第一章　原　子

在本章中，讲述了所有管理原子的函数。原子是一个独一无二的值，它等同于一个字符串。这样，你可以使用原子迅速地引用和管理一些公共使用的字符串。原子是无符号的整数值。

原子有两种类型，局部的和全程的。在程序中局部的原子有着范围限制，全程的原子可以在任何一个正在执行的应用程序中访问，全程原子提供了一个进程之间通讯的途径。

ATOM AddAtom(LPCSTR lpszString)

AddAtom()增加lpszString指向的字符串到一应用程序的局部原子表中。同时，它返回一个独一无二的代表这个字符串的原子。如果有错误发生，将返回为0。

如果指定的字符串已经在原子表中，那么该串的引用计数将加一。引用计数是用来防止DeleteAtom()删除一个其他程序正在使用的字符串。

相关函数

DeleteAtom()

ATOM DeleteAtom(ATOM atom)

DeleteAtom()函数首先将指定局部原子的引用值减一，如果值为0，从局部原子表中删除与指定原子相联系的字符串。假若原子删除成功，则返回为0，否则返一非零值(返回为零仅仅发生在该原子被实际删除以后，若仅仅是简单的引用值减一，则为零)。

要删除的原子通过Atom来指定。

用　法

下面框架展示了如何增加、删除一个原子。

```
ATOM a1;

a1 = AddAtom("My Atom");

/* ... */

if(DeleteAtom(a1))
  MessageBox(hwnd, "Cannot delete", "Atom", MB_OK);
else
  MessageBox(hwnd, "Deleted", "Atom", MB_OK);
```

相关函数

AddAtom()，

FindAtom()

ATOM FindAtom(LPCSTR lpszString)

FindAtom()函数返回指定字符串的局部原子,也就是说,它在原子表中搜寻指定的字符串。如果发现,返回与之相连的原子。如果没有发现则返回为零。

要搜寻的字符串是指指针 lpszString 所指的字符串。

相关函数

AddAtom()

DeleteAtom()

HLOCAL GetAtomHandle(ATOM atom)

GetAtomHandle()函数对于 Windows 3.0 以上版本来说已经过时了,它用来返回一个指定原子的句柄。

UINT GetAtomName(ATOM atom,
LPSTR lpszString, int nSize)

GetAtomName()函数拷贝与用 atom 指定的局部原子相关的字符串到用 lpszString 指定的字符数组之中。如果成功,它返回字符串长度,否则如果该原子没找到,返回为零。

用 lpszString 指定的字符数组大小要用 nSize 来传送。

用 法

下面的框架程序增加一个字符串到原子表中,然后再使用 GetAtomName()显示它。

```
ATOM a1;

/* ... */

a1 = AddAtom("My Atom");
/* ... */
if(GetAtomName(a1, str, sizeof(str)))
  MessageBox(hwnd, str, "Atom Name", MB_OK);
```

相关函数

AddAtom()

ATOM GlobalAddAtom(LPCSTR lpszString)

GlobalAddAtom()函数用来增加一个字符串到全程原子表中。它返回一个独一无二的原子代表这个字符串。如果有错误发生,则返回为零。

如果指定字符串已存在于全程原子表中,那么这个字符串的引用值将加一。引用值是用来防止.通过一个对 GlobalDeleteAtom()的调用删除一个还需使用的字符串。

用 法

当一个应用终止之时,全程原子自动消除,不像局部原子那样,也就是说,你必须在程序

结束前删除所有的全程原子。

相关函数

GlobalDeleteAtom()

ATOM GlobalDeleteAtom(ATOM atom)

GlobalDeleteAtom()函数首先将指定原子的引用值减一。如果该引用值为零,则从全程原子表中删除与指定原子相联系的字符串。如果该原子被删除,则返回为零,否则返回值为非零值(仅仅当该原子被实际删除时,才返回零,如果仅仅是其引用值减一,返回值将不为零)。

要删除的原子是通过 atom 来传递的。

用 法

下面的程序框架展现了如何增加,并且删除一原子。

```
ATOM a1;

a1 = GlobalAddAtom("My Atom");

/* ... */

if(GlobalDeleteAtom(a1))
  MessageBox(hwnd, "Cannot delete", "Atom", MB_OK);
else
  MessageBox(hwnd, "Deleted", "Atom", MB_OK);
```

相关函数

GlobalAddAtom(),
GlobalFindAtom()

ATOM GlobalFindAtom(LPCSTR lpszString)

GlobalFindAtom()函数返回与指定字符串相联系的原子,即它在全程原子表中搜索指定字符串。如果找到了,便返回与之链接的原子,否则如果没有发现该字符串,便返回零值。

要搜索的字符串是 lpszString 所指向的字符串。

相关函数

GlobalAddAtom()
GlobalDeleteAtom()

UINT GlobalGetAtomName(ATOM atom, LPSTR lpszString, int nSize)

GlobalGetAtomName()函数从全程原子表中拷贝由 atom 指定的字符串到指针 lpszString 指向的字符数组之中,它返回字符串的长度,如果没有发现,返回值为零。

lpszString 指向的字符数组大小必需由 nSize 来传递。

用 法

下面的程序框架展示了增加字符串到原子表中,然后用函数 GlobalGetAtomName()再现它。

```
ATOM a1;

/* ... */

a1 = GlobalAddAtom("My Atom");
/* ... */
if(GlobalGetAtomName(a1, str, sizeof(str)))
  MessageBox(hwnd, str, "Atom Name", MB_OK);
```

相关函数

GlobalAddAtom()

BOOL Init AtomTable(int nSize)

InitAtomTable()函数用来初始化一个局部原子表,并设置它大小为 nSize。由于原子表存取采用哈希算法(hashing),所以表的大小必须为一质数。

用 法

局部的原子表在程序开始时便自动初始化,并设置其大小为 37。如果你想产生一较大的局部原子表,便仅仅需要调用 InitAtomTable()即可。

由于 InitAtomTable()用来建立原子表,所以,必须在将任一原子放入表中之前调用它。

相关函数

AddAtom()

LPCSTR MAKEINTATOM(WORD wValue)

MAKEINTATOM()函数是一个宏。用来生成一个与用 wValue 指定整数相对应的字符串,它返回指向这个字符串的指针。通过对 AddAtom()函数的调用加此字符串进入局部原子表中,你便可以使用这个指针(你可以增加这个字符串到全程原子表中,不过你得首先调用函数 GlobalAddAtom())。

一个完整的编程实例

下面这个程序是用来讲述几个原子的函数的用法。

```
/*
  Demonstrate atoms.
*/
#include <windows.h>
#include <string.h>

LONG FAR PASCAL WndProc(HWND, UINT, WPARAM, LPARAM);

char szProgName[] = "ProgName"; /* name of window class */
```

```
ATOM a1, a2;
char str[255];

int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPreInst,
                   LPSTR lpszCmdLine, int nCmdShow)
{
HWND hWnd;
MSG lpMsg;
WNDCLASS wcApp;

if (!hPreInst) { /* if no previous instance */
  wcApp.lpszClassName = szProgName; /* window class name */
  wcApp.hInstance     = hInst; /* handle to this instance */
  wcApp.lpfnWndProc   = WndProc; /* window function */
  wcApp.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wcApp.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
  wcApp.lpszMenuName  = NULL;

  /* make the background of window white */
  wcApp.hbrBackground = GetStockObject(WHITE_BRUSH);
  wcApp.style         = 0; /* use default window style */
  wcApp.cbClsExtra    = 0; /* no extra info */
  wcApp.cbWndExtra    = 0; /* no extra info */

  /* register the window class */
  if (!RegisterClass (&wcApp))
    return FALSE;
}

/* Now that window has been registered, a window
   can be created. */
hWnd = CreateWindow(
  szProgName,
  "Demonstrate Atoms",
  WS_OVERLAPPEDWINDOW,
  CW_USEDEFAULT,
  CW_USEDEFAULT,
  CW_USEDEFAULT,
  CW_USEDEFAULT,
  (HWND)NULL,
  (HMENU) NULL,
  (HANDLE)hInst,
  (LPSTR)NULL
);

/* Display the Window. */
ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

/* Create the message loop. */
```

```
  while (GetMessage(&lpMsg, NULL, 0, 0)) {
    TranslateMessage(&lpMsg); /* allow use of keyboard */
    DispatchMessage(&lpMsg); /* return control to Windows */
  }
  return(lpMsg.wParam);
}

/* Window Function. */
LONG FAR PASCAL WndProc(HWND hWnd, UINT messg,
                        WPARAM wParam, LPARAM lParam)
{
  HDC hdc;
  PAINTSTRUCT ps;

  switch (messg) {
    case WM_CREATE:
      /* define two atoms */
      a1 = AddAtom("This is a sample string. ");
      a2 = AddAtom("This is another sample string.");
      break;
    case WM_PAINT: /* process a paint request */
      hdc = BeginPaint(hWnd, &ps); /* get DC */

      /* display atomic strings */
      GetAtomName(a1, str, sizeof(str));
      TextOut(hdc, 1, 1, str, strlen(str));

      GetAtomName(a2, str, sizeof(str));
      TextOut(hdc, 1, 20, str, strlen(str));

      EndPaint(hWnd, &ps);
      break;
    case WM_DESTROY: /* terminate the program */
      /* remove the atoms */
      DeleteAtom(a1);
      DeleteAtom(a2);
      PostQuitMessage(0);
      break;
    /* Let Windows process any messages not specified in
       the preceding cases. */
    default:
      return(DefWindowProc(hWnd, messg, wParam, lParam))
  }
  return(0L);
}
```

第二章　通讯函数

这一章所描述的函数管理并配置通讯设备(端口)。用户将用这些函数来设置和检验对这些外设的不同选择。

注意:在 16 位 Windows 中,通讯设备用 OpenComm()函数打开,其状态由该函数返回的 ID 值所指示。而在 Windows NT/Win32 中,通讯外设由 CreateFile()函数打开,并由这个函数返回的句柄指示其状态。

int BuildCommDCB(LPCSTR lpszMODE,
DCB FAR * lpCtrlBlock)

BuildCommDCB()函数将一个 MODE 型的串行口工作方式设置命令转换成一个 DCB(设备控制块)结构(MODE 是 DOS 型的命令,它负责设置串行口和其他一些东西),这个函数在当操作成功时返回一个非零值,否则返回一个零值。

注意:在 Windows NT/Win32中 BuildCommDCB()函数返回值的类型不再是 int 型,而是 BOOL 型的。

MODE 型的命令被包含在由 lppszMODE 指向的串中。这个命令转换成一个 DCB 结构可用的形式,并拷贝到一个由 lpCtrlBlock 指向的结构中。

在16位 Windows 中,DCB 结构与在 Windows NT/Win32中所定义的有显著的不同。16位 Windows 的 DCB 结构定义如下:

```
typedef struct tagDCB
{
  BYTE Id;
  UINT BaudRate;
  BYTE ByteSize;
  BYTE Parity;
  BYTE StopBits;
  UINT RlsTimeout;
  UINT CtsTimeout;
  UINT DsrTimeout;

  UINT fBinary :1;
  UINT fRtsDisable :1;
  UINT fParity :1;
  UINT fOutxCtsFlow :1;
  UINT fOutxDsrFlow :1;
  UINT fDummy :2;
  UINT fDtrDisable :1;

  UINT fOutX :1;
  UINT fInX :1;
```

```
    UINT :PeChar :1;
    UINT fNull :1;
    UINT fChEvt :1;
    UINT fDtrflow :1;
    UINT fRtsflow :1;
    UINT fDummy2 :1;

    char XonChar;
    char XoffChar;
    UINT XonLim;
    UINT XoffLim;
    char PeChar;
    char EofChar;
    char EvtChar;
    UINT TxDelay;
} DCB;
```

在这里,Id 是用来定义外设的值,它是由设备驱动程序来设置的。

BaudRate 指定波特率,如果高位字节的值是255,那么低位字节采用以下自注释值中的一个,来决定波特率:

```
CBR_110
CBR_300
CBR_600
CBR_1200
CBR_2400
CBR_4800
CBR_9600
CBR_14400
CBR_19200
CBR_38400
CBR_56000
CBR_128000
CBR_256000
```

ByteSize 指定数据位的位数,其值必须在4到8之间(包括4和8),Parity 指定校验方式,它的值必须为以下各值中的一个:EVENPARITY(偶校验),ODDPARITY(奇校验),NOPARITY(不校验),或 MARKPARITY(1校验)。StopBits 设定停止位的位数,其值必须是 ONESTOPBIT(一位停止位),ONE5STOPBITS(1.5位停止位)或 TWOSTOPBITS(两位停止位)。

RlsTimeout 设定"载波检测"定时,其值以毫秒为单位。CtsTimeout 设定"允许发送"定时,其值也是以毫秒为单位。DsrTimeout 设定"数据装备准备好"定时,以毫秒为单位。

以下是15个标志位,在分别设置它们时产生相应的作用:

单元	作用
fBinary	二进制数通讯，EOF 不被认为是数据结束的标志（也就是说，EOF 只被简单地认为是一个普通的二进制数）
fRtsDisable	"请求发送"被禁止
fParity	使用奇偶校验方式
fOutxCtsFlow	使用"允许发送"功能
fOutxDsrFlow	使用"数据装置准备好"功能
fDummy	未使用
fDtrDisable	禁止使用"数据终端准备好"功能
fOutX	输出采用 XON/XOF 协议
fInX	输入采用 XON/XOF 协议
fPeChar	用 PeChar 中指定的字符去代替校验有错的字节
fNull	抛弃无效字符
fChEvt	当接收到的字符与 EvtChar 中指定的相符时，标记一个事件
fDtrFlow	输入采用"数据终端准备好"功能
fRtsFlow	输入采用"准备发送"功能
fDummy2	未使用

XonChar 中包含了用来表示 XON 的字符，XoffChar 中包含了用来表示 XOFF 的字符。XonLim 指定了在输入序列中一个 XON 信号发出以前发送字符的最小个数，XoffLim 指定了在输入序列中当一个 XOFF 信号发出以前发送字符的最大个数。

当设置 fPeChar 标志位后，由 PeChar 指定的字符来取代校验有错的字符。

EofChar 中指定了 EOF 字符。如果设置了 fChEvt 标志，接收到 EvtChar 中指定的字符将导致一个事件指示。

TxDelay 未使用。

在 Winodows NT/Win32中，DCB 结构是这样定义的：

```
typedef struct _DCB {
  DWORD DCBlength;
  DWORD BaudRate;
  DWORD fBinary: 1;
  DWORD fParity: 1;
  DWORD fOutxCtsFlow:1;
  DWORD fOutxDsrFlow:1;
  DWORD fDtrControl:2;
  DWORD fDsrSensitivity:1;
  DWORD fTXContinueOnXoff: 1;
  DWORD fOutX: 1;
  DWORD fInX: 1;
  DWORD fErrorChar: 1;
  DWORD fNull: 1;
  DWORD fRtsControl:2;
  DWORD fAbortOnError:1;
```

```
    DWORD fDummy2:17;
    WORD wReserved;
    WORD XonLim;
    WORD XoffLim;
    BYTE ByteSize;
    BYTE Parity;
    BYTE StopBits;
    char XonChar;
    char XoffChar;
    char ErrorChar;
    char EofChar;
    char EvtChar;
} DCB;
```

在这里,DCBlength 决定了 DCB 结构的大小,以字节为单位。

BaudRate 指定了波特率。如果高位字节的值是255,那么低位字节使用以下自注释值中的一个,来决定波特率:

```
CBR_110
CBR_300
CBR_600
CBR_1200
CBR_2400
CBR_4800
CBR_9600
CBR_14400
CBR_19200
CBR_38400
CBR_56000
CBR_128000
CBR_256000
```

以下是14个标志位,在分别设置它们时产生相应的作用:

单元	作用
fBinary	二进制数通讯。对于 Windows NT/Win32,这一位应该总是设置的
fParity	采用奇偶校验方式
fOutxCtsFlow	采用"允许发送"功能
fOutxDsrFlow	采用"数据装置准备好"功能
fDtrcontrol	数据终端准备好,必须为下列值中的一个: DTR_CONTROL_DISABLE DTR_CONTROL_ENABLE DTR_CONTROL_HANDSHAKE
fDsrSensitivity	驱动程序使用"数据装置准备好"信号
fTXContinueOnXoff	即使 XOFF 信号被 发出仍然继续发送

（续表）

单元	作用
fOutX	输出采用 XON/XOFF 协议
fInX	输入采用 XON/XOFF 协议
fErrorChar	采用 ErrorChar 中指定的字符来替代奇偶错误的字符
fNull	丢弃无效字符
fRtsControl	指定"请求发送"协议，其值必须为下列中的一个：
	RTS_CONTROL_DISABLE
	RTS_CONTROL_ENABLE
	RTS_CONTROL_HANDSHAKE
	RTS_CONTROL_TOGGLE
fAbortOnError	如果遇到错误，通讯中断
fDummy2	未使用

wReserved 是保留单元，必须设置为零。

XonLim 指定了在一个 XON 信号发出之前输入序列中字符的最小个数。XoffLim 指定了在一个 XOFF 信号送出以前输入序列中包含字符的最大个数。

ByteSize 指定了数据位的位数，必须在4至8之间(包括4和8)。Parity 指定了奇偶校验方式，其值须为下列中的一个：EVENPARITY，ODDPARITY，NOPARITY，MARKPARITY。StopBits 设定停止位的个数，其值应为 ONESTOPBIT，ONE5STOPBITS(1.5位停止位)，或 TWOSTOPBITS。

XonChar 中包含了用来表示 XON 信号的字符，XoffChar 中包含了用来表示 XOFF 信号的字符。

如果设置了 fErrorChar 标志，奇偶校验出错的字符将被 ErrorChar 指定的字符代替。

由 EofChar 指定的字符用来标志 EOF。如果接收到的字符与 EvtChar 中指定的字符相符，将导致一个事件标记。

用　法

下面的 DCB 结构设定了第二串行口，到1200波特，采用偶校验、7位数据位和2位停止位。

```
DCB  com2dcb;
BuildCommDCB("COM2:1200,e,7,2",&com2dcb);
```

相关函数

SetCommState()

int ClearCommBreak(int Device)

ClearCommBreak()对指定的外设重新开始通讯。它在操作成功时将返回一个零，如果出错则返回一个－1。

需要重新开始通讯的外设的 ID 由 Device 来指定。

如果调用 SetCommBreak()引发了系统挂起，那么调用 ClearCommBreak()来重新开始通讯。

注意:在 Windows NT/Win32中,ClearCommBreak()函数的推荐形式是

BOOL ClearCommBreak(HANDLE hDevice)

该函数对成功的操作返回非零值,否则返回零值。这时,外设的句柄被放置在 hDevice 中。

相关函数

SetCommBreak()

*** *Win32专用**

BOOL ClearCommError(HANDLE hDevice,
LPDWORD lpdwErr,LPCOMSTAT lpStatus)

ClearCommError()清除一个通讯错误,并允许后继的通讯进行。它同时获取了指定外设的出错和状态信息。该函数仅在通讯出错后才应被调用,它对成功的操作返回一个非零值,否则返回零值。

发送错误设备的句柄被放置在 hDevice 中。

由 lpdwErr 指出的变量接收描述出错类型的值。它是下表中所列值的组合。

值	错误类型
CE_BREAK	中断发送
CE_DNS	外设未被选定
CE_FRAME	帧错误
CE_IOE	I/O 错误
CE_MODE	无效的设备方式
CE_OOP	纸用完
CE_OVERRUN	缓冲区溢出
CE_PTO	并行口定时到
CE_RXOVER	输入缓冲区溢出
CE_RXPARITY	输入校验错误
CE_PXPARITY	输入校验错误
CE_TXFULL	输出缓冲区满

COMSTAT 结构由 lpStatus 指出,它接收指定设备的状态信息。COMSTAT 结构定义如下:

```
typedef struct _COMSTAT {
  DWORD fCtsHold : 1;
  DWORD fDsrHold : 1;
  DWORD fRlsdHold : 1;
  DWORD fXoffHold : 1;
  DWORD fXoffSent : 1;
  DWORD fEof : 1;
  DWORD fTxim : 1;
  DWORD fReserved : 25;
  DWORD cbInQue;
```

```
    DWORD cbOutQue;
} COMSTAT;
```

在这里，如果设备等待"允许发送"信号，fCtsHold 应被设置。如果设备等待"数据装置准备好"信号，fDsrHold 应被设置。如果设备等待"接收线路信号检测"信号，fRisdHold 应被设置。如果设备在收到一个 XOFF 信号以后要等待 XON 信号，fXoffHold 应被设置。如果送出一个 XOFF 信号，fXoffSent 将会为真。如果接收到一个 EOF 信号，fEof 将被设置。如果有一个字符要发送，fTxim 将被设置。

cbInQue 中包含了当前接收序列中字符的个数，cbOutQue 中包含了当前发送序列中字符的个数。

用　法

如果不需要知道设备的状态信息，lpStatus 参数可设为 NULL。

相关函数

ClearCommBreak()

SetCommBreak()

int CloseComm(int Device)

CloseComm()负责关闭由 OpenComm()函数打开的通讯设备。如果操作成功，它将返回零，失败则返回负值。

注意：本函数对于 Windows NT/Win32是不适用的。

被关闭的设备由 Device 来指定。

相关函数

OpenComm()

BOOL EnableCommNotification(int Device, HWND hWnd, int nNumBytes, int nQueueSize)

EnableCommNotification()函数决定是否将 WM_COMMNOTIFY 消息送到指定的窗口。它对成功的操作返回一个非零值，而对失败则返回零。

注意：在 Windows NT/Win32中不能使用该函数。

设备由 Device 来指定，它是 OpenComm()函数的返回值。所影响的窗口的句柄由 hWnd 来指定。在一个通知消息送出以前，应用程序的输入序列中的字节数由 nNumBytes 来指定。在一个通知消息送出以前，输出序列中保存字节的最小个数由 nQueueSize 来指定。

相关函数

OpenComm()

LONG EscapeCommNotification(int Device, int Code)

EscapeCommFunction()使由 Device 指定的设备去执行由 Code 所指定的功能。该函数对成功的操作返回零，否则为负值。

注意：对本函数的描述只对16位 Windows 有用。

对通讯设备操作的操作码放置在 Code 中,它必须是以下的一个值:

宏定义	功能
CLRDTR	重置"数据终端准备好"
CLRRTS	重置"请求发送"
GETMAXCOM	获取系统中串行口的数目
GETMAXLPT	获取系统中并行口的数目
RESETDEV	重置连接在一个并行口上的打印机(对串行口无效)
SETDTR	传送"数据终端准备好"
ETRTS	传送"请求发送"
SETXOFF	模拟收到一个 XOFF 信号
SETXON	模拟收到一个 XON 信号

用　法

EscapeCommFunction()用来使通讯口执行一个扩展功能。有可能在将来会支持附加的扩展功能。

相关函数

EscapeCommfunction()(Win32版)

****Win32专用**

BOOL EscapeCommFunction(HANDLE hDevice,
DWORD Code)

EscapeCommFunction()使由 hDevice 指定的设备去执行由 Code 所指定的功能。对成功的操作函数返回真,否则返回假。

注意:对本函数的描述只对 Windos NT/Win32有用。

hDevice 中的句柄是通过对 CreateFile()函数的调用获得的。

Code 中的通讯设备所执行的功能的操作码,必须为下列各值中的一个:

宏定义	功能
CLRBREAK	与调用 ClearCommBreak()函数相同
CLRDTR	重置"数据终端准备好"
CLRRTS	重置"请求发送"
SETBREAK	与调用 SetCommBreak()函数相同
SETDTR	传送"数据终端准备好"信号
SETRTS	传送"请求发送"信号
SETXOFF	模拟接收到 XOFF 信号
SETXON	模拟接收到 XON 信号

用　法

EscapeCommFunction()使指定的通讯端口去执行一个扩展的功能。它将来有可能会支持更多的扩展功能。

相关函数

EscapeCommBreak()(16位 Windows 版本)

int GetCommError(int Device, COMSTAT FAR *lpErr)

GetCommError()函数用来获取指定通讯设备的错误信息。它返回的错误值,是下表中的一个或多个:

值	错误类型
CE_BREAK	发生中断
CE_CTSTO	"允许发送"定时到
CE_DNS	设备未选定
CEE_DSRTO	"数据装置准备好"定时到
CE_FRAME	帧错误
CE_IOE	I/O 错误
CE_MODE	无效的设备工作方式
CE_OOP	纸尽
CE_OVERRUN	缓冲区溢出
CE_PTO	并行口定时到
CE_RLSDTO	"接收线路信号检测"定时到
CE_RXOVER	输入缓冲区溢出
CE_RXPARITY	输入校验出错
CE_TXFULL	输出缓冲区溢出

注意:该函数在 Windows NT/Win32中已废弃不用,今后的的版本将不再支持。

对设备的定义由 Device 来完成。

在函数的返回值中,由 lpErr 指出的 COMSTAT 结构将包含设备的有关信息。在16位 Windows 中 COMSTAT 结构的定义是这样的:

```
typedef struct tagCOMSTAT
{
  BYTE status;
  UINT cbInQue;
  UINT cbOutQue;
} COMSTAT;
```

在这里,status 包含了设备的状态信息。它是以下的一个或几个值:

值	状态
CSTF_CTSHOLD	等待"允许发送"信号
CSTF_DSRHOLD	等待"数据装置准备好"信号
CSTF_EOF	收到"文件结束"信号
CSTF_RLSDHOLD	等待"接收线路信号检测"信号

(续表)

值	状态
CSTF_TXIM	字符准备发送
CSTF_XOFFHOLD	收到了 XOFF 信号,设备处在等待状态
CSTF_XOFFSENT	发送 XOFF 信号

cbInQue 和 cbOutQue 中分别包含了输入和输出序列里字符的个数。

用 法

如果不需要任何状态信息,可将 lpErr 单元设为 NULL。

相关函数

ClearLCommError()

UNIT GetCommEventMask(int Device,int Events)

注意:该函数仅在16位 Windows 中有效,Windows NT/Win32中对应的函数是 GetCommMask()

GetCommEventMask()函数返回当前的设备事件控制字,并重置之。

这个返回值出当前出现的事件。它是下表中的一个或几个值:

值	事件
EV_BREAK	中断
EV_CTS	允许发送
EV_CTSS	"允许发送"的状态改变
EV_DSR	数据装置准备好
EV_ERR	出错
EV_PERR	打印机出错
EV_RING	振铃指示
EV_RLSD	接收线路信号检测
EV_RLSDS	"接收线路信号检测"状态改变
EV_RXCHAR	收到字符
EV_RXFLAG	收到事件字符
EV_TXEMPTY	发送缓冲区空

相关函数

SetCommEventState()

****Win32专用**

BOOL GetCommMask(HANDLE hDevice, LPDWORD lpdwMask)

GetCommMask()函数获取对指定外设的事件响应。它在操作成功时返回一个非零值,否则返回零。

外设的句柄由 hDevice 来指定。

lpdwMask 指出的变量接收事件响应，其值为下表中的一个或几个：

值	事件
EV_BREAK	输入中断
EV_CTS	"允许发送"状态改变
EV_DSR	"数据装置准备好"状态改变
EV_ERR	发生帧错误、奇偶校验出错或是缓冲区溢出
EV_RING	振铃检测
EV_RLSD	"接收线路信号检测"状态改变
EV_RXCHAR	收到字符
EV_RXFLAG	收到事件字符
EV_TXEMPTY	输出缓冲区空

用　法

以下的程序段是判断是否收到一个字符。

```
DWORD value;

/* ... */

GetCommMask(hDevice, &value);

if(value & EV_RXCHAR) /* character received */
```

相关函数

GetCommModemStatus()

SetCommMask()

*** *Win32专用**

BOOL GetCommModemStatus(HANDLE hDevice,
LPDWORD lpdwStatus)

GetCommModemStatus()函数取得指定调制解调器的状态信息。它对成功的操作返回非零值，否则返回零。

调制解调器的端口句柄在 hDevice 中。

lpdwStatus 指出的变量接收调制解调器的状态。它将包含下列的一个或几个值：

值	意义
MS_CTS_ON	调制解调器允许发送
MS_DSR_ON	调制解调器处在"数据装置准备好"状态
MS_RING_ON	调制解调器振铃指示开
MS_RLSD_ON	调制解调器"接收线路信号检测"开

相关函数