define WM_MOUSEWHEEL 0x020E
```

参数

wParam

高序字指示滚轮旋转的距离，以 WHEEL_DELTA 的倍数或因子表示，设置为 120。正值表示滚轮向右旋转；负值表示滚轮向左旋转。

低序字指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2	按下了第二个 X 按钮。

值	含义
0x0040	

lParam

低序字指定指针相对于屏幕左上角的 x 坐标。

高序字指定指针相对于屏幕左上角的 y 坐标。

返回值

如果应用程序处理此消息，它应返回零。

备注

使用以下代码获取 wParam 参数中的信息。

```
fwKeys = GET_KEYSTATE_WPARAM(wParam);
zDelta = GET_WHEEL_DELTA_WPARAM(wParam);
```

使用以下代码获取水平和垂直位置。

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低序位；y 坐标位于高序位（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

滚轮旋转是 WHEEL_DELTA 的倍数，设置为 120。这是要执行的操作的阈值，应针对每个增量执行一个此类操作（例如，滚动一个增量）。

增量设置为 120，以允许 Microsoft 或其他供应商构建更精细的分辨率滚轮（例如，一个无槽位的自由旋转轮），从而在每次旋转时发送更多消息，但每条消息中的值较小。若要使用此功能，要么添加传入的增量值，直到达到 WHEEL_DELTA（以便在增量旋转中获得相同的响应），要么滚动部分行以响应更频繁的消息。此外，还可以选择滚动粒度并累积增量，直到达到它。

要求

 展开表

要求	值
最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_KEYSTATE_WPARAM](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GET_WHEEL_DELTA_WPARAM](#)

[HIWORD](#)

[LOWORD](#)

[mouse_event](#)

[Conceptual](#)

[鼠标输入](#)

[其他资源](#)

[GetSystemMetrics](#)

[MAKEPOINTS](#)

[POINTS](#)

[SystemParametersInfo](#)

反馈

此页面是否有帮助?

 是

 否

WM_MOUSELEAVE消息

项目 • 2023/06/22

当光标离开之前调用 [TrackMouseEvent](#) 中指定的窗口工作区时，发布到窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MOUSELEAVE 0x02A3
```

参数

wParam

不使用此参数，并且必须为零。

lParam

不使用此参数，并且必须为零。

返回值

如果应用程序处理此消息，则它应返回零。

备注

生成此消息时，[将取消 TrackMouseEvent](#) 请求的所有跟踪。如果鼠标需要进一步跟踪鼠标悬停行为，则当鼠标重新进入其窗口时，应用程序必须调用 [TrackMouseEvent](#)。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[GetCapture](#)

[SetCapture](#)

[TrackMouseEvent](#)

[TRACKMOUSEEVENT](#)

[WM_NCMOUSELEAVE](#)

概念性

[鼠标输入](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

WM_MOUSEMOVE 消息

项目 • 2024/01/06

光标移动时发布到窗口。如果未捕获鼠标，则消息将发布到包含光标的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MOUSEMOVE 0x0200
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。 坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?



WM_MOUSEWHEEL 消息

项目 • 2024/01/06

当旋转鼠标滚轮时，发送到焦点窗口。 [DefWindowProc](#) 函数将消息传播到窗口的父级。不应在内部转发消息，因为 DefWindowProc 将它向上传播到父链，直到找到处理它的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_MOUSEWHEEL 0x020A
```

参数

wParam

高序字指示滚轮旋转的距离，以 WHEEL_DELTA (120) 的倍数或除法表示。 正值表示滚轮向前旋转（远离用户）；负值表示滚轮向后旋转（朝向用户）。

低序字指示各种虚拟键是否已按下。 此参数可使用以下一个或多个值。

 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2	按下了第二个 X 按钮。

值	含义
0x0040	

lParam

低序字指定指针相对于屏幕左上角的 x 坐标。

高序字指定指针相对于屏幕左上角的 y 坐标。

返回值

如果应用程序处理此消息，它应返回零。

备注

使用以下代码获取 wParam 参数中的信息：

```
fwKeys = GET_KEYSTATE_WPARAM(wParam);
zDelta = GET_WHEEL_DELTA_WPARAM(wParam);
```

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

滚轮旋转将是 WHEEL_DELTA 的倍数，设置为 120。这是要执行的操作的阈值，应针对每个增量执行一个此类操作（例如，滚动一个增量）。

增量设置为 120，以允许 Microsoft 或其他供应商构建更精细的分辨率滚轮（一个无槽位的自由旋转轮），从而在每次旋转发送更多消息，但每条消息中的值较小。若要使用此功能，请添加传入的增量值，直到达到 WHEEL_DELTA（以便在增量旋转中获取相同响应），或者滚动部分行以响应更频繁的消息。还可以选择滚动粒度并累积增量，直到达到它。

请注意，MSH_MOUSEWHEEL 没有 fwKey。否则，WM_MOUSEWHEEL 的参数完全相同。

由应用程序将 MSH_MOUSEWHEEL 转发到任何嵌入的对象或控件。应用程序需要将消息发送到活动的嵌入式 OLE 应用程序。应用程序可以选择将其发送到具有焦点的已启用滚轮的控件。如果应用程序确实将消息发送到控件，它可以检查返回值以查看消息是否已处理。如果控件处理消息，则需要返回 TRUE 值。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_KEYSTATE_WPARAM](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GET_WHEEL_DELTA_WPARAM](#)

[HIWORD](#)

[LOWORD](#)

[mouse_event](#)

[Conceptual](#)

[鼠标输入](#)

[其他资源](#)

[GetSystemMetrics](#)

[MAKEPOINTS](#)

[POINTS](#)

[SystemParametersInfo](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

WM_NCHITTEST 消息

项目 • 2024/01/09

发送到窗口以确定窗口的哪个部分对应于特定的屏幕坐标。例如，当光标移动、按下或释放鼠标按钮或响应对 [WindowFromPoint](#) 等函数的调用时，可能会发生这种情况。如果未捕获鼠标，则消息将发送到光标下方的窗口。否则，消息将发送到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCHITTEST 0x0084
```

参数

wParam

未使用此参数。

lParam

低序字指定光标的 x 坐标。坐标相对于屏幕的左上角。

高序字指定光标的 y 坐标。坐标相对于屏幕的左上角。

返回值

[DefWindowProc](#) 函数的返回值是下列值之一，指示光标热点的位置。

展开表

返回代码/值	说明
HTBORDER 18	在没有大小调整边框的窗口边框中。
HTBOTTOM 15	在可调整大小的窗口的下水平边框中（用户可以单击鼠标以垂直调整窗口大小）。
HTBOTTOMLEFT 16	在可调整大小的窗口的边框左下角（用户可以单击鼠标以对角线调整窗口大小）。

返回代码/值	说明
HTBOTTOMRIGHT 17	在可调整大小的窗口的边框右下角（用户可以单击鼠标以对角线调整窗口大小）。
HTCAPTION 2	在标题栏中。
HTCLIENT 1	在工作区中。
HTCLOSE 20	在“关闭”按钮中。
HTERROR -2	在屏幕背景上或窗口之间的分割线上（与 HTNOWHERE 相同，只是 DefWindowProc 函数会生成系统蜂鸣音以指示错误）。
HTGROWBOX 4	在大小框中（与 HTSIZE 相同）。
HTHELP 21	在“帮助”按钮中。
HTHSCROLL 6	在水平滚动条中。
HTLEFT 10	在可调整大小的窗口的左边框中（用户可以单击鼠标以水平调整窗口大小）。
HTMENU 5	在菜单中。
HTMAXBUTTON 9	在“最大化”按钮中。
HTMINBUTTON 8	在“最小化”按钮中。
HTNOWHERE 0	在屏幕背景上，或在窗口之间的分隔线上。
HTREDUCE 8	在“最小化”按钮中。
HTRIGHT 11	在可调整大小的窗口的右左边框中（用户可以单击鼠标以水平调整窗口大小）。
HTSIZE 4	在大小框中（与 HTGROWBOX 相同）。
HTSYSMENU 3	在窗口菜单或子窗口的 关闭 按钮中。

返回代码/值	说明
HTTOP 12	在窗口的上水平边框中。
HTTOPLEFT 13	在窗口边框的左上角。
HTTOPRIGHT 14	在窗口边框的右上角。
HTTRANSPARENT -1	在同一线程当前由另一个窗口覆盖的窗口中（消息将发送到同一线程中的基础窗口，直到其中一个窗口返回不是 HTTRANSPARENT 的代码）。
HTVSCROLL 7	在垂直滚动条中。
HTZOOM 9	在 最大化 按钮中。

备注

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

Windows Vista： 创建包含标准描述文字按钮的自定义帧时，应首先将此消息传递给 [DwmDefWindowProc](#) 函数。这使桌面窗口管理器 (DWM) 可为字幕按钮提供命中测试。如果 [DwmDefWindowProc](#) 不处理消息，则可能需要进一步处理 [WM_NCHITTEST](#)。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助？

 是

 否

WM_NCLBUTTONDOWNDBLCLK 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户双击鼠标左键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCLBUTTONDOWNDBLCLK 0x00A3
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① **重要**

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

默认情况下，[DefWindowProc](#) 函数测试指定的点以获取光标的位置并执行相应的操作。如果合适，[DefWindowProc](#) 会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

窗口不需要具有 [CS_DBLCLKS](#) 样式即可接收 [WM_NCLBUTTONDOWNDBLCLK](#) 消息。

当用户在系统的双击时间限制内按下、松开并再次按下鼠标左键时，系统会生成 [WM_NCLBUTTONDOWNDBLCLK](#) 消息。双击鼠标左键实际上会生成四条消息：

[WM_NCLBUTTONDOWN](#)、[WM_NCLBUTTONUP](#)、[WM_NCLBUTTONDOWNDBLCLK](#) 和 [WM_NCLBUTTONUP](#)。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCLBUTTONDOWN](#)

[WM_NCLBUTTONUP](#)

[WM_SYSCOMMAND](#)

Conceptual

[鼠标输入](#)

[其他资源](#)

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_NCLBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户按下鼠标左键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCLBUTTONDOWN 0x00A1
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

[DefWindowProc](#) 函数测试指定的点以找到光标的位置并执行相应的操作。如果合适，[DefWindowProc](#) 会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

ⓘ 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCLBUTTONDOWNDBLCLK](#)

[WM_NCLBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

[其他资源](#)

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

是

否

WM_NCLBUTTONUP 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户释放鼠标左键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCLBUTTONUP 0x00A2
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

[DefWindowProc](#) 函数测试指定的点以获取光标的位置并执行相应的操作。如果合适，[DefWindowProc](#) 会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

ⓘ 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

如果这样做合适，系统会将 **WM_SYSCOMMAND** 消息发送到窗口。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCLBUTTONDOWNDBLCLK](#)

[WM_NCLBUTTONDOWN](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

[其他资源](#)

[MAKEPOINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_NCMBUTTONONDBLCLK 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户双击鼠标中键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCMBUTTONONDBLCLK 0x00A9
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

窗口不需要具有 CS_DBLCLKS 样式即可接收 WM_NCMBUTTONONDBLCLK 消息。

当用户在系统的双击时间限制内按下、松开并再次按下鼠标中键时，系统会生成 WM_NCMBUTTONONDBLCLK 消息。双击鼠标中键实际上会生成四条消息：

[WM_NCMBUTTONDOWN](#)、[WM_NCMBUTTONUP](#)、[WM_NCMBUTTONONDBLCLK](#) 和 [WM_NCMBUTTONUP](#)。

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

如果这样做合适，系统会将 **WM_SYSCOMMAND** 消息发送到窗口。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCMBUTTONDOWN](#)

[WM_NCMBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

[其他资源](#)

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

[!\[\]\(66702b83b114fb9ed7744ca204f8a179_img.jpg\) 是](#)

[!\[\]\(c1c4a296a52af3b33016b9befdd94f62_img.jpg\) 否](#)

WM_NCMBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户释放鼠标中键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCMBUTTONDOWN 0x00A7
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① **重要**

请勿使用 `LOWORD` 或 `HIWORD` 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，`LOWORD` 和 `HIWORD` 会将坐标视为无符号数量。

如果这样做合适，系统会将 `WM_SYSCOMMAND` 消息发送到窗口。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCMBUTTONONDBLCLK](#)

[WM_NCMBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

是

否

WM_NCMBUTTONUP 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户释放鼠标中键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCMBUTTONUP 0x00A8
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① 重要

请勿使用 `LOWORD` 或 `HIWORD` 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，`LOWORD` 和 `HIWORD` 会将坐标视为无符号数量。

如果这样做合适，系统会将 `WM_SYSCOMMAND` 消息发送到窗口。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCMBUTTONDBLCLK](#)

[WM_NCMBUTTONDOWN](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

是

否

WM_NCMOUSEHOVER 消息

项目 • 2024/01/06

当光标悬停在窗口的非工作区上的时间达到上次 [TrackMouseEvent](#) 调用中指定的时间段时，发布到一个窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCMOUSEHOVER 0x02A0
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

生成了此消息时，悬停跟踪将停止。如果应用程序需要进一步跟踪鼠标悬停行为，则必须再次调用 [TrackMouseEvent](#)。

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

ⓘ 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[TrackMouseEvent](#)

[TRACKMOUSEEVENT](#)

[WM_NCHITTEST](#)

[WM_MOUSEHOVER](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

是

否

WM_NCMOUSELEAVE消息

项目 • 2023/06/22

当光标离开之前调用 [TrackMouseEvent](#) 中指定的窗口的非工作区时，将发布到窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCMOUSELEAVE 0x02A2
```

参数

wParam

不使用此参数，并且必须为零。

lParam

不使用此参数，并且必须为零。

返回值

如果应用程序处理此消息，则它应返回零。

备注

生成此消息时，[将取消 TrackMouseEvent](#) 请求的所有跟踪。如果鼠标需要进一步跟踪鼠标悬停行为，则当鼠标重新进入其窗口时，应用程序必须调用 [TrackMouseEvent](#)。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[TrackMouseEvent](#)

[TRACKMOUSEEVENT](#)

[WM_SYSCOMMAND](#)

[WM_MOUSELEAVE](#)

概念性

[鼠标输入](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_NCMOUSEMOVE 消息

项目 • 2024/01/06

当光标在窗口的非工作区内移动时发布到窗口。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCMOUSEMOVE 0x00A0
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

如果这样做合适，系统会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

ⓘ 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_NCRBUTTONDOWNDBLCLK 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户双击鼠标右键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCRBUTTONDOWNDBLCLK 0x00A6
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

窗口不需要具有 [CS_DBLCLKS](#) 样式即可接收 [WM_NCRBUTTONDOWNDBLCLK](#) 消息。

当用户在系统的双击时间限制内按下、松开并再次按下鼠标右键时，系统会生成 [WM_NCRBUTTONDOWNDBLCLK](#) 消息。双击鼠标右键实际上会生成四条消息：[WM_NCRBUTTONDOWN](#)、[WM_NCRBUTTONUP](#)、[WM_NCRBUTTONDOWNDBLCLK](#) 和 [WM_NCRBUTTONUP](#)。

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

如果这样做合适，系统会将 **WM_SYSCOMMAND** 消息发送到窗口。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCRBUTTONDOWN](#)

[WM_NCRBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

[其他资源](#)

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

[!\[\]\(137489dece12ea4ee5706e9783120d6c_img.jpg\) 是](#)

[!\[\]\(9ec155b876a92e7be964b354fb8058aa_img.jpg\) 否](#)

WM_NCRBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户按下鼠标右键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCRBUTTONDOWN 0x00A4
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① **重要**

请勿使用 `LOWORD` 或 `HIWORD` 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，`LOWORD` 和 `HIWORD` 会将坐标视为无符号数量。

如果这样做合适，系统会将 `WM_SYSCOMMAND` 消息发送到窗口。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCRBUTTONDOWNDBLCLK](#)

[WM_NCRBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

是

否

WM_NCRBUTTONUP 消息

项目 • 2023/06/22

当光标位于窗口非工作区中并且用户释放鼠标右键时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCRBUTTONUP 0x00A5
```

参数

wParam

[DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

lParam

包含光标的 x 和 y 坐标的 [POINTS](#) 结构。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，则它应返回零。

备注

还可以使用 [GET_X_LPARAM](#) 和 [GET_Y_LPARAM](#) 宏从 *lParam* 中提取 x 坐标和 y 坐标的值。

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① **重要**

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

如果这样做合适，系统会将 **WM_SYSCOMMAND** 消息发送到窗口。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCRBUTTONDOWNDBLCLK](#)

[WM_NCRBUTTONDOWN](#)

[WM_SYSCOMMAND](#)

概念性

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_NCXBUTTONONDBLCLK 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户双击第一个或第二个 X 按钮时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCXBUTTONONDBLCLK 0x00AD
```

参数

wParam

低序字指定 [DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。

高阶字指示双击了哪个按钮。可以是下列值之一。

展开表

值	含义
XBUTTON1 0x0001	双击第一个 X 按钮。
XBUTTON2 0x0002	双击第二个 X 按钮。

lParam

指向包含光标的 x 和 y 坐标的 [POINTS](#) 结构的指针。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，则应返回 [TRUE](#)。有关处理此返回值的详细信息，请参阅“[备注](#)”部分。

备注

使用以下代码获取 wParam 参数中的信息。

```
nHitTest = GET_NCHITTEST_WPARAM(wParam);  
fwButton = GET_XBUTTON_WPARAM(wParam);
```

还可以使用以下代码从 lParam 获取 x 和 y 坐标：

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

默认情况下，[DefWindowProc](#) 函数测试指定的点以获取光标的位置并执行相应的操作。如果合适，它会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

窗口不需要具有 **CS_DBLCLKS** 样式即可接收 **WM_NCXBUTTONDDBLCLK** 消息。当用户在系统的双击时间限制内按下、松开，然后再次按下 X 按钮时，系统会生成 **WM_NCXBUTTONDDBLCLK** 消息。双击其中一个按钮实际上会生成四条消息：**WM_NCXBUTTONDOWN**、**WM_NCXBUTTONUP**、**WM_NCXBUTTONDDBLCLK**，以及再次生成 **WM_NCXBUTTONUP**。

与 [WM_NCLBUTTONDOWNDBLCLK](#)、[WM_NCMBUTTONDBLCLK](#) 和 [WM_NCRBUTTONDOWNDBLCLK](#) 消息不同，如果应用程序处理此消息，则应从此消息返回 **TRUE**。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 [DefWindowProc](#) 来处理它。

要求

 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]

要求	值
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCXBUTTONDOWN](#)

[WM_NCXBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助？

 是

 否

WM_NCXBUTTONDOWM 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户按下第一个或第二个 X 按钮时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCXBUTTONDOWM 0x00AB
```

参数

wParam

低序字指定 [DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅“[WM_NCHITTEST](#)”。高序字指示按下了哪个按钮。可以是下列值之一。

[+] 展开表

值	含义
XBUTTON1 0x0001	按下了第一个 X 按钮。
XBUTTON2 0x0002	按下了第二个 X 按钮。

lParam

指向包含光标的 x 和 y 坐标的 [POINTS](#) 结构的指针。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，则应返回 [TRUE](#)。有关处理此返回值的详细信息，请参阅“[备注](#)”部分。

备注

使用以下代码获取 wParam 参数中的信息。

```
nHitTest = GET_NCHITTEST_WPARAM(wParam);  
fwButton = GET_XBUTTON_WPARAM(wParam);
```

还可以使用以下代码从 lParam 获取 x 和 y 坐标：

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

默认情况下，[DefWindowProc](#) 函数测试指定的点以获取光标的位置并执行相应的操作。如果合适，它会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

与 [WM_NCLBUTTONDOWN](#)、[WM_NCMBUTTONDOWN](#) 和 [WM_NCRBUTTONDOWN](#) 消息不同，如果应用程序处理此消息，则应从此消息返回 TRUE。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 [DefWindowProc](#) 来处理它。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCXBUTTONONDBLCLK](#)

[WM_NCXBUTTONUP](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_NCXBUTTONUP 消息

项目 • 2024/01/06

当光标位于窗口非工作区中并且用户释放第一个或第二个 X 按钮时发布。此消息将发布到包含光标的窗口。如果窗口捕获了鼠标，则不会发布此消息。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_NCXBUTTONUP 0x00AC
```

参数

wParam

低序字指定 [DefWindowProc](#) 函数在处理 [WM_NCHITTEST](#) 消息时返回的命中测试值。有关命中测试值的列表，请参阅 [WM_NCHITTEST](#)。

高序字指示释放了哪个按钮。可以是下列值之一。

 展开表

值	含义
XBUTTON1 0x0001	释放了第一个 X 按钮。
XBUTTON2 0x0002	释放了第二个 X 按钮。

lParam

指向包含光标的 x 和 y 坐标的 [POINTS](#) 结构的指针。坐标相对于屏幕的左上角。

返回值

如果应用程序处理此消息，则应返回 [TRUE](#)。有关处理此返回值的详细信息，请参阅“备注”部分。

注解

使用以下代码获取 wParam 参数中的信息。

```
nHitTest = GET_NCHITTEST_WPARAM(wParam);  
fwButton = GET_XBUTTON_WPARAM(wParam);
```

还可以使用以下代码从 lParam 获取 x 和 y 坐标：

```
xPos = GET_X_LPARAM(lParam);  
yPos = GET_Y_LPARAM(lParam);
```

① 重要

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

默认情况下，[DefWindowProc](#) 函数测试指定的点以获取光标的位置并执行相应的操作。如果合适，它会将 [WM_SYSCOMMAND](#) 消息发送到窗口。

与 [WM_NCLBUTTONDOWN](#)、[WM_NCMBUTTONDOWN](#) 和 [WM_NCRBUTTONDOWN](#) 消息不同，如果应用程序处理此消息，则应从此消息返回 TRUE。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 [DefWindowProc](#) 来处理它。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[WM_NCHITTEST](#)

[WM_NCXBUTTONONDBLCLK](#)

[WM_NCXBUTTONONDOWWN](#)

[WM_SYSCOMMAND](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_RBUTTONDOWNDBLCLK 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户双击鼠标右键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_RBUTTONDOWNDBLCLK 0x0206
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

只有具有 CS_DBLCLKS 样式的窗口才能接收 WM_RBUTTONDOWNDBLCLK 消息，每当用户在系统的双击时间限制内按下、松开并再次按下鼠标右键时，系统都会生成此消息。双击鼠标右键实际上会生成四条消息：[WM_RBUTTONDOWN](#)、[WM_RBUTTONUP](#)、[WM_RBUTTONDOWNDBLCLK](#) 和 [WM_RBUTTONUP](#)。

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]

要求	值
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetDoubleClickTime](#)

[SetCapture](#)

[SetDoubleClickTime](#)

[WM_RBUTTONDOWN](#)

[WM_RBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

是

否

WM_RBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户按下鼠标右键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_RBUTTONDOWN 0x0204
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

注解

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 **MAKEPOINTS** 宏从返回值中获取一个 **POINTS** 结构。还可以使用 **GET_X_LPARAM** 或 **GET_Y_LPARAM** 宏提取 x 或 y 坐标。

① 重要

请勿使用 **LOWORD** 或 **HIGHWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIGHWORD** 会将坐标视为无符号数量。

若要检测是否已按下 Alt 键，请检查是否是 **GetKeyState**，其中 **VK_MENU <= 0**。请记住，不要使用 **GetAsyncKeyState**。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetKeyState](#)

[SetCapture](#)

[WM_RBUTTONDOWNDBLCLK](#)

[WM_RBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_RBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户释放鼠标右键时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_RBUTTONDOWN 0x0205
```

参数

wParam

指示各种虚拟键是否已按下。此参数可使用以下一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，它应返回零。

备注

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① 重要

请勿使用 [LOWORD](#) 或 [HIWORD](#) 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，[LOWORD](#) 和 [HIWORD](#) 会将坐标视为无符号数量。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_X_LPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[WM_RBUTTONDOWNDBLCLK](#)

[WM_RBUTTONDOWN](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

 是

 否

WM_XBUTTONDOWNDBLCLK 消息

项目 • 2023/06/22

当光标位于窗口客户端区中并且用户双击第一个或第二个 X 按钮时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_XBUTTONDOWNDBLCLK 0x020D
```

参数

wParam

低序字指示各种虚拟键是否已按下。它可以是下面的一个或多个值。

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

高阶字指示双击了哪个按钮。可以是下列值之一。

值	含义

值	含义
XBUTTON1 0x0001	双击第一个 X 按钮。
XBUTTON2 0x0002	双击第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，则应返回 TRUE。有关处理此返回值的详细信息，请参阅“备注”部分。

备注

使用以下代码获取 wParam 参数中的信息：

```
fwKeys = GET_KEYSTATE_WPARAM (wParam);
fwButton = GET_XBUTTON_WPARAM (wParam);
```

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低序位；y 坐标位于高序位（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。如果返回值被赋给一个变量，则你可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。你还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

 **重要**

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

只有具有 **CS_DBLCLKS** 样式的窗口才能接收 **WM_XBUTTONDOWNDBLCLK** 消息，每当用户按下、松开和再次按下鼠标左键时，系统都会在系统的双击时间限制内生成消息。双击其中一个按钮实际上会生成四条消息：**WM_XBUTTONDOWN**、**WM_XBUTTONUP**、**WM_XBUTTONDOWNDBLCLK**，以及再次生成 **WM_XBUTTONUP**。

与 **WM_LBUTTONDOWNDBLCLK**、**WM_MBUTTONDOWNDBLCLK** 和 **WM_RBUTTONDOWNDBLCLK** 消息不同，如果应用程序处理此消息，则应从此消息返回 **TRUE**。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 **DefWindowProc** 来处理它。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[DefWindowProc](#)

[GET_KEYSTATE_WPARAM](#)

[GET_X_LPARAM](#)

[GET_XBUTTON_WPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[GetDoubleClickTime](#)

[SetDoubleClickTime](#)

[WM_XBUTTONDOWN](#)

[WM_XBUTTONUP](#)

概念性

[鼠标输入](#)

其他资源

[MAKEPOINTS](#)

[POINTS](#)

反馈

此页面是否有帮助?

[!\[\]\(b7a64224bcc1f71e35b9c6b514b1cb66_img.jpg\) 是](#)

[!\[\]\(d2fe8796ff373382054c5960b8fc19c9_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

WM_XBUTTONDOWN 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户按下第一个或第二个 X 按钮时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_XBUTTONDOWN 0x020B
```

参数

wParam

低序字指示各种虚拟键是否已按下。它可以是下面的一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

高序字指示单击了哪个按钮。可以是下列值之一。

值	含义
XBUTTON1 0x0001	单击了第一个 X 按钮。
XBUTTON2 0x0002	单击了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。 坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。 坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，则应返回 TRUE。 有关处理此返回值的详细信息，请参阅“备注”部分。

备注

使用以下代码获取 wParam 参数中的信息：

```
fwKeys = GET_KEYSTATE_WPARAM (wParam);
fwButton = GET_XBUTTON_WPARAM (wParam);
```

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。 如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。 还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① **重要**

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

与 [WM_LBUTTONDOWN](#)、[WM_MBUTTONDOWN](#) 和 [WM_RBUTTONDOWN](#) 消息不同，如果应用程序处理此消息，则应从此消息返回 TRUE。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 [DefWindowProc](#) 来处理它。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_KEYSTATE_WPARAM](#)

[GET_X_LPARAM](#)

[GET_XBUTTON_WPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[WM_XBUTTONDOWNBLCLK](#)

[WM_XBUTTONUP](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

MAKEPOINTS

POINTS

反馈

此页面是否有帮助?

 是

 否

WM_XBUTTONUP 消息

项目 • 2024/01/06

当光标位于窗口工作区中并且用户释放第一个或第二个 X 按钮时发布。如果未捕获鼠标，则消息将发布到光标下方的窗口。否则，消息将发布到捕获了鼠标的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_XBUTTONUP 0x020C
```

参数

wParam

低序字指示各种虚拟键是否已按下。它可以是下面的一个或多个值。

[+] 展开表

值	含义
MK_CONTROL 0x0008	按下了 CTRL 键。
MK_LBUTTON 0x0001	按下了鼠标左键。
MK_MBUTTON 0x0010	按下了鼠标中键。
MK_RBUTTON 0x0002	按下了鼠标右键。
MK_SHIFT 0x0004	按下了 SHIFT 键。
MK_XBUTTON1 0x0020	按下了第一个 X 按钮。
MK_XBUTTON2 0x0040	按下了第二个 X 按钮。

高序字指示释放了哪个按钮。可以为下列值之一：

值	含义
XBUTTON1 0x0001	释放了第一个 X 按钮。
XBUTTON2 0x0002	释放了第二个 X 按钮。

lParam

低序字指定光标的 x 坐标。 坐标相对于工作区的左上角。

高序字指定光标的 y 坐标。 坐标相对于工作区的左上角。

返回值

如果应用程序处理此消息，则应返回 TRUE。 有关处理此返回值的详细信息，请参阅“备注”部分。

备注

使用以下代码获取 wParam 参数中的信息：

```
fwKeys = GET_KEYSTATE_WPARAM (wParam);
fwButton = GET_XBUTTON_WPARAM (wParam);
```

使用以下代码获取水平和垂直位置：

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

如上所述，x 坐标位于返回值的低位**短值**；y 坐标位于高位**短值**（两者都表示有符号值，因为它们在具有多个监视器的系统上可以取负值）。 如果返回值被赋给一个变量，则可以使用 [MAKEPOINTS](#) 宏从返回值中获取一个 [POINTS](#) 结构。 还可以使用 [GET_X_LPARAM](#) 或 [GET_Y_LPARAM](#) 宏提取 x 或 y 坐标。

① **重要**

请勿使用 **LOWORD** 或 **HIWORD** 宏提取光标位置的 x 和 y 坐标，因为这些宏在具有多个监视器的系统上会返回不正确的结果。具有多个监视器的系统可以具有负 x 坐标和 y 坐标，**LOWORD** 和 **HIWORD** 会将坐标视为无符号数量。

与 [WM_LBUTTONDOWN](#)、[WM_MBUTTONDOWN](#) 和 [WM_RBUTTONDOWN](#) 消息不同，如果应用程序处理此消息，则应从此消息返回 TRUE。这样做将允许在早于 Windows 2000 的 Windows 系统上模拟此消息的软件确定窗口过程是处理该消息还是调用 [DefWindowProc](#) 来处理它。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
Header	Winuser.h (包括 Windowsx.h)

另请参阅

引用

[GET_KEYSTATE_WPARAM](#)

[GET_X_LPARAM](#)

[GET_XBUTTON_WPARAM](#)

[GET_Y_LPARAM](#)

[GetCapture](#)

[SetCapture](#)

[WM_XBUTTONDOWNDBLCLK](#)

[WM_XBUTTONDOWN](#)

[Conceptual](#)

[鼠标输入](#)

其他资源

MAKEPOINTS

POINTS

反馈

此页面是否有帮助?

 是

 否

鼠标输入结构

项目 · 2024/02/24

本节内容

- [MOUSEMOVEPOINT](#)
- [TRACKMOUSEEVENT](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

MOUSEMOVEPOINT 结构 (winuser.h)

项目2024/03/04

包含有关鼠标在屏幕坐标中的位置的信息。

语法

C++

```
typedef struct tagMOUSEMOVEPOINT {
    int          x;
    int          y;
    DWORD        time;
    ULONG_PTR    dwExtraInfo;
} MOUSEMOVEPOINT, *PMOUSEMOVEPOINT, *LPMOUSEMOVEPOINT;
```

成员

x

类型: int

鼠标的 x 坐标。

y

类型: int

鼠标的 y 坐标。

time

类型: DWORD

鼠标坐标的时间戳。

dwExtraInfo

类型: ULONG_PTR

与此坐标关联的其他信息。

要求

要求	值
最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetMouseMovePointsEx](#)

[鼠标输入](#)

引用

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

TRACKMOUSEEVENT 结构 (winuser.h)

项目2023/08/28

由 TrackMouseEvent 函数用来跟踪在指定的时间范围内，鼠标指针何时离开窗口或鼠标悬停在窗口上。

语法

C++

```
typedef struct tagTRACKMOUSEEVENT {
    DWORD cbSize;
    DWORD dwFlags;
    HWND hwndTrack;
    DWORD dwHoverTime;
} TRACKMOUSEEVENT, *LPTRACKMOUSEEVENT;
```

成员

cbSize

类型: DWORD

TRACKMOUSEEVENT 结构的大小 (以字节为单位)。

dwFlags

类型: DWORD

请求的服务。此成员可以是以下值的组合。

Value	含义
TME_CANCEL 0x80000000	调用方希望取消先前的跟踪请求。调用方还应指定要取消的跟踪类型。例如，若要取消悬停跟踪，调用方必须传递 TME_CANCEL 和 TME_HOVER 标志。
TME_HOVER 0x00000001	调用方需要悬停通知。通知作为 WM_MOUSEHOVER 消息传递。 如果调用方请求悬停跟踪，而悬停跟踪已处于活动状态，则将重置悬停计时器。 如果鼠标指针不在指定的窗口或区域上，则忽略此标志。

TME_LEAVE 0x00000002	调用方想要离开通知。 通知作为 WM_MOUSELEAVE 消息传递。 如果鼠标未在指定的窗口或区域上，则会立即生成离开通知，并且不会执行进一步的跟踪。
TME_NONCLIENT 0x00000010	调用方希望悬停并保留非工作区的通知。 通知以 WM_NCMOUSEHOVER 和 WM_NCMOUSELEAVE 消息的形式传递。
TME_QUERY 0x40000000	函数填充结构，而不是将其视为跟踪请求。 结构已填充，如果结构已传递到 TrackMouseEvent ，它将生成当前跟踪。 唯一的异常是，如果在原始 TrackMouseEvent 请求期间指定了 HOVER_DEFAULT ，则返回的悬停超时始终是实际超时，而不是 HOVER_DEFAULT 。

`hwndTrack`

类型: [HWND](#)

要跟踪的窗口的句柄。

`dwHoverTime`

类型: [DWORD](#)

如果在 `dwFlags` (以毫秒为单位) 指定了 [TME_HOVER](#)，则悬停超时 (。 可以 [HOVER_DEFAULT](#)，这意味着使用系统默认悬停超时。

注解

系统默认悬停超时最初是菜单下拉列表时间，即 400 毫秒。 可以调用 [SystemParametersInfo](#) 并使用 [SPI_GETMOUSEHOVERTIME](#) 检索默认悬停超时。

系统默认悬停矩形与双击矩形相同。 可以调用 [SystemParametersInfo](#) 并使用 [SPI_GETMOUSEHOVERWIDTH](#) 和 [SPI_GETMOUSEHOVERHEIGHT](#) 检索鼠标指针必须保留的矩形的大小，以便 [TrackMouseEvent](#) 生成 [WM_MOUSEHOVER](#) 消息。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

另请参阅

鼠标输入

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

原始输入

项目 · 2023/06/12

本部分介绍系统如何向应用程序提供原始输入，以及应用程序如何接收和处理该输入。原始输入有时称为泛型输入。

本节内容

名称	说明
关于原始输入	讨论来自游戏杆、触摸屏和麦克风等设备的用户输入。
使用原始输入	为与原始输入相关的任务提供示例代码。
原始输入参考	包含 API 引用。

函数

名称	说明
DefRawInputProc	调用默认原始输入过程，为应用程序未处理的任何原始输入消息提供默认处理。此函数可确保处理每条消息。使用窗口过程接收的相同参数调用 DefRawInputProc 。
GetRawInputBuffer	执行原始输入数据的缓冲读取。
GetRawInputData	从指定设备获取原始输入。
GetRawInputDeviceInfo	获取有关原始输入设备的信息。
GetRawInputDeviceList	枚举附加到系统的原始输入设备。
GetRegisteredRawInputDevices	获取有关当前应用程序的原始输入设备的信息。
RegisterRawInputDevices	注册提供原始输入数据的设备。

宏

名称	说明
GET_RAWINPUT_CODE_WPARAM	在 WM_INPUT 中从 <i>wParam</i> 获取输入代码。
NEXTRAWINPUTBLOCK	获取下一个结构在 RAWINPUT 结构数组中的位置。

通知

名称	说明
WM_INPUT	发送到正在获取原始输入的窗口。
WM_INPUT_DEVICE_CHANGE	发送到注册以接收原始输入的窗口。

结构

名称	说明
RAWHID	描述来自人机接口设备 (HID) 的原始输入的格式。
RAWINPUT	包含来自设备的原始输入。
RAWINPUTDEVICE	定义原始输入设备的信息。
RAWINPUTDEVICELIST	包含有关原始输入设备的信息。
RAWINPUTHEADER	包含属于原始输入数据的标头信息。
RAWKEYBOARD	包含有关键盘状态的信息。
RAWMOUSE	包含有关鼠标状态的信息。
RID_DEVICE_INFO	定义来自任何设备的原始输入数据。
RID_DEVICE_INFO_HID	定义来自指定 HID 的原始输入数据。
RID_DEVICE_INFO_KEYBOARD	定义来自指定键盘的原始输入数据。
RID_DEVICE_INFO_MOUSE	定义来自指定鼠标的原始输入数据。

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获取帮助

原始输入概述

项目 • 2023/06/12

传统键盘和鼠标旁边有许多用户输入设备。例如，用户输入可以来自游戏杆、触摸屏、麦克风或其他设备，这些设备在用户输入方面具有极大的灵活性。这些设备统称为人机接口设备 (HID)。原始输入 API 为应用程序提供了一种稳定且可靠的方法，用于接受来自任何 HID（包括键盘和鼠标）的原始输入。

本部分涵盖了以下主题：

- 原始输入模型
- 原始输入的注册
- 读取原始输入

原始输入模型

以前，键盘和鼠标通常生成输入数据。系统以消除原始信息特定于设备的详细信息的方式解释来自这些设备的数据。例如，键盘生成特定于设备的扫描代码，但系统为应用程序提供虚拟键代码。除了隐藏原始输入的详细信息外，窗口管理器还不支持所有新的 HID。若要从不受支持的 HID 获取输入，应用程序必须执行许多操作：打开设备、管理共享模式、定期读取设备或设置 I/O 完成端口等。开发原始输入模型和关联的 API 允许从所有输入设备（包括键盘和鼠标）轻松访问原始输入。

原始输入模型不同于键盘和鼠标的原始 Windows 输入模型。在原始输入模型中，应用程序以消息的形式接收与设备无关的输入，这些消息将发送到或发布到其窗口，例如 **WM_CHAR**、**WM_MOUSEMOVE** 和 **WM_APPCOMMAND**。相比之下，对于原始输入，应用程序必须注册它想要从中获取数据的设备。此外，应用程序通过 **WM_INPUT** 消息获取原始输入。

原始输入模型有几个优点：

- 应用程序不必检测或打开输入设备。
- 应用程序直接从设备获取数据，并根据需要处理数据。
- 应用程序可以区分输入的源，即使它来自同一类型的设备。例如，两个鼠标设备。
- 应用程序通过指定设备集合中的数据或仅指定特定设备类型的数据来管理数据流量。
- HID 设备可在市场中可用时使用，而无需等待新消息类型或更新的 OS 在 **WM_APPCOMMAND** 中拥有新命令。

请注意，**WM_APPCOMMAND** 确实为某些 HID 设备提供。但是，**WM_APPCOMMAND** 是与设备无关的更高级别的输入事件，而 **WM_INPUT** 发送特定于

设备的原始低级别数据。

原始输入的注册

默认情况下，没有应用程序接收原始输入。 若要从设备接收原始输入，应用程序必须注册该设备。

为了注册设备，应用程序首先创建一个 [RAWINPUTDEVICE](#) 结构的数组，这些结构为所需的设备指定 [顶级集合 \(TLC\)](#)。 TLC 由[“使用情况页”](#)定义，(设备) 类 (设备)。 例如，若要获取键盘 TLC，请设置 UsagePage = 0x01 和 UsageID = 0x06。 应用程序调用 [RegisterRawInputDevices](#) 来注册设备。

请注意，应用程序可以注册当前未附加到系统的设备。附加此设备后，Windows 管理器将自动将原始输入发送到应用程序。 若要获取系统上的原始输入设备列表，应用程序会调用 [GetRawInputDeviceList](#)。 应用程序使用此调用中的 *hDevice* 调用 [GetRawInputDeviceInfo](#) 以获取设备信息。

通过 [RAWINPUTDEVICE](#) 的 *dwFlags* 成员，应用程序可以选择要侦听的设备以及要忽略的设备。例如，应用程序可以请求所有电话设备（应答机除外）的输入。有关示例代码，请参阅 [注册原始输入](#)。

请注意，鼠标和键盘也是 HID，因此来自它们的数据可以通过 HID 消息 [WM_INPUT](#) 和来自传统消息。应用程序可以通过在 [RAWINPUTDEVICE](#) 中正确选择标志来选择任一方方法。

若要获取应用程序的注册状态，请随时调用 [GetRegisteredRawInputDevices](#)。

读取原始输入

应用程序从任何 HID 接收原始输入，其 [顶级集合 \(TLC\)](#) 与注册中的 TLC 匹配。当应用程序收到原始输入时，其消息队列将获取 [WM_INPUT](#) 消息，[并且队列](#) 状态标志 [QS_RAWINPUT](#) 设置为 ([QS_INPUT](#) 还包括此标志)。应用程序在前台和后台时可以接收数据。

有两种方法可以读取原始数据：无缓冲 (或标准) 方法和缓冲方法。无缓冲区方法一次获取一个 [RAWINPUT](#) 结构的原始数据，并且足以用于许多 HID。在这里，应用程序调用 [GetMessage](#) 以获取 [WM_INPUT](#) 消息。然后，应用程序使用包含在 [WM_INPUT](#) 中的 [RAWINPUT](#) 句柄调用 [GetRawInputData](#)。有关示例，请参阅 [执行原始输入的标准读取](#)。

相比之下，缓冲方法一次获取 [RAWINPUT](#) 结构的数组。这是为可以生成大量原始输入的设备提供的。在此方法中，应用程序调用 [GetRawInputBuffer](#) 来获取 [RAWINPUT](#) 结构

的数组。请注意，**NEXTRAWINPUTBLOCK** 宏用于遍历 **RAWINPUT** 结构的数组。有关示例，请参阅 [执行原始输入的缓冲读取](#)。

若要解释原始输入，需要有关 HID 的详细信息。应用程序通过使用设备句柄调用 **GetRawInputDeviceInfo** 来获取设备信息。此句柄可以来自 **WM_INPUT**，也可以来自 **RAWINPUTHEADER** 的 **hDevice** 成员。

另请参阅

- [键盘输入](#)
- [关于键盘输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用原始输入

项目 • 2023/06/12

本部分包括用于以下目的的示例代码：

- [注册原始输入](#)
 - [示例 1](#)
 - [示例 2](#)
- [执行原始输入的标准读取](#)
- [执行原始输入的缓冲读取](#)

注册原始输入

示例 1

在此示例中，应用程序指定来自游戏控制器的原始输入（游戏板和游戏杆）以及电话使用页面（应答计算机除外）的所有设备。

C++

```
RAWINPUTDEVICE Rid[4];

Rid[0].usUsagePage = 0x01;           // HID_USAGE_PAGE_GENERIC
Rid[0].usUsage = 0x05;               // HID_USAGE_GENERIC_GAMEPAD
Rid[0].dwFlags = 0;                 // adds game pad
Rid[0].hwndTarget = 0;

Rid[1].usUsagePage = 0x01;           // HID_USAGE_PAGE_GENERIC
Rid[1].usUsage = 0x04;               // HID_USAGE_GENERIC_JOYSTICK
Rid[1].dwFlags = 0;                 // adds joystick
Rid[1].hwndTarget = 0;

Rid[2].usUsagePage = 0x0B;           // HID_USAGE_PAGE_TELEPHONY
Rid[2].usUsage = 0x00;               // adds all devices from telephony page
Rid[2].dwFlags = RIDEV_PAGEONLY;
Rid[2].hwndTarget = 0;

Rid[3].usUsagePage = 0x0B;           // HID_USAGE_PAGE_TELEPHONY
Rid[3].usUsage = 0x02;               // HID_USAGE_TELEPHONY_ANSWERING_MACHINE
Rid[3].dwFlags = RIDEV_EXCLUDE;
Rid[3].hwndTarget = 0;

if (RegisterRawInputDevices(Rid, 4, sizeof(Rid[0])) == FALSE)
{
    //registration failed. Call GetLastError for the cause of the error.
}
```

示例 2

在此示例中，应用程序需要来自键盘和鼠标的原始输入，但希望忽略来自同一 键盘 和鼠标) (旧键盘和鼠标 窗口消息 。

C++

```
RAWINPUTDEVICE Rid[2];

Rid[0].usUsagePage = 0x01;           // HID_USAGE_PAGE_GENERIC
Rid[0].usUsage = 0x02;              // HID_USAGE_GENERIC_MOUSE
Rid[0].dwFlags = RIDEV_NOLEGACY;    // adds mouse and also ignores legacy
mouse messages
Rid[0].hwndTarget = 0;

Rid[1].usUsagePage = 0x01;           // HID_USAGE_PAGE_GENERIC
Rid[1].usUsage = 0x06;              // HID_USAGE_GENERIC_KEYBOARD
Rid[1].dwFlags = RIDEV_NOLEGACY;    // adds keyboard and also ignores legacy
keyboard messages
Rid[1].hwndTarget = 0;

if (RegisterRawInputDevices(Rid, 2, sizeof(Rid[0])) == FALSE)
{
    //registration failed. Call GetLastError for the cause of the error
}
```

执行原始输入的标准读取

此示例演示应用程序如何执行无缓冲区 (或标准) 从键盘或鼠标人机接口设备读取原始输入 (HID)，然后从设备打印出各种信息。

C++

```
case WM_INPUT:
{
    UINT dwSize;

    GetRawInputData((HRAWINPUT)lParam, RID_INPUT, NULL, &dwSize,
sizeof(RAWINPUTHEADER));
    LPBYTE lpb = new BYTE[dwSize];
    if (lpb == NULL)
    {
        return 0;
    }

    if (GetRawInputData((HRAWINPUT)lParam, RID_INPUT, lpb, &dwSize,
sizeof(RAWINPUTHEADER)) != dwSize)
        OutputDebugString (TEXT("GetRawInputData does not return correct
size !\n));
```

```

RAWINPUT* raw = (RAWINPUT*)lpb;

if (raw->header.dwType == RIM_TYPEKEYBOARD)
{
    hResult = StringCchPrintf(szTempOutput, STRSAFE_MAX_CCH,
        TEXT(" Kbd: make=%04x Flags:%04x Reserved:%04x
ExtraInformation:%08x, msg=%04x VK=%04x \n"),
        raw->data.keyboard.MakeCode,
        raw->data.keyboard.Flags,
        raw->data.keyboard.Reserved,
        raw->data.keyboard.ExtraInformation,
        raw->data.keyboard.Message,
        raw->data.keyboard.VKey);
    if (FAILED(hResult))
    {
        // TODO: write error handler
    }
    OutputDebugString(szTempOutput);
}
else if (raw->header.dwType == RIM_TYPEMOUSE)
{
    hResult = StringCchPrintf(szTempOutput, STRSAFE_MAX_CCH,
        TEXT("Mouse: usFlags=%04x ulButtons=%04x usButtonFlags=%04x
usButtonData=%04x ulRawButtons=%04x lLastX=%04x lLastY=%04x
ulExtraInformation=%04x\r\n"),
        raw->data.mouse.usFlags,
        raw->data.mouse.ulButtons,
        raw->data.mouse.usButtonFlags,
        raw->data.mouse.usButtonData,
        raw->data.mouse.ulRawButtons,
        raw->data.mouse.lLastX,
        raw->data.mouse.lLastY,
        raw->data.mouse.ulExtraInformation);

    if (FAILED(hResult))
    {
        // TODO: write error handler
    }
    OutputDebugString(szTempOutput);
}

delete[] lpb;
return 0;
}

```

执行原始输入的缓冲读取

此示例演示应用程序如何从通用 HID 对原始输入执行缓冲读取。

```

case MSG_GETRIBUFFER: // Private message
{
    UINT cbSize;
    Sleep(1000);

    VERIFY(GetRawInputBuffer(NULL, &cbSize, sizeof(RAWINPUTHEADER)) == 0);
    cbSize *= 16; // up to 16 messages
    Log(_T("Allocating %d bytes"), cbSize);
    PRAWINPUT pRawInput = (PRAWINPUT)malloc(cbSize);
    if (pRawInput == NULL)
    {
        Log(_T("Not enough memory"));
        return;
    }

    for (;;)
    {
        UINT cbSizeT = cbSize;
        UINT nInput = GetRawInputBuffer(pRawInput, &cbSizeT,
    sizeof(RAWINPUTHEADER));
        Log(_T("nInput = %d"), nInput);
        if (nInput == 0)
        {
            break;
        }

        ASSERT(nInput > 0);
        PRAWINPUT* paRawInput = (PRAWINPUT*)malloc(sizeof(PRAWINPUT) *
nInput);
        if (paRawInput == NULL)
        {
            Log(_T("paRawInput NULL"));
            break;
        }

        PRAWINPUT pri = pRawInput;
        for (UINT i = 0; i < nInput; ++i)
        {
            Log(_T(" input[%d] = @%p"), i, pri);
            paRawInput[i] = pri;
            pri = NEXTRAWINPUTBLOCK(pri);
        }

        free(paRawInput);
    }
    free(pRawInput);
}

```

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

原始输入参考

项目 · 2024/02/24

本节内容

- 原始输入函数
- 原始输入宏
- 原始输入通知
- 原始输入结构

反馈

此页面是否有帮助?

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

原始输入函数

项目 · 2023/06/13

本节内容

- [DefRawInputProc](#)
- [GetRawInputBuffer](#)
- [GetRawInputData](#)
- [GetRawInputDeviceInfo](#)
- [GetRawInputDeviceList](#)
- [GetRegisteredRawInputDevices](#)
- [RegisterRawInputDevices](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

DefRawInputProc 函数 (winuser.h)

项目2024/03/04

与 [DefWindowProcA](#) 和 [DefWindowProcW](#) 不同，此函数不执行任何处理。

DefRawInputProc 仅检查 cbSizeHeader 的值是否对应于 [RAWINPUTHEADER](#) 的预期大小。

语法

C++

```
LRESULT DefRawInputProc(
    [in] PRAWINPUT *paRawInput,
    [in] INT      nInput,
    [in] UINT     cbSizeHeader
);
```

参数

[in] paRawInput

类型: [PRAWINPUT*](#)

已忽略。

[in] nInput

类型: [INT](#)

已忽略。

[in] cbSizeHeader

类型: [UINT](#)

[RAWINPUTHEADER](#) 结构的大小 (以字节为单位) 。

返回值

类型: [LRESULT](#)

如果成功，函数将返回 0。否则返回 -1。

要求

 展开表

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-rawinput-l1-1-0 ()

请参阅

概念性

[RAWINPUT](#)

[RAWINPUTHEADER](#)

[原始输入](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

getRawInputBuffer 函数 (winuser.h)

项目2024/03/04

对调用线程的消息队列中找到的原始输入消息数据执行缓冲读取。

语法

C++

```
UINT GetRawInputBuffer(
    [out, optional] PRAWINPUT pData,
    [in, out]        PUINT     pcbSize,
    [in]             UINT      cbSizeHeader
);
```

参数

[out, optional] pData

类型: PRAWINPUT

指向包含原始输入数据的 RAWINPUT 结构的缓冲区的指针。 缓冲区应在指针边界上对齐，该边界是 32 位体系结构上的 DWORD，在 64 位体系结构上是 QWORD。

如果 为 NULL，则第一个原始输入消息数据的大小 (所需的最小缓冲区) (以字节为单位) 在 * 中返回。

[in, out] pcbSize

类型: PUINT

所提供的 RAWINPUT 缓冲区的大小 (以字节为单位)。

[in] cbSizeHeader

类型: UINT

RAWINPUTHEADER 结构的大小 (以字节为单位)。

返回值

类型: UINT

如果 *pData* 为 NULL 且函数成功，则返回值为零。如果 *pData* 不为 NULL 且函数成功，则返回值为写入 *pData* 的 RAWINPUT 结构数。

如果发生错误，则返回值 (UINT) -1。调用 [GetLastError](#) 获取错误代码。

注解

当应用程序收到原始输入时，其消息队列将获取 WM_INPUT 消息，并设置队列状态标志 QS_RAWINPUT。

使用 [GetRawInputBuffer](#)，在可变大小的 RAWINPUT 结构的数组中读取原始输入数据，并从调用线程的消息队列中删除相应的 WM_INPUT 消息。可以使用缓冲区多次调用此方法，这些缓冲区在读取所有原始输入消息之前无法容纳所有消息的数据。

[NEXTRAWINPUTBLOCK](#) 宏允许应用程序遍历 RAWINPUT 结构的数组。

如果已成功从消息队列中读取所有原始输入消息，则会从调用线程的消息队列状态中清除 QS_RAWINPUT 标志。

① 备注

WOW64：若要获取原始输入缓冲区的正确大小，请不要使用 **hpsize*，而应改用 **hpsize* * 8。若要确保 [GetRawInputBuffer](#) 在 WOW64 上正常运行，必须将 [RAWINPUT](#) 结构对齐 8 个字节。以下代码演示如何对齐 WOW64 的 RAWINPUT。

C#

```
[StructLayout(LayoutKind.Explicit)]
internal struct RAWINPUT
{
    [FieldOffset(0)]
    public RAWINPUTHEADER header;

    [FieldOffset(16+8)]
    public RAWMOUSE mouse;

    [FieldOffset(16+8)]
    public RAWKEYBOARD keyboard;

    [FieldOffset(16+8)]
    public RAWHID hid;
}
```

要求

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[GetMessage](#)

[NEXTRAWINPUTBLOCK](#)

[RAWINPUT](#)

[RAWINPUTHEADER](#)

[原始输入](#)

引用

[原始输入概述](#)

反馈

此页面是否有帮助?

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

getRawInputData 函数 (winuser.h)

项目2024/03/04

从指定设备检索原始输入。

语法

C++

```
UINT GetRawInputData(
    [in]          HRAWINPUT hRawInput,
    [in]          UINT      uiCommand,
    [out, optional] LPVOID   pData,
    [in, out]     PUINT    pcbSize,
    [in]          UINT      cbSizeHeader
);
```

参数

[in] hRawInput

类型: **HRAWINPUT**

RAWINPUT 结构的句柄。 这来自 [WM_INPUT](#) 中的 *IParam*。

[in] uiCommand

类型: **UINT**

命令标志。 此参数的取值可为下列值之一：

[] 展开表

值	含义
RID_HEADER 0x10000005	从 RAWINPUT 结构获取标头信息。
RID_INPUT 0x10000003	从 RAWINPUT 结构获取原始数据。

[out, optional] pData

类型: **LPVOID**

指向来自 [RAWINPUT](#) 结构的数据的指针。这取决于 *uiCommand* 的值。如果 *pData* 为 **NULL**，则会在 **pcSize* 中返回所需的缓冲区大小。

[in, out] *pcbSize*

类型: **PUINT**

pData 中数据的大小 (以字节为单位)。

[in] *cbSizeHeader*

类型: **UINT**

[RAWINPUTHEADER](#) 结构的大小 (以字节为单位)。

返回值

类型: **UINT**

如果 *pData* 为 **NULL** 且函数成功，则返回值为 0。如果 *pData* 不为 **NULL** 且函数成功，则返回值是复制到 *pData* 中的字节数。

如果出现错误，则返回值为 **UINT** () -1。

注解

[GetRawInputData](#) 一次获取一个 [RAWINPUT](#) 结构的原始输入。相比之下，[GetRawInputBuffer](#) 获取 [RAWINPUT](#) 结构的数组。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

要求	值
Library	User32.lib
DLL	User32.dll
API 集	在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-rawinput-l1-1-0 ()

请参阅

概念性

[GetRawInputBuffer](#)

[RAWINPUT](#)

[RAWINPUTHEADER](#)

[原始输入](#)

引用

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

getRawInputDeviceInfoA 函数 (winuser.h)

项目2023/08/27

检索有关原始输入设备的信息。

语法

C++

```
UINT GetRawInputDeviceInfoA(
    [in, optional]     HANDLE hDevice,
    [in]                UINT   uiCommand,
    [in, out, optional] LPVOID pData,
    [in, out]           PUINT pcbSize
);
```

参数

[in, optional] hDevice

类型: HANDLE

原始输入设备的句柄。 这来自 [RAWINPUTHEADER](#) 的 hDevice 成员或 [GetRawInputDeviceList](#)。

[in] uiCommand

类型: UINT

指定将在 *pData* 中返回的数据。 此参数的取值可为下列值之一：

值	含义
RIDI_PREPARSEDDATA 0x20000005	<i>pData</i> 是指向 顶级集合的预先分析数据的 缓冲区的 PHIDP_PREPARSED_DATA 指针。
RIDI_DEVICENAME 0x20000007	<i>pData</i> 指向包含 设备接口名称 的字符串。 如果此设备 是使用共享访问模式打开的 ，则可以使用此名称调用 CreateFile 以打开 HID 集合，并使用返回的句柄调用 ReadFile 来读取输入报告，并使用 WriteFile 发送输出报告。

有关详细信息，请参阅 [打开 HID 集合](#) 和 [处理 HID 报表](#)。

仅对于此 *uiCommand*，则 *pData* 的字符计数 (,

RIDI_DEVICEINFO
0x2000000b

pData 指向 [RID_DEVICE_INFO](#) 结构。

[in, out, optional] *pData*

类型: LPVOID

指向包含 *uiCommand* 指定信息的缓冲区的指针。

如果 *uiCommand* RIDI_DEVICEINFO，请在调用 [GetRawInputDeviceInfo](#) 之前将 [RID_DEVICE_INFO](#) `sizeof(RID_DEVICE_INFO)` 的 *cbSize* 成员设置为。

[in, out] *pcbSize*

类型: PUINT

pData 中数据的大小 (以字节为单位)。

返回值

类型: UINT

如果成功，此函数将返回一个非负数，指示复制到 *pData* 的字节数。

如果 *pData* 不足以容纳数据，则函数返回 -1。如果 *pData* 为 NULL，则该函数将返回零值。在这两种情况下，将为 *pData* 缓冲区所需的最小大小。

调用 [GetLastError](#) 以识别任何其他错误。

注解

① 备注

winuser.h 标头将 [GetRawInputDeviceInfo](#) 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	ext-ms-win-ntuser-rawinput-l1-1-0 (在 Windows 10 版本 10.0.14393 中引入)

请参阅

概念性

[RAWINPUTHEADER](#)

[RID_DEVICE_INFO](#)

[原始输入](#)

引用

[WM_INPUT](#)

[顶级集合](#)

[预分析的数据](#)

[PHIDP_PREPARSED_DATA](#)

[打开 HID 集合](#)

[处理 HID 报告](#)

反馈

[此页面是否有帮助？](#)

是

否

在 Microsoft Q&A 获取帮助

GetRawInputDeviceList 函数 (winuser.h)

项目2023/08/27

枚举附加到系统的原始输入设备。

语法

C++

```
UINT GetRawInputDeviceList(
    [out, optional] PRAWINPUTDEVICELIST pRawInputDeviceList,
    [in, out]          PUINT            puiNumDevices,
    [in]               UINT             cbSize
);
```

参数

[out, optional] pRawInputDeviceList

类型: PRAWINPUTDEVICELIST

附加到系统的设备的 RAWINPUTDEVICELIST 结构的数组。如果为 NULL，则设备数在 *puiNumDevices 中返回。

[in, out] puiNumDevices

类型: PUINT

如果 pRawInputDeviceList 为 NULL，函数将使用附加到系统的设备数填充此变量；否则，此变量指定 pRawInputDeviceList 指向的缓冲区中可以包含的 RAWINPUTDEVICELIST 结构的数目。如果此值小于附加到系统的设备数，则函数将返回此变量中的实际设备数，并失败并 ERROR_INSUFFICIENT_BUFFER。如果此值大于或等于附加到系统的设备数，则该值保持不变，并将设备数报告为返回值。

[in] cbSize

类型: UINT

RAWINPUTDEVICELIST 结构的大小（以字节为单位）。

返回值

类型: **UINT**

如果函数成功，则返回值是 *pRawInputDeviceList* 指向的缓冲区中存储的设备数。

如果出现任何其他错误，该函数返回 (UINT) -1，[GetLastError](#) 返回错误指示。

注解

从此函数返回的设备是鼠标、键盘和其他人机接口设备 (HID) 设备。

若要获取有关附加设备的详细信息，请使用 [RAWINPUTDEVICELIST](#) 中的 *hDevice* 调用 [GetRawDeviceInfo](#)。

示例

以下示例代码演示对 [GetRawInputDeviceList](#) 的典型调用：

C++

```
UINT nDevices;
PRAWINPUTDEVICELIST pRawInputDeviceList = NULL;
while (true) {
    if (GetRawInputDeviceList(NULL, &nDevices, sizeof(RAWINPUTDEVICELIST))
!= 0) { Error(); }
    if (nDevices == 0) { break; }
    if ((pRawInputDeviceList = malloc(sizeof(RAWINPUTDEVICELIST) *
nDevices)) == NULL) { Error(); }
    nDevices = GetRawInputDeviceList(pRawInputDeviceList, &nDevices,
sizeof(RAWINPUTDEVICELIST));
    if (nDevices == (UINT)-1) {
        if (GetLastError() != ERROR_INSUFFICIENT_BUFFER) { Error(); }
        // Devices were added.
        free(pRawInputDeviceList);
        continue;
    }
    break;
}
// do the job...
// after the job, free the RAWINPUTDEVICELIST
free(pRawInputDeviceList);
```

要求

端	
最低受支持的服务 器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-rawinput-l1-1-0 ()

请参阅

概念性

[GetRawInputDeviceInfo](#)

[RAWINPUTDEVICELIST](#)

[原始输入](#)

引用

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

GetRegisteredRawInputDevices 函数 (winuser.h)

项目2023/08/27

检索有关当前应用程序的原始输入设备的信息。

语法

C++

```
UINT GetRegisteredRawInputDevices(
    [out, optional] PRAWINPUTDEVICE pRawInputDevices,
    [in, out]        PUINT          puiNumDevices,
    [in]             UINT           cbSize
);
```

参数

[out, optional] pRawInputDevices

类型: **PRAWINPUTDEVICE**

应用程序的 **RAWINPUTDEVICE** 结构的数组。

[in, out] puiNumDevices

类型: **PUINT**

pRawInputDevices* 中的 **RAWINPUTDEVICE 结构数。

[in] cbSize

类型: **UINT**

RAWINPUTDEVICE 结构的大小 (以字节为单位)。

返回值

类型: **UINT**

如果成功, 函数将返回一个非负数, 即写入缓冲区的 **RAWINPUTDEVICE** 结构数。

如果 *pRawInputDevices* 缓冲区太小或为 **NULL**，则该函数将最后一个错误设置为 **ERROR_INSUFFICIENT_BUFFER**，返回 -1，并将 *puiNumDevices* 设置为所需的设备数。如果函数因任何其他原因而失败，则返回 -1。有关更多详细信息，请调用 [GetLastError](#)。

注解

若要从设备接收原始输入，应用程序必须使用 [RegisterRawInputDevices](#) 对其进行注册。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll

请参阅

概念性

[RAWINPUTDEVICE](#)

[原始输入](#)

引用

[RegisterRawInputDevices](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

RegisterRawInputDevices 函数 (winuser.h)

项目2023/08/27

注册提供原始输入数据的设备。

语法

```
C++  
  
BOOL RegisterRawInputDevices(  
    [in] PCRAWINPUTDEVICE pRawInputDevices,  
    [in] UINT             uiNumDevices,  
    [in] UINT             cbSize  
) ;
```

参数

[in] pRawInputDevices

类型: PCRAWINPUTDEVICE

RAWINPUTDEVICE 结构的数组，表示提供原始输入的设备。

[in] uiNumDevices

类型: UINT

pRawInputDevices 指向的 RAWINPUTDEVICE 结构的数目。

[in] cbSize

类型: UINT

RAWINPUTDEVICE 结构的大小 (以字节为单位)。

返回值

类型: BOOL

如果函数成功，则为 TRUE;否则为 FALSE。 如果函数失败，请调用 GetLastError 以获取详细信息。

注解

若要接收 [WM_INPUT](#) 消息，应用程序必须先使用 [RegisterRawInputDevices](#) 注册原始输入设备。默认情况下，应用程序不接收原始输入。

若要接收 [WM_INPUT_DEVICE_CHANGE](#) 消息，应用程序必须为 [由 RAWINPUTDEVICE 结构](#) 的 usUsagePage 和 usUsage 字段指定的每个设备类指定 RIDEV_DEVNOTIFY 标志。默认情况下，应用程序不会收到原始输入设备到达和删除 [WM_INPUT_DEVICE_CHANGE](#) 通知。

如果 [RAWINPUTDEVICE](#) 结构设置了 RIDEV_REMOVE 标志，并且 hwndTarget 参数未设置为 NULL，则参数验证将失败。

每个原始输入设备类只能注册一个窗口，以在进程 (上次调用 [RegisterRawInputDevices](#)) 传递的窗口内接收原始输入。因此，不应从库使用 [RegisterRawInputDevices](#)，因为它可能会干扰加载它的应用程序中已存在的任何原始输入处理逻辑。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)
Library	User32.lib
DLL	User32.dll
API 集	在 Windows 10 版本 10.0.14393 中引入的 ext-ms-win-ntuser-rawinput-l1-1-0 ()

请参阅

概念性

[RAWINPUTDEVICE](#)

[原始输入](#)

引用

WM_INPUT

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

原始输入宏

项目 · 2024/02/24

本节内容

- [GET_RAWINPUT_CODE_WPARAM](#)
- [NEXTRAWINPUTBLOCK](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

GET_RAWINPUT_CODE_WPARAM 宏 (winuser.h)

项目2024/03/04

从 wParam 检索 WM_INPUT 消息中的输入代码。

语法

C++

```
void GET_RAWINPUT_CODE_WPARAM(  
    wParam  
);
```

参数

wParam

来自WM_INPUT消息的wParam。

返回值

输入代码值。 可以是以下值之一：

[+] 展开表

值	含义
RIM_INPUT 0	输入在应用程序位于前台时发生。
RIM_INPUTSINK 1	输入在应用程序不位于前台时发生。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]

要求	值
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[RAWINPUT](#)

[原始输入](#)

引用

[WM_INPUT](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

NEXTRAWINPUTBLOCK 宏 (winuser.h)

项目2023/08/27

检索 [RAWINPUT](#) 结构数组中下一个结构的位置。

语法

C++

```
void NEXTRAWINPUTBLOCK(  
    ptr  
) ;
```

参数

ptr

指向 [RAWINPUT](#) 结构数组中的 结构的指针。

返回值

无

备注

重复调用此宏以遍历 [RAWINPUT](#) 结构的数组。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[RAWINPUT](#)

原始输入

引用

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

原始输入通知

项目 · 2024/01/28

本节内容

- WM_INPUT
- WM_INPUT_DEVICE_CHANGE

反馈

此页面是否有帮助?

是

否

WM_INPUT 消息

项目 • 2023/06/22

发送到正在获取原始输入的窗口。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_INPUT 0x00FF
```

参数

wParam

输入代码。 使用 [GET_RAWINPUT_CODE_WPARAM](#) 宏获取值。

可以是以下其中一个值：

值	含义
RIM_INPUT	应用程序在前台时发生输入。
0	应用程序必须调用 DefWindowProc ，以便系统可以执行清理。
RIM_INPUTSINK	当应用程序不在前台时发生输入。
1	

lParam

包含设备原始输入的 [RAWINPUT](#) 结构的 [HRAWINPUT](#) 句柄。 若要获取原始数据，请在调用 [GetRawInputData](#) 时使用此句柄。

返回值

如果应用程序处理此消息，则它应返回零。

备注

仅当应用程序调用具有有效设备规范的 [RegisterRawInputDevices](#) 时，原始输入才可用。

要求

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	Winuser.h (包括 Windows.h)

另请参阅

引用

[GetRawInputData](#)

[RegisterRawInputDevices](#)

[RAWINPUT](#)

[GET_RAWINPUT_CODE_WPARAM](#)

概念性

[原始输入](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

WM_INPUT_DEVICE_CHANGE消息

项目 • 2023/06/13

说明

发送到注册以接收原始输入的窗口。

原始输入通知仅在应用程序调用具有 [RIDEV_DEVNOTIFY](#) 标志的 [RegisterRawInputDevices](#) 之后可用。

窗口通过其 [WindowProc](#) 函数接收此消息。

C++

```
#define WM_INPUT_DEVICE_CHANGE 0x00FE
```

参数

wParam

类型: [WPARAM](#)

此参数的取值可为下列值之一:

值	含义
GIDC_ARRIVAL1	已将新设备添加到系统。 可以调用 GetRawInputDeviceInfo 以获取有关设备的详细信息。
GIDC_REMOVAL2	设备已从系统中删除。

lParam

类型: [LPARAM](#)

原始输入设备的 [句柄](#)。

返回值

如果应用程序处理此消息，则它应返回零。

另请参阅

概念性

[原始输入](#)

引用

[RegisterRawInputDevices](#)

[RAWINPUTDEVICE 结构](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

原始输入结构

项目 · 2023/06/13

本节内容

- [RAWHID](#)
- [RAWINPUT](#)
- [RAWINPUTDEVICE](#)
- [RAWINPUTDEVICELIST](#)
- [RAWINPUTHEADER](#)
- [RAWKEYBOARD](#)
- [RAWMOUSE](#)
- [RID_DEVICE_INFO](#)
- [RID_DEVICE_INFO_HID](#)
- [RID_DEVICE_INFO_KEYBOARD](#)
- [RID_DEVICE_INFO_MOUSE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

rawHID 结构 (winuser.h)

项目2024/03/04

描述来自人机接口设备 (HID) 的原始输入的格式。

语法

C++

```
typedef struct tagRAWHID {
    DWORD dwSizeHid;
    DWORD dwCount;
    BYTE bRawData[1];
} RAWHID, *PRAWHID, *LPRAWHID;
```

成员

`dwSizeHid`

类型: `DWORD`

`bRawData` 中每个 HID 输入的大小 (以字节为单位)。

`dwCount`

类型: `DWORD`

`bRawData` 中的 HID 输入数。

`bRawData[1]`

类型: `BYTE[1]`

原始输入数据，作为字节数组。

注解

每个 `WM_INPUT` 可以指示多个输入，但所有输入都来自同一 HID。 `bRawData` 数组的大小为 `dwSizeHid * dwCount`。

有关详细信息，请参阅 [解释 HID 报表](#)。

要求

 展开表

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[RAWINPUT](#)

[原始输入](#)

[人体学接口设备 \(HID\) 简介](#)

引用

[WM_INPUT](#)

[解释 HID 报告](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

RAWINPUT 结构 (winuser.h)

项目2023/03/18

包含来自设备的原始输入。

语法

C++

```
typedef struct tagRAWINPUT {
    RAWINPUTHEADER header;
    union {
        RAWMOUSE     mouse;
        RAWKEYBOARD keyboard;
        RAWHID       hid;
    } data;
} RAWINPUT, *PRAWINPUT, *LPRAWINPUT;
```

成员

header

类型: [RAWINPUTHEADER](#)

原始输入数据。

data

data.mouse

类型: [RAWMOUSE](#)

如果数据来自鼠标，则这是原始输入数据。

data.keyboard

类型: [RAWKEYBOARD](#)

如果数据来自键盘，则这是原始输入数据。

data.hid

类型: [RAWHID](#)

如果数据来自 HID，则这是原始输入数据。

备注

此结构的句柄在 [WM_INPUT](#) 的 *lParam* 参数中传递。

若要获取详细信息（如标头和原始输入的内容），请调用 [GetRawInputData](#)。

若要将消息循环中的 RAWINPUT 作为缓冲读取，请调用 [GetRawInputBuffer](#)。

若要获取设备特定信息，请使用 RAWINPUTHEADER 中的 *hDevice* 调用 [GetRawInputDeviceInfo](#)。

仅当应用程序使用有效设备规范调用 [RegisterRawInputDevices](#) 时，原始输入才可用。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

另请参阅

概念性

[GetRawInputBuffer](#)

[GetRawInputData](#)

[RAWHID](#)

[RAWINPUTHEADER](#)

[RAWKEYBOARD](#)

[RAWMOUSE](#)

[原始输入](#)

引用

RAWINPUTDEVICE 结构 (winuser.h)

项目2023/08/28

定义原始输入设备的信息。

语法

C++

```
typedef struct tagRAWINPUTDEVICE {
    USHORT usUsagePage;
    USHORT usUsage;
    DWORD dwFlags;
    HWND hwndTarget;
} RAWINPUTDEVICE, *PRAWINPUTDEVICE, *LPRAWINPUTDEVICE;
```

成员

usUsagePage

类型: USHORT

原始输入设备的[顶级集合“使用情况”页](#)。有关可能值的详细信息，请参阅[Windows 中支持的 HID 客户端](#)。

usUsage

类型: USHORT

原始输入设备的[顶级集合使用情况 ID](#)。有关可能值的详细信息，请参阅[Windows 中支持的 HID 客户端](#)。

dwFlags

类型: DWORD

指定如何解释 usUsagePage 和 usUsage 提供的信息 的模式标志。(默认) 或以下值之一，可为零。默认情况下，只要具有窗口焦点，操作系统就会从具有指定[顶级集合](#)(TLC) 的设备发送原始输入。

Value	含义
-------	----

RIDEV_REMOVE 0x00000001	如果已设置，则会从包含列表中删除顶级集合。 这会告知操作系统停止从与顶级集合匹配的设备读取。
RIDEV_EXCLUDE 0x00000010	如果已设置，则指定在读取完整使用情况页时要排除的顶级集合。 此标志仅影响已使用 RIDEV_PAGEONLY 指定其使用情况页的 TLC。
RIDEV_PAGEONLY 0x00000020	如果已设置，则指定其顶级集合来自指定 usUsagePage 的所有设备。 请注意，usUsage 必须为零。 若要排除特定的顶级集合，请使用 RIDEV_EXCLUDE。
RIDEV_NOLEGACY 0x00000030	如果设置，这将阻止由 usUsagePage 或 usUsage 指定的任何设备生成 旧消息。 这仅适用于鼠标和键盘。 请参阅“备注”。
RIDEV_INPUTSINK 0x00000100	如果设置，即使调用方不在前台，调用方也能接收输入。 请注意，必须指定 hwndTarget。
RIDEV_CAPTUREMOUSE 0x00000200	如果设置，鼠标按钮单击不会激活另一个窗口。 仅当为鼠标设备指定了 RIDEV_NOLEGACY 时，才能指定 RIDEV_CAPTUREMOUSE。
RIDEV_NOHOTKEYS 0x00000200	如果设置，则不处理应用程序定义的键盘设备热键。 但是，系统热键;例如，仍处理 Alt+TAB 和 CTRL+ALT+DEL。 默认情况下，处理所有键盘热键。 即使未指定 RIDEV_NOLEGACY 且 hwndTarget 为 NULL，也可以指定 RIDEV_NOHOTKEYS。
RIDEV_APPKEYS 0x00000400	如果设置，则处理应用程序命令键。 仅当为键盘设备指定 RIDEV_NOLEGACY 时，才能指定 RIDEV_APPKEYS。
RIDEV_EXINPUTSINK 0x00001000	如果设置，则仅当前台应用程序不处理输入时，调用方才能在后台接收输入。 换句话说，如果前台应用程序未注册原始输入，则已注册的后台应用程序将接收输入。 Windowsxp: 在 Windows Vista 之前不支持此标志
RIDEV_DEVNOTIFY 0x00002000	如果已设置，则调用方能够接收设备到达和设备移除 WM_INPUT_DEVICE_CHANGE 通知。 Windowsxp: 在 Windows Vista 之前不支持此标志

hwndTarget

类型: HWND

目标窗口的句柄。 如果 为 NULL，则它跟随键盘焦点。

注解

如果为鼠标或键盘设置了 RIDEV_NOLEGACY，则系统不会为该应用程序的该设备生成任何旧消息。例如，如果使用 RIDEV_NOLEGACY 设置鼠标 TLC，则不会生成 WM_LBUTTONDOWN 和相关 [旧鼠标消息](#)。同样，如果键盘 TLC 设置为 RIDEV_NOLEGACY，则不会生成 WM_KEYDOWN 和相关 [旧键盘消息](#)。

如果设置了 RIDEV_REMOVE 并且 hwndTarget 成员未设置为 NULL，则 RegisterRawInputDevices 函数将失败。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetRegisteredRawInputDevices](#)

[原始输入](#)

[人体学接口设备 \(HID\) 简介](#)

[Windows 中支持的 HID 客户端](#)

[HID USB 主页](#) ↗

引用

[RegisterRawInputDevices](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

RAWINPUTDEVICELIST 结构 (winuser.h)

项目2024/03/04

包含有关原始输入设备的信息。

语法

C++

```
typedef struct tagRAWINPUTDEVICELIST {  
    HANDLE hDevice;  
    DWORD dwType;  
} RAWINPUTDEVICELIST, *PRAWINPUTDEVICELIST;
```

成员

hDevice

类型：句柄

原始输入设备的句柄。

dwType

类型：DWORD

设备类型。 这可以是以下值之一。

[] 展开表

值	含义
RIM_TYPEHID 2	设备是非键盘和鼠标的 HID。
RIM_TYPEKEYBOARD 1	设备是键盘。
RIM_TYPEMOUSE 0	设备是鼠标。

要求

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetRawInputDeviceList](#)

原始输入

引用

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

RAWINPUTHEADER 结构 (winuser.h)

项目2023/08/28

包含属于原始输入数据的标头信息。

语法

C++

```
typedef struct tagRAWINPUTHEADER {
    DWORD dwType;
    DWORD dwSize;
    HANDLE hDevice;
    WPARAM wParam;
} RAWINPUTHEADER, *PRAWINPUTHEADER, *LPRAWINPUTHEADER;
```

成员

`dwType`

类型: `DWORD`

原始输入的类型。 可以为下列值之一：

值	含义
<code>RIM_TYPEMOUSE 0</code>	原始输入来自鼠标。
<code>RIM_TYPEKEYBOARD 1</code>	原始输入来自键盘。
<code>RIM_TYPEHID 2</code>	原始输入来自不是键盘或鼠标的某些设备。

`dwSize`

类型: `DWORD`

整个输入数据包的大小 (以字节为单位) 。 这包括 `RAWINPUT` 以及 `RAWHID` 可变长度数组中可能的额外输入报告。

`hDevice`

类型: `HANDLE`

生成原始输入数据的设备的句柄。

wParam

类型: WPARAM

在[WM_INPUT](#)消息的 *wParam* 参数中传递的值。

注解

若要获取有关设备的详细信息, 请在调用 [GetRawInputDeviceInfo](#) 时使用 *hDevice*。如果从精确式触摸板接收输入, 则 *hDevice* 可以为零。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetRawInputDeviceInfo](#)

[RAWINPUT 结构](#)

[RAWKEYBOARD 结构](#)

[RAWMOUSE 结构](#)

[RAWHID 结构](#)

[原始输入](#)

引用

[WM_INPUT](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

rawKEYBOARD 结构 (winuser.h)

项目2023/08/28

包含有关键盘状态的信息。

语法

C++

```
typedef struct tagRAWKEYBOARD {
    USHORT MakeCode;
    USHORT Flags;
    USHORT Reserved;
    USHORT VKey;
    UINT   Message;
    ULONG  ExtraInformation;
} RAWKEYBOARD, *PRAWKEYBOARD, *LPRAWKEYBOARD;
```

成员

MakeCode

类型: USHORT

指定与按键关联的扫描代码。请参阅“备注”。

Flags

类型: USHORT

扫描代码信息的标志。可以是以下一项或多项：

Value	含义
RI_KEY_MAKE 0	键已关闭。
RI_KEY_BREAK 1	键已启动。
RI_KEY_E0 2	扫描代码具有 E0 前缀。
RI_KEY_E1 4	扫描代码具有 E1 前缀。

Reserved

类型: USHORT

保留;必须为零。

VKey

类型: USHORT

相应的 [旧虚拟密钥代码](#)。

Message

类型: UINT

相应的 [旧键盘窗口消息](#), 例如 [WM_KEYDOWN](#)、[WM_SYSKEYDOWN](#)等。

ExtraInformation

类型: ULONG

事件的特定于设备的其他信息。

注解

“[键盘输入概述](#)”中提供了 MakeCode 值的列表 (请参阅[扫描 1 个 make 列](#))。

对于 HID 键盘, MakeCode 值由 [HID 客户端映射器驱动程序](#) 生成, 该驱动程序根据 [USB HID 到 PS/2 扫描代码转换表](#) 将 HID ↴ 用法转换为扫描代码 (请参阅 [PS/2 Set 1 Make 列](#))。

`KEYBOARD_OVERRUN_MAKE_CODE` 是当按下无效或无法识别的键组合或按下的键数超出此键盘的限制时发送的特殊 MakeCode 值。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

另请参阅

- [GetRawInputDeviceInfo](#)

- RAWINPUT
 - 原始输入
 - 键盘和鼠标 HID 客户端驱动程序
 - USB HID 到 PS/2 扫描代码转换表 ↴
 - KEYBOARD_INPUT_DATA 结构
 - 键盘输入
-

反馈

此页面是否有帮助?

 是

 否

在 Microsoft Q&A 获取帮助

RAWMOUSE 结构 (winuser.h)

项目2024/03/04

包含有关鼠标状态的信息。

语法

C++

```
typedef struct tagRAWMOUSE {
    USHORT usFlags;
    union {
        ULONG ulButtons;
        struct {
            USHORT usButtonFlags;
            USHORT usButtonData;
        } DUMMYSTRUCTNAME;
    } DUMMYUNIONNAME;
    ULONG ulRawButtons;
    LONG lLastX;
    LONG lLastY;
    ULONG ulExtraInformation;
} RAWMOUSE, *PRAWMOUSE, *LPRAWMOUSE;
```

成员

usFlags

类型: USHORT

鼠标状态。此成员可以是以下各项的任意合理组合。

[] 展开表

值	含义
MOUSE_MOVE_RELATIVE 0x00	鼠标移动数据相对于最后一个鼠标位置。有关鼠标运动的详细信息，请参阅以下“备注”部分。
MOUSE_MOVE_ABSOLUTE 0x01	鼠标移动数据基于绝对位置。有关鼠标运动的详细信息，请参阅以下“备注”部分。
MOUSE_VIRTUAL_DESKTOP 0x02	鼠标坐标映射到多个监视系统) 的虚拟桌面 (。有关鼠标运动的详细信息，请参阅以下“备注”部分。

值	含义
MOUSE_ATTRIBUTES_CHANGED0x04	鼠标属性已更改;应用程序需要查询鼠标属性。
MOUSE_MOVE_NO_COALESCE 0x08	此鼠标移动事件未合并。默认情况下，可以合并鼠标移动事件。 Windows XP/2000: 不支持此值。

DUMMYUNIONNAME

DUMMYUNIONNAME.ulButtons

类型: ULONG

保留。

DUMMYUNIONNAME.DUMMYSTRUCTNAME

DUMMYUNIONNAME.DUMMYSTRUCTNAME.usButtonFlags

类型: USHORT

鼠标按钮的转换状态。此成员可以是以下一个或多个值。

[+] 展开表

值	含义
RI_MOUSE_BUTTON_1_DOWN	向左按钮更改为向下。
RI_MOUSE_LEFT_BUTTON_DOWN0x0001	
RI_MOUSE_BUTTON_1_UP	向左按钮更改为向上。
RI_MOUSE_LEFT_BUTTON_UP0x0002	
RI_MOUSE_BUTTON_2_DOWNRI_MOUSE_RIGHT_BUTTON_DOWN 0x0004	向右按钮更改为向下键。
RI_MOUSE_BUTTON_2_UP	向右按钮更改为向上。
RI_MOUSE_RIGHT_BUTTON_UP0x0008	
RI_MOUSE_BUTTON_3_DOWN	中间按钮更改为向下。
RI_MOUSE_MIDDLE_BUTTON_DOWN0x0010	

值	含义
RI_MOUSE_BUTTON_3_UP 0x0020	中间按钮更改为向上。
RI_MOUSE_BUTTON_4_DOWN 0x0040	XBUTTON1更改为关闭。
RI_MOUSE_BUTTON_4_UP 0x0080	XBUTTON1更改为 up。
RI_MOUSE_BUTTON_5_DOWN 0x0100	XBUTTON2更改为关闭。
RI_MOUSE_BUTTON_5_UP 0x0200	XBUTTON2更改为 up。
RI_MOUSE_WHEEL 0x0400	<p>原始输入来自鼠标滚轮。 滚轮增量存储在 usButtonData 中。</p> <p>正值表示滚轮向前旋转（远离用户）；负值表示滚轮向后旋转（朝向用户）。有关详细信息，请参阅以下“备注”部分。</p>
RI_MOUSE_HWHEEL 0x0800	<p>原始输入来自水平鼠标滚轮。 滚轮增量存储在 usButtonData 中。</p> <p>正值表示滚轮向右旋转；负值表示滚轮向左旋转。有关详细信息，请参阅以下“备注”部分。</p> <p>Windows XP/2000：不支持此值。</p>

DUMMYUNIONNAME.DUMMYSTRUCTNAME.usButtonData

类型： USHORT

如果 **usButtonFlags** 具有 RI_MOUSE_WHEEL 或 RI_MOUSE_HWHEEL，则此成员指定滚轮旋转的距离。有关详细信息，请参阅以下“备注”部分。

ulRawButtons

类型： ULONG

鼠标按钮的原始状态。 Win32 子系统不使用此成员。

lLastX

类型: LONG

X 方向的运动。这是有符号的相对运动或绝对运动，具体取决于 usFlags 的值。

lLastY

类型: LONG

Y 方向的运动。这是有符号的相对运动或绝对运动，具体取决于 usFlags 的值。

ulExtraInformation

类型: ULONG

事件的特定于设备的其他信息。

注解

如果鼠标已移动（由 MOUSE_MOVE_RELATIVE 或 MOUSE_MOVE_ABSOLUTE 指示），则 lLastX 和 lLastY 指定有关该移动的信息。信息指定为相对或绝对整数值。

如果指定了 MOUSE_MOVE_RELATIVE 值，则 lLastX 和 lLastY 指定相对于上一个鼠标事件（上一个报告位置）的移动。正值表示鼠标向右移动（或向下移动）；负值表示鼠标向左移动（或向上移动）。

如果指定了 MOUSE_MOVE_ABSOLUTE 值，则 lLastX 和 lLastY 包含介于 0 和 65,535 之间的规范化绝对坐标。坐标 (0,0) 贴图到显示图面的左上角；坐标 (65535,65535) 映射到右下角。在多监视器系统中，坐标映射到主监视器。

如果除了 MOUSE_MOVE_ABSOLUTE 之外还指定了 MOUSE_VIRTUAL_DESKTOP，则坐标将映射到整个虚拟桌面。

C++

```
case WM_INPUT:  
{  
    UINT dwSize = sizeof(RAWINPUT);  
    static BYTE lp[(sizeof(RAWINPUT));  
  
    GetRawInputData((HRAWINPUT)lParam, RID_INPUT, lp, &dwSize,  
    sizeof(RAWINPUTHEADER));  
  
    RAWINPUT* raw = (RAWINPUT*)lp;  
  
    if (raw->header.dwType == RIM_TYPEMOUSE)  
    {
```

```

    if (raw->mouse.usFlags & MOUSE_MOVE_ABSOLUTE)
    {
        RECT rect;
        if (raw->mouse.usFlags & MOUSE_VIRTUAL_DESKTOP)
        {
            rect.left = GetSystemMetrics(SM_XVIRTUALSCREEN);
            rect.top = GetSystemMetrics(SM_YVIRTUALSCREEN);
            rect.right = GetSystemMetrics(SM_CXVIRTUALSCREEN);
            rect.bottom = GetSystemMetrics(SM_CYVIRTUALSCREEN);
        }
        else
        {
            rect.left = 0;
            rect.top = 0;
            rect.right = GetSystemMetrics(SM_CXSCREEN);
            rect.bottom = GetSystemMetrics(SM_CYSCREEN);
        }

        int absoluteX = MulDiv(raw->mouse.lLastX, rect.right, 65535) +
rect.left;
        int absoluteY = MulDiv(raw->mouse.lLastY, rect.bottom, 65535) +
rect.top;
        ...
    }
    else if (raw->mouse.lLastX != 0 || raw->mouse.lLastY != 0)
    {
        int relativeX = raw->mouse.lLastX;
        int relativeY = raw->mouse.lLastY;
        ...
    }
    ...
}

```

与旧的[WM_MOUSEMOVE](#)窗口消息原始输入鼠标事件不受控制面板的鼠标属性表中设置的**鼠标速度**的影响。有关详细信息，请参阅[鼠标输入概述](#)。

如果移动鼠标滚轮（由 **usButtonFlags** 中的**RI_MOUSE_WHEEL**或**RI_MOUSE_HWHEEL**指示），则 **usButtonData** 包含一个有符号的**短值**，该值指定滚轮的旋转距离。

滚轮旋转将是 **WHEEL_DELTA** 的倍数，设置为 120。这是要执行的操作的阈值，应针对每个增量执行一个此类操作（例如，滚动一个增量）。

增量设置为 120，以允许 Microsoft 或其他供应商构建更精细的分辨率滚轮（一个无槽位的自由旋转轮），从而在每次旋转发送更多消息，但每条消息中的值较小。若要使用此功能，请添加传入的增量值，直到达到 **WHEEL_DELTA**（以便在增量旋转中获取相同响应），或者滚动部分行以响应更频繁的消息。还可以选择滚动粒度并累积增量，直到达到它。

应用程序还可以使用带有 SPI_GETWHEELSCROLLINES 或 SPI_GETWHEELSCROLLCHARS 参数的 SystemParametersInfo API 检索当前的行滚动和字符滚动用户设置。

下面是此类滚轮处理代码的示例：

C++

```
if ((rawMouse.usButtonFlags & RI_MOUSE_WHEEL) == RI_MOUSE_WHEEL ||  
    (rawMouse.usButtonFlags & RI_MOUSE_HWHEEL) == RI_MOUSE_HWHEEL)  
{  
    static const unsigned long defaultScrollLinesPerWheelDelta = 3;  
    static const unsigned long defaultScrollCharsPerWheelDelta = 1;  
  
    float wheelDelta = (float)(short)rawMouse.usButtonData;  
    float numTicks = wheelDelta / WHEEL_DELTA;  
  
    bool isHorizontalScroll = (rawMouse.usButtonFlags & RI_MOUSE_HWHEEL) ==  
        RI_MOUSE_HWHEEL;  
    bool isScrollByPage = false;  
    float scrollDelta = numTicks;  
  
    if (isHorizontalScroll)  
    {  
        unsigned long scrollChars = defaultScrollCharsPerWheelDelta;  
        SystemParametersInfo(SPI_GETWHEELSCROLLCHARS, 0, &scrollChars, 0);  
        scrollDelta *= scrollChars;  
    }  
    else  
    {  
        unsigned long scrollLines = defaultScrollLinesPerWheelDelta;  
        SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, &scrollLines, 0);  
        if (scrollLines == WHEEL_PAGESCROLL)  
            isScrollByPage = true;  
        else  
            scrollDelta *= scrollLines;  
    }  
}
```

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetRawInputDeviceInfo](#)

[RAWINPUT](#)

[原始输入](#)

引用

[MOUSEINPUT 结构](#)

[SendInput 函数](#)

[MOUSE_INPUT_DATA 结构](#)

[鼠标输入概述 \(旧版\)](#)

[旧版\) \(鼠标输入通知](#)

[SystemParametersInfo](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

RID_DEVICE_INFO 结构 (winuser.h)

项目2023/08/28

定义来自任何设备的原始输入数据。

语法

C++

```
typedef struct tagRID_DEVICE_INFO {
    DWORD cbSize;
    DWORD dwType;
    union {
        RID_DEVICE_INFO_MOUSE     mouse;
        RID_DEVICE_INFO_KEYBOARD keyboard;
        RID_DEVICE_INFO_HID       hid;
    } DUMMYUNIONNAME;
} RID_DEVICE_INFO, *PRID_DEVICE_INFO, *LPRID_DEVICE_INFO;
```

成员

cbSize

类型: DWORD

RID_DEVICE_INFO结构的大小 (以字节为单位)。

dwType

类型: DWORD

原始输入数据的类型。此成员可以是以下值之一。

Value	含义
RIM_TYPEMOUSE 0	数据来自鼠标。
RIM_TYPEKEYBOARD 1	数据来自键盘。
RIM_TYPEHID 2	数据来自不是键盘或鼠标的 HID。

DUMMYUNIONNAME

DUMMYUNIONNAME.mouse

类型: [RID_DEVICE_INFO_MOUSE](#)

如果 dwType是RIM_TYPEMOUSE, 则这是定义鼠标 的[RID_DEVICE_INFO_MOUSE](#) 结构。

DUMMYUNIONNAME.keyboard

类型: [RID_DEVICE_INFO_KEYBOARD](#)

如果 dwType是RIM_TYPEKEYBOARD, 则这是定义键盘 的[RID_DEVICE_INFO_KEYBOARD](#) 结构。

DUMMYUNIONNAME.hid

类型: [RID_DEVICE_INFO_HID](#)

如果 dwType是RIM_TYPEHID, 则这是定义 HID 设备的 [RID_DEVICE_INFO_HID](#) 结构。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[GetRawInputDeviceInfo](#)

[RID_DEVICE_INFO_HID](#)

[RID_DEVICE_INFO_KEYBOARD](#)

[RID_DEVICE_INFO_MOUSE](#)

[原始输入](#)

引用

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

winuser.h) (RID_DEVICE_INFO_HID 结构

项目2023/08/28

定义来自指定的人机接口设备 (HID) 的原始输入数据。

语法

C++

```
typedef struct tagRID_DEVICE_INFO_HID {
    DWORD dwVendorId;
    DWORD dwProductId;
    DWORD dwVersionNumber;
    USHORT usUsagePage;
    USHORT usUsage;
} RID_DEVICE_INFO_HID, *PRID_DEVICE_INFO_HID;
```

成员

`dwVendorId`

类型: DWORD

HID 的供应商标识符。

`dwProductId`

类型: DWORD

HID 的产品标识符。

`dwVersionNumber`

类型: DWORD

HID 的版本号。

`usUsagePage`

类型: USHORT

设备的顶级集合使用情况页。

`usUsage`

类型: USHORT

设备的顶级集合使用情况。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[RID_DEVICE_INFO](#)

原始输入

引用

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

RID_DEVICE_INFO_KEYBOARD 结构 (winuser.h)

项目2023/08/28

定义来自指定键盘的原始输入数据。

语法

C++

```
typedef struct tagRID_DEVICE_INFO_KEYBOARD {
    DWORD dwType;
    DWORD dwSubType;
    DWORD dwKeyboardMode;
    DWORD dwNumberOfFunctionKeys;
    DWORD dwNumberOfIndicators;
    DWORD dwNumberOfKeysTotal;
} RID_DEVICE_INFO_KEYBOARD, *PRID_DEVICE_INFO_KEYBOARD;
```

成员

dwType

类型: DWORD

键盘的类型。请参阅[备注](#)。

Value	说明
0x4	增强的 101 或 102 键键盘 (兼容)
0x7	日语键盘
0x8	朝鲜语键盘
0x51	未知类型或 HID 键盘

dwSubType

类型: DWORD

键盘的供应商特定子类型。请参阅[备注](#)。

`dwKeyboardMode`

类型: DWORD

扫描代码模式。 通常为 1，这意味着使用 扫描代码集 1。 请参阅 [键盘扫描代码规范](#)。

`dwNumberOfFunctionKeys`

类型: DWORD

键盘上的功能键数。

`dwNumberOfIndicators`

类型: DWORD

键盘上的 LED 指示灯数。

`dwNumberOfKeysTotal`

类型: DWORD

键盘上的键总数。

注解

有关键盘类型、子类型、扫描代码模式和相关键盘布局的信息，请参阅 Windows SDK 中 *kbd.h*、*ntdd8042.h* 和 *ntddkbd.h* 标头中的文档以及 [键盘布局示例](#)。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[RID_DEVICE_INFO](#)

[原始输入](#)

[引用](#)

[KEYBOARD_ATTRIBUTES结构](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

RID_DEVICE_INFO_MOUSE 结构 (winuser.h)

项目2024/03/04

定义来自指定鼠标的原始输入数据。

语法

C++

```
typedef struct tagRID_DEVICE_INFO_MOUSE {
    DWORD dwId;
    DWORD dwNumberOfButtons;
    DWORD dwSampleRate;
    BOOL fHasHorizontalWheel;
} RID_DEVICE_INFO_MOUSE, *PRID_DEVICE_INFO_MOUSE;
```

成员

`dwId`

类型: `DWORD`

鼠标设备标识属性的位字段:

[+] 展开表

值	ntddmou.h 常量	说明
0x0080	MOUSE_HID_HARDWARE	HID 鼠标
0x0100	WHEELMOUSE_HID_HARDWARE	HID 滚轮鼠标
0x8000	HORIZONTAL_WHEEL_PRESENT	带水平滚轮的鼠标

`dwNumberOfButtons`

类型: `DWORD`

鼠标的按钮数。

`dwSampleRate`

类型: DWORD

每秒的数据点数。此信息可能不适用于每个鼠标设备。

fHasHorizontalWheel

类型: BOOL

如果鼠标具有用于水平滚动的滚轮，则为 TRUE；否则为 FALSE。

Windowsxp: 仅从 Windows Vista 开始支持此成员。

注解

对于鼠标，“使用情况页”为 1，“使用情况”为 2。

要求

[+] 展开表

要求	值
最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	winuser.h (包括 Windows.h)

请参阅

概念性

[RID_DEVICE_INFO](#)

[原始输入](#)

引用

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助