Communication Reference

The following elements are used with communications devices.

- Communication Functions
- Communication Structures

Communication Functions

The following functions are used with communications devices.

BuildCommDCB
BuildCommDCBAndTimeouts
ClearCommBreak
ClearCommError
CommConfigDialog
EscapeCommFunction
GetCommConfig
GetCommMask
GetCommModemStatus
GetCommProperties
GetCommState
GetCommTimeouts
GetDefaultCommConfig
PurgeComm
SetCommBreak
SetCommConfig
SetCommMask
SetCommState
SetCommTimeouts
SetDefaultCommConfig
SetupComm
TransmitCommChar
WaitCommEvent

BuildCommDCB

The **BuildCommDCB** function fills a specified **DCB** structure with values specified in a device-control string. The device-control string uses the syntax of the **mode** command.

```
BOOL BuildCommDCB(
  LPCTSTR lpDef,  // pointer to device-control string
```

```
  LPDCB lpDCB     // pointer to device-control block
);
```

Parameters

lpDef

Pointer to a null-terminated string that specifies device-control information. The string must have t same form as the **mode** command's command-line arguments. For example, the following string specifies a baud rate of 1200, no parity, 8 data bits, and 1 stop bit:

```
baud=1200 parity=N data=8 stop=1
```

The device name is ignored if it is included in the string, but it must specify a valid device, as follo

```
COM1: baud=1200 parity=N data=8 stop=1
```

For further information on **mode** command syntax, refer to the end-user documentation for your operating system.

lpDCB

Pointer to a **DCB** structure to be filled in.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **BuildCommDCB** function adjusts only those members of the **DCB** structure that are specifically affected by the *lpDef* parameter, with the following exceptions:

- If the specified baud rate is 110, the function sets the stop bits to 2 to remain compatible with the system's **mode** command.
- By default, **BuildCommDCB** disables XON/XOFF and hardware flow control. To enable flow control, you must explicitly set the appropriate members of the **DCB** structure.

The **BuildCommDCB** function only fills in the members of the **DCB** structure. To apply these settings t a serial port, use the **SetCommState** function.

There are older and newer forms of the **mode** command syntax. The **BuildCommDCB** function supports both forms. However, you cannot mix the two forms together.

The newer form of the **mode** command syntax lets you explicitly set the values of the flow control members of the **DCB** structure. If you use an older form of the **mode** syntax, the **BuildCommDCB** function sets the flow control members of the **DCB** structure, as follows:

- For a string such as **96,n,8,1** or any other older-form **mode** string that doesn't end with an **x** or a **p**:

  fInX, **fOutX**,**fOutXDsrFlow**,and **fOutXCtsFlow** are all set to FALSE

  fDtrControl is set to DTR_CONTROL_ENABLE

  fRtsControl is set to RTS_CONTROL_ENABLE
- For a string such as **96,n,8,1,x** or any other older-form **mode** string that finishes with an **x**:

fInX, **fOutX** are both set to TRUE

fOutXDsrFlow,**fOutXCtsFlow** are both set to FALSE.

fDtrControl is set to DTR_CONTROL_ENABLE

fRtsControl is set to RTS_CONTROL_ENABLE

- For a string such as **96,n,8,1,p** or any other older-form **mode** string that finishes with a **p**:

fInX, **fOutX** are both set to FALSE

fOutXDsrFlow,**fOutXCtsFlow** are both set to TRUE.

fDtrControl is set to DTR_CONTROL_HANDSHAKE

fRtsControl is set to RTS_CONTROL_HANDSHAKE

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

Communications Overview, Communication Functions, **DCB**, **SetCommState**

BuildCommDCBAndTimeouts

The **BuildCommDCBAndTimeouts** function translates a device-definition string into appropriate devic control block codes and then places these codes into a device control block. The function can also set up time-out values, including the possibility of no time-outs, for a device; the function's behavior in this rega varies based on the contents of the device-definition string.

```
BOOL BuildCommDCBAndTimeouts(
  LPCTSTR lpDef,                  // pointer to device-control string
  LPDCB lpDCB,                    // pointer to device-control block
  LPCOMMTIMEOUTS  lpCommTimeouts  // pointer to comm time-out structure
);
```

Parameters

lpDef
Pointer to a null-terminated string that specifies device-control information for the device. The function takes this string, parses it, and then sets appropriate values in the **DCB** structure pointed to by *lpDCB*.

lpDCB
Pointer to a **DCB** structure that the function fills with information from the device-control information string pointed to by *lpDef*. This **DCB** structure defines the control settings for a

communications device.

lpCommTimeouts

Pointer to a **COMMTIMEOUTS** structure that the function can use to set device time-out values.

The **BuildCommDcbAndTimeouts** function modifies its time-out setting behavior based on the presence or absence of a "TO=xxx" substring in the string specified by *lpDef*:

- If that string contains the substring "TO=ON", the function sets up total read and write time-out values for the device based on the time-out structure pointed to by *lpCommTimeouts*.
- If that string contains the substring "TO=OFF", the function sets up the device with no time-outs.
- If that string contains neither of the aforementioned "TO=xxx" substrings, the function ignor the time-out structure pointed to by *lpCommTimeouts*. The time-out structure will not be accessed.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

Communications Overview, Communication Functions, **BuildCommDCB**, **COMMTIMEOUTS**, **DCB**, **GetCommTimeouts**, **SetCommTimeouts**

ClearCommBreak

The **ClearCommBreak** function restores character transmission for a specified communications device a places the transmission line in a nonbreak state.

```
BOOL ClearCommBreak(
  HANDLE hFile   // handle to communications device
);
```

Parameters

hFile

Handle to the communications device. The **CreateFile** function returns this handle.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

A communications device is placed in a break state by the **SetCommBreak** or **EscapeCommFunction** function. Character transmission is then suspended until the break state is cleared by calling **ClearCommBreak**.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **ClearCommError**, **CreateFile**, **EscapeCommFunction**, **SetCommBreak**

ClearCommError

The **ClearCommError** function retrieves information about a communications error and reports the current status of a communications device. The function is called when a communications error occurs, a it clears the device's error flag to enable additional input and output (I/O) operations.

```
BOOL ClearCommError(
  HANDLE hFile,     // handle to communications device
  LPDWORD lpErrors, // pointer to variable to receive error codes
  LPCOMSTAT lpStat  // pointer to buffer for communications status
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.

lpErrors
Pointer to a 32-bit variable to be filled with a mask indicating the type of error. This parameter can be one or more of the following error codes:

Value	Meaning
CE_BREAK	The hardware detected a break condition.
CE_DNS	**Windows 95 and Windows 98:** A parallel device is not selected.
CE_FRAME	The hardware detected a framing error.
CE_IOE	An I/O error occurred during communications with the device.
CE_MODE	The requested mode is not supported, or the *hFile* parameter is invali If this value is specified, it is the only valid error.
CE_OOP	**Windows 95 and Windows 98:** A parallel device signaled that it is out of paper.
CE_OVERRUN	A character-buffer overrun has occurred. The next character is lost.

CE_PTO	**Windows 95 and Windows 98:** A time-out occurred on a parallel device.
CE_RXOVER	An input buffer overflow has occurred. There is either no room in the input buffer, or a character was received after the end-of-file (EOF) character.
CE_RXPARITY	The hardware detected a parity error.
CE_TXFULL	The application tried to transmit a character, but the output buffer was full.

lpStat

Pointer to a **COMSTAT** structure in which the device's status information is returned. If *lpStat* is NULL, no status information is returned.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

If a communications port has been set up with a TRUE value for the **fAbortOnError** member of the setu **DCB** structure, the communications software will terminate all read and write operations on the communications port when a communications error occurs. No new read or write operations will be accepted until the application acknowledges the communications error by calling the **ClearCommError** function.

The **ClearCommError** function fills the status buffer pointed to by the *lpStat* parameter with the current status of the communications device specified by the *hFile* parameter.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **ClearCommBreak**, **COMSTAT**, **CreateFile**, **DCB**

CommConfigDialog

The **CommConfigDialog** function displays a driver-supplied configuration dialog box.

```
BOOL CommConfigDialog(
  LPTSTR lpszName,    // pointer to device name string
  HWND hWnd,          // handle to window
  LPCOMMCONFIG lpCC   // pointer to comm configuration structure
);
```

Parameters

lpszName
Pointer to a null-terminated string specifying the name of the device for which a dialog box should displayed.
hWnd
Handle to the window that owns the dialog box. This parameter can be any valid window handle, o it should be NULL if the dialog box is to have no owner.
lpCC
Pointer to a **COMMCONFIG** structure. This structure contains initial settings for the dialog box before the call, and changed values after the call.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **CommConfigDialog** function requires a dynamic-link library (DLL) provided by the communicatio hardware vendor.

QuickInfo

Windows NT: Requires version 3.51 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

Communications Overview, Communication Functions, **COMMCONFIG**

EscapeCommFunction

The **EscapeCommFunction** function directs a specified communications device to perform an extended function.

```
BOOL EscapeCommFunction(
  HANDLE hFile,  // handle to communications device
  DWORD dwFunc   // extended function to perform
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
dwFunc
Specifies the code of the extended function to perform. This parameter can be one of the following

values:

Value	Meaning
CLRDTR	Clears the DTR (data-terminal-ready) signal.
CLRRTS	Clears the RTS (request-to-send) signal.
SETDTR	Sends the DTR (data-terminal-ready) signal.
SETRTS	Sends the RTS (request-to-send) signal.
SETXOFF	Causes transmission to act as if an XOFF character has been received.
SETXON	Causes transmission to act as if an XON character has been received.
SETBREAK	Suspends character transmission and places the transmission line in a bre state until the **ClearCommBreak** function is called (or **EscapeCommFunction** is called with the CLRBREAK extended functic code). The SETBREAK extended function code is identical to the **SetCommBreak** function. Note that this extended function does not flus data that has not been transmitted.
CLRBREAK	Restores character transmission and places the transmission line in a nonbreak state. The CLRBREAK extended function code is identical to the **ClearCommBreak** function.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Windows CE: Windows CE supports the following additional flags for the *dwFunc* parameter:

SETIR
: Sets the serial port in IR mode.

CLRIR
: Sets port into normal serial mode.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **ClearCommBreak**, **CreateFile**, **SetCommBrea**

GetCommConfig

The **GetCommConfig** function gets the current configuration of a communications device.

```
BOOL GetCommConfig(
```

```
  HANDLE hCommDev,     // handle to communications service
  LPCOMMCONFIG lpCC,   // pointer to comm configuration structure
  LPDWORD lpdwSize     // pointer to size of buffer
);
```

Parameters

hCommDev
Handle to the open communications device.
lpCC
Pointer to the buffer that receives the **COMMCONFIG** structure.
lpdwSize
Pointer to a 32-bit variable that specifies the size, in bytes, of the buffer pointed to by *lpCC*. When the function returns, the variable contains the number of bytes copied if the function succeeds, or tl number of bytes required if the buffer was too small.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the **GetLastError** function.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **SetCommConfig**, **COMMCONFIG**

GetCommMask

The **GetCommMask** function retrieves the value of the event mask for a specified communications devi

```
BOOL GetCommMask(
  HANDLE hFile,       // handle to communications device
  LPDWORD lpEvtMask   // pointer to variable to get event mask
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
lpEvtMask
Pointer to the 32-bit variable to be filled with a mask of events that are currently enabled. This parameter can be one or more of the following values:

Value	Meaning
EV_BREAK	A break was detected on input.
EV_CTS	The CTS (clear-to-send) signal changed state.
EV_DSR	The DSR (data-set-ready) signal changed state.
EV_ERR	A line-status error occurred. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY.
EV_EVENT1	An event of the first provider-specific type occured.
EV_EVENT2	An event of the second provider-specific type occured.
EV_PERR	A printer error occured.
EV_RING	A ring indicator was detected.
EV_RLSD	The RLSD (receive-line-signal-detect) signal changed state.
EV_RX80FULL	The receive buffer is 80 percent full.
EV_RXCHAR	A character was received and placed in the input buffer.
EV_RXFLAG	The event character was received and placed in the input buffer. The event character is specified in the device's **DCB** structure, which is applied to a serial port by using the **SetCommState** function.
EV_TXEMPTY	The last character in the output buffer was sent.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **GetCommMask** function uses a 32-bit mask variable to indicate the set of events that can be monitored for a particular communications resource. A handle to the communications resource can be specified in a call to the **WaitCommEvent** function, which waits for one of the events to occur. To modi the event mask of a communications resource, use the **SetCommMask** function.

Windows CE: Windows CE does not support the following values for the *lpEvtMask* parameter:

EV_EVENT1

EV_EVENT2

EV_RX80FULL

EV_PERR

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **DCB**, **SetCommMask**,

WaitCommEvent,

GetCommModemStatus

The **GetCommModemStatus** function retrieves modem control-register values.

```
BOOL GetCommModemStatus(
  HANDLE hFile,        // handle to communications device
  LPDWORD lpModemStat  // pointer to control-register values
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.

lpModemStat
Pointer to a 32-bit variable that specifies the current state of the modem control-register values. Thi parameter can be a combination of the following values:

Value	Meaning
MS_CTS_ON	The CTS (clear-to-send) signal is on.
MS_DSR_ON	The DSR (data-set-ready) signal is on.
MS_RING_ON	The ring indicator signal is on.
MS_RLSD_ON	The RLSD (receive-line-signal-detect) signal is on.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **GetCommModemStatus** function is useful when you are using the **WaitCommEvent** function to monitor the CTS, RLSD, DSR, or ring indicator signals. To detect when these signals change state, use **WaitCommEvent** and then use **GetCommModemStatus** to determine the state after a change occurs.

The function fails if the hardware does not support the control-register values.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **WaitCommEvent**

GetCommProperties

The **GetCommProperties** function fills a buffer with information about the communications properties f a specified communications device.

```
BOOL GetCommProperties(
  HANDLE hFile,           // handle to comm device
  LPCOMMPROP lpCommProp   // pointer to comm properties structure
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.

lpCommProp
Pointer to a **COMMPROP** structure in which the communications properties information is returned. This information can be used in subsequent calls to the **SetCommState**, **SetCommTimeouts**, or **SetupComm** function to configure the communications device.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **GetCommProperties** function returns information from a device driver about the configuration settings that are supported by the driver.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **COMMPROP**, **CreateFile**, **SetCommState**, **SetCommTimeouts**, **SetupComm**

GetCommState

The **GetCommState** function fills in a device-control block (a **DCB** structure) with the current control settings for a specified communications device.

```
BOOL GetCommState(
  HANDLE hFile,  // handle to communications device
  LPDCB lpDCB    // pointer to device-control block structure
```

```
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
lpDCB
Pointer to the **DCB** structure in which the control settings information is returned.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **DCB**, **SetCommState**

GetCommTimeouts

The **GetCommTimeouts** function retrieves the time-out parameters for all read and write operations on a specified communications device.

```
BOOL GetCommTimeouts(
  HANDLE hFile,                // handle to comm device
  LPCOMMTIMEOUTS lpCommTimeouts  // pointer to comm time-outs structure
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
lpCommTimeouts
Pointer to a **COMMTIMEOUTS** structure in which the time-out information is returned.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

For more information about time-out values for communications devices, see the **SetCommTimeouts** function.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **COMMTIMEOUTS**, **SetCommTimeouts**

GetDefaultCommConfig

The **GetDefaultCommConfig** function gets the default configuration for a communications device.

```
BOOL GetDefaultCommConfig(
  LPCSTR lpszName,    // pointer to device name string
  LPCOMMCONFIG lpCC,  // pointer to buffer that receives structure
  LPDWORD lpdwSize    // pointer to size of buffer
);
```

Parameters

lpszName
Pointer to a null-terminated string specifying the name of the device.
lpCC
Pointer to the buffer that receives the **COMMCONFIG** structure.
lpdwSize
Pointer to a 32-bit variable that specifies the size, in bytes, of the buffer pointed to by *lpCC*. Upon return, the variable contains the number of bytes copied if the function succeeds, or the number of bytes required if the buffer was too small.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the **GetLastError** function.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

Communications Overview, Communication Functions, **SetDefaultCommConfig**, **COMMCONFIG**

PurgeComm

The **PurgeComm** function can discard all characters from the output or input buffer of a specified communications resource. It can also terminate pending read or write operations on the resource.

```
BOOL PurgeComm(
  HANDLE hFile,  // handle to communications resource
  DWORD dwFlags  // action to perform
);
```

Parameters

hFile
Handle to the communications resource. The **CreateFile** function returns this handle.

dwFlags
Specifies the action to take. This parameter can be a combination of the following values:

Value	Meaning
PURGE_TXABORT	Terminates all outstanding overlapped write operations and retur immediately, even if the write operations have not been complete
PURGE_RXABORT	Terminates all outstanding overlapped read operations and return immediately, even if the read operations have not been complete
PURGE_TXCLEAR	Clears the output buffer (if the device driver has one).
PURGE_RXCLEAR	Clears the input buffer (if the device driver has one).

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

If a thread uses **PurgeComm** to flush an output buffer, the deleted characters are not transmitted. To empty the output buffer while ensuring that the contents are transmitted, call the **FlushFileBuffers** functi (a synchronous operation). Note, however, that **FlushFileBuffers** is subject to flow control but not to write time-outs, and it will not return until all pending write operations have been transmitted.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**

SetCommBreak

The **SetCommBreak** function suspends character transmission for a specified communications device ar places the transmission line in a break state until the **ClearCommBreak** function is called.

```
BOOL SetCommBreak(
  HANDLE hFile   // handle to communications device
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **ClearCommBreak**, **CreateFile**

SetCommConfig

The **SetCommConfig** function sets the current configuration of a communications device.

```
BOOL SetCommConfig(
  HANDLE hCommDev,     // handle to comm device
  LPCOMMCONFIG lpCC,   // pointer to comm configuration services
  DWORD dwSize         // size of structure
);
```

Parameters

hCommDev
Handle to the open communications device.
lpCC

Pointer to a **COMMCONFIG** structure.

dwSize

Specifies the size, in bytes, of the structure pointed to by *lpCC*.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **GetCommConfig**, **COMMCONFIG**

SetCommMask

The **SetCommMask** function specifies a set of events to be monitored for a communications device.

```
BOOL SetCommMask(
  HANDLE hFile,     // handle to communications device
  DWORD dwEvtMask  // mask that identifies enabled events
);
```

Parameters

hFile

Handle to the communications device. The **CreateFile** function returns this handle.

dwEvtMask

Specifies the events to be enabled. A value of zero disables all events. This parameter can be a combination of the following values:

Value	Meaning
EV_BREAK	A break was detected on input.
EV_CTS	The CTS (clear-to-send) signal changed state.
EV_DSR	The DSR (data-set-ready) signal changed state.
EV_ERR	A line-status error occurred. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY.
EV_RING	A ring indicator was detected.
EV_RLSD	The RLSD (receive-line-signal-detect) signal changed state.
EV_RXCHAR	A character was received and placed in the input buffer.
EV_RXFLAG	The event character was received and placed in the input buffer. The event character is specified in the device's **DCB** structure, which is applied to a serial port by using the **SetCommState** function.

EV_TXEMPTY The last character in the output buffer was sent.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **SetCommMask** function specifies the set of events that can be monitored for a particular communications resource. A handle to the communications resource can be specified in a call to the **WaitCommEvent** function, which waits for one of the events to occur. To get the current event mask of communications resource, use the **GetCommMask** function.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **DCB**, **GetCommMask**, **SetCommState**, **WaitCommEvent**

SetCommState

The **SetCommState** function configures a communications device according to the specifications in a device-control block (a **DCB** structure). The function reinitializes all hardware and control settings, but it does not empty output or input queues.

```
BOOL SetCommState(
  HANDLE hFile,  // handle to communications device
  LPDCB lpDCB    // pointer to device-control block structure
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
lpDCB
Pointer to a **DCB** structure containing the configuration information for the specified communications device.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **SetCommState** function uses a **DCB** structure to specify the desired configuration. The **GetCommState** function returns the current configuration.

To set only a few members of the **DCB** structure, you should modify a **DCB** structure that has been filled in by a call to **GetCommState**. This ensures that the other members of the **DCB** structure have appropriate values.

The **SetCommState** function fails if the **XonChar** member of the **DCB** structure is equal to the **XoffChar** member.

When **SetCommState** is used to configure the 8250, the following restrictions apply to the values for the **DCB** structure's **ByteSize** and **StopBits** members:

- The number of data bits must be 5 to 8 bits.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **BuildCommDCB**, **CreateFile**, **DCB**, **GetCommState**

SetCommTimeouts

The **SetCommTimeouts** function sets the time-out parameters for all read and write operations on a specified communications device.

```
BOOL SetCommTimeouts(
  HANDLE hFile,                // handle to comm device
  LPCOMMTIMEOUTS lpCommTimeouts  // pointer to comm time-out structure
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
lpCommTimeouts
Pointer to a **COMMTIMEOUTS** structure that contains the new time-out values.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **COMMTIMEOUTS**, **GetCommTimeouts**, **ReadFile**, **ReadFileEx**, **WriteFile**, **WriteFileEx**

SetDefaultCommConfig

The **SetDefaultCommConfig** function sets the default configuration for a communications device.

```
BOOL SetDefaultCommConfig(
  LPCSTR lpszName,     // pointer to device name string
  LPCOMMCONFIG lpCC,   // pointer to structure
  DWORD dwSize         // size of structure
);
```

Parameters

lpszName
Pointer to a null-terminated string specifying the name of the device.
lpCC
Pointer to a **COMMCONFIG** structure.
dwSize
Specifies the size, in bytes, of the structure pointed to by *lpCC*.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

Communications Overview, Communication Functions, **GetDefaultCommConfig**, **COMMCONFIG**

SetupComm

The **SetupComm** function initializes the communications parameters for a specified communications device.

```
BOOL SetupComm(
  HANDLE hFile,     // handle to communications device
  DWORD dwInQueue,  // size of input buffer
  DWORD dwOutQueue  // size of output buffer
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
dwInQueue
Specifies the recommended size, in bytes, of the device's internal input buffer.
dwOutQueue
Specifies the recommended size, in bytes, of the device's internal output buffer.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

After a process uses the **CreateFile** function to open a handle to a communications device, it can call **SetupComm** to set the communications parameters for the device. If it does not set them, the device uses the default parameters when the first call to another communications function occurs.

The *dwInQueue* and *dwOutQueue* parameters specify the recommended sizes for the internal buffers used by the driver for the specified device. For example, YMODEM protocol packets are slightly larger than 1024 bytes. Therefore, a recommended buffer size might be 1200 bytes for YMODEM communications. For Ethernet-based communications, a recommended buffer size might be 1600 bytes, which is slightly larger than a single Ethernet frame.

The device driver receives the recommended buffer sizes, but is free to use any input and output (I/O) buffering scheme, as long as it provides reasonable performance and data is not lost due to overrun (except under extreme circumstances). For example, the function can succeed even though the driver does not allocate a buffer, as long as some other portion of the system provides equivalent functionality.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **SetCommState**

TransmitCommChar

The **TransmitCommChar** function transmits a specified character ahead of any pending data in the outp buffer of the specified communications device.

```
BOOL TransmitCommChar(
  HANDLE hFile,  // handle to communications device
  char cChar     // character to transmit
);
```

Parameters

hFile
Handle to the communications device. The **CreateFile** function returns this handle.
cChar
Specifies the character to be transmitted.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **TransmitCommChar** function is useful for sending an interrupt character (such as a ctrl+c) to a hos system.

If the device is not transmitting, **TransmitCommChar** cannot be called repeatedly. Once **TransmitCommChar** places a character in the output buffer, the character must be transmitted before th function can be called again. If the previous character has not yet been sent, **TransmitCommChar** retur an error.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **WaitCommEvent**

WaitCommEvent

The **WaitCommEvent** function waits for an event to occur for a specified communications device. The s of events that are monitored by this function is contained in the event mask associated with the device

handle.

```
BOOL WaitCommEvent(
  HANDLE hFile,                 // handle to communications device
  LPDWORD lpEvtMask,            // pointer to variable to receive event
  LPOVERLAPPED lpOverlapped,    // pointer to overlapped structure
);
```

Parameters

hFile

Handle to the communications device. The **CreateFile** function returns this handle.

lpEvtMask

Pointer to a 32-bit variable that receives a mask indicating the type of event that occurred. If an err occurs, the value is zero; otherwise, it is one of the following values:

Value	Meaning
EV_BREAK	A break was detected on input.
EV_CTS	The CTS (clear-to-send) signal changed state.
EV_DSR	The DSR (data-set-ready) signal changed state.
EV_ERR	A line-status error occurred. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY.
EV_RING	A ring indicator was detected.
EV_RLSD	The RLSD (receive-line-signal-detect) signal changed state.
EV_RXCHAR	A character was received and placed in the input buffer.
EV_RXFLAG	The event character was received and placed in the input buffer. The event character is specified in the device's **DCB** structure, which is applied to a serial port by using the **SetCommState** function.
EV_TXEMPTY	The last character in the output buffer was sent.

lpOverlapped

Pointer to an **OVERLAPPED** structure. This structure is required if *hFile* was opened with FILE_FLAG_OVERLAPPED.

If *hFile* was opened with FILE_FLAG_OVERLAPPED, the *lpOverlapped* parameter must not be NULL. It must point to a valid **OVERLAPPED** structure. If *hFile* was opened with FILE_FLAG_OVERLAPPED and *lpOverlapped* is NULL, the function can incorrectly report that the operation is complete.

If hFile was opened with FILE_FLAG_OVERLAPPED and *lpOverlapped* is not NULL, **WaitCommEvent** is performed as an overlapped operation. In this case, the **OVERLAPPED** structure must contain a handle to a manual-reset event object (created by using the **CreateEvent** function).

If *hFile* handle was not opened with FILE_FLAG_OVERLAPPED, **WaitCommEvent** does not return until one of the specified events or an error occurs.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **WaitCommEvent** function monitors a set of events for a specified communications resource. To set and query the current event mask of a communications resource, use the **SetCommMask** and **GetCommMask** functions.

If the overlapped operation cannot be completed immediately, the function returns FALSE and the **GetLastError** function returns ERROR_IO_PENDING, indicating that the operation is executing in the background. When this happens, the system sets the **hEvent** member of the **OVERLAPPED** structure to the not-signaled state before **WaitCommEvent** returns, and then it sets it to the signaled state when one of the specified events or an error occurs. The calling process can use one of the wait functions to determine the event object's state and then use the **GetOverlappedResult** function to determine the results of the **WaitCommEvent** operation. **GetOverlappedResult** reports the success or failure of the operation, and the variable pointed to by the *lpEvtMask* parameter is set to indicate the event that occurred.

If a process attempts to change the device handle's event mask by using the **SetCommMask** function wh an overlapped **WaitCommEvent** operation is in progress, **WaitCommEvent** returns immediately. The variable pointed to by the *lpEvtMask* parameter is set to zero.

Windows CE: Windows CE supports an additional flag for the *lpEvtMask* parameter:

EV_POWER
Power event, which is generated whenever the device is powered on.

The **WaitCommEvent** function cannot be performed as an overlapped operation. The *lpOverlapped* parameter is not supported and is ignored. It should be set to NULL before calling **WaitCommEvent**.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.
Import Library: Use kernel32.lib.

See Also

Communications Overview, Communication Functions, **CreateFile**, **DCB**, **GetCommMask**, **GetOverlappedResult**, **OVERLAPPED**, **SetCommMask**, **SetCommState**

Communication Structures

The following structures are used with communications devices.

COMMCONFIG
COMMPROP
COMMTIMEOUTS
COMSTAT
DCB
MODEMDEVCAPS
MODEMSETTINGS

COMMCONFIG

The **COMMCONFIG** structure contains information about the configuration state of a communications device.

```
typedef struct _COMM_CONFIG {
    DWORD dwSize;              // size of structure
    WORD  wVersion;            // version of structure
    WORD  wReserved;           // reserved
    DCB   dcb;                 // device-control block
    DWORD dwProviderSubType;   // type of provider-specific data
    DWORD dwProviderOffset;    // offset of provider-specific data
    DWORD dwProviderSize;      // size of provider-specific data
    WCHAR wcProviderData[1];   // provider-specific data
} COMMCONFIG, *LPCOMMCONFIG;
```

Members

dwSize
Specifies the size, in bytes, of the **COMMCONFIG** structure.

wVersion
Specifies the version number of the **COMMCONFIG** structure. This parameter can be 1. The version of the provider-specific structure should be included in the **wcProviderData** member.

wReserved
Reserved; do not use.

dcb
Specifies a device-control block (**DCB**) structure for RS-232 serial devices. A **DCB** structure is always present regardless of the port driver subtype specified in the device's **COMMPROP** structure.

dwProviderSubType
Identifies the type of communications provider, and thus the format of the provider-specific data. F a list of communications provider types, see the description of the **COMMPROP** structure.

dwProviderOffset
Specifies the offset, in bytes, of the provider-specific data relative to the beginning of the structure. This member is zero if there is no provider-specific data.

dwProviderSize
Specifies the size, in bytes, of the provider-specific data.

wcProviderData
Contains the provider-specific data, if any. This member may be of any size or may be omitted. Because the **COMMCONFIG** structure may be expanded in the future, applications should use th **dwProviderOffset** member to determine the location of this member.

Remarks

If the provider subtype is PST_RS232 or PST_PARALLELPORT the **wcProviderData** member is omitted. If the provider subtype is PST_MODEM, the **wcProviderData** member contains a **MODEMSETTINGS** structure.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winbase.h.

See Also

Communications Overview, Communication Structures, **DCB**, **COMMPROP**, **GetCommProperties**, **MODEMSETTINGS**

COMMPROP

The **COMMPROP** structure is used by the **GetCommProperties** function to return information about a given communications driver.

```
typedef struct _COMMPROP {
    WORD  wPacketLength;       // packet size, in bytes
    WORD  wPacketVersion;      // packet version
    DWORD dwServiceMask;       // services implemented
    DWORD dwReserved1;         // reserved
    DWORD dwMaxTxQueue;        // max Tx bufsize, in bytes
    DWORD dwMaxRxQueue;        // max Rx bufsize, in bytes
    DWORD dwMaxBaud;           // max baud rate, in bps
    DWORD dwProvSubType;       // specific provider type
    DWORD dwProvCapabilities;  // capabilities supported
    DWORD dwSettableParams;    // changeable parameters
    DWORD dwSettableBaud;      // allowable baud rates
    WORD  wSettableData;       // allowable byte sizes
    WORD  wSettableStopParity; // stop bits/parity allowed
    DWORD dwCurrentTxQueue;    // Tx buffer size, in bytes
    DWORD dwCurrentRxQueue;    // Rx buffer size, in bytes
    DWORD dwProvSpec1;         // provider-specific data
    DWORD dwProvSpec2;         // provider-specific data
    WCHAR wcProvChar[1];       // provider-specific data
} COMMPROP;
```

Members

wPacketLength
Specifies the size, in bytes, of the entire data packet, regardless of the amount of data requested.

wPacketVersion
Specifies the version of the structure.

dwServiceMask
Specifies a bitmask indicating which services are implemented by this provider. The SP_SERIALCOMM value is always specified for communications providers, including modem providers.

dwReserved1
Reserved; do not use.

dwMaxTxQueue
Specifies the maximum size, in bytes, of the driver's internal output buffer. A value of zero indicate that no maximum value is imposed by the serial provider.

dwMaxRxQueue
Specifies the maximum size, in bytes, of the driver's internal input buffer. A value of zero indicates that no maximum value is imposed by the serial provider.

dwMaxBaud
Specifies the maximum allowable baud rate, in bits per second (bps). This member can be one of th following values:

Value	Meaning
BAUD_075	75 bps
BAUD_110	110 bps
BAUD_134_5	134.5 bps
BAUD_150	150 bps
BAUD_300	300 bps
BAUD_600	600 bps
BAUD_1200	1200 bps
BAUD_1800	1800 bps
BAUD_2400	2400 bps
BAUD_4800	4800 bps
BAUD_7200	7200 bps
BAUD_9600	9600 bps
BAUD_14400	14400 bps
BAUD_19200	19200 bps
BAUD_38400	38400 bps
BAUD_56K	56K bps
BAUD_57600	57600 bps
BAUD_115200	115200 bps
BAUD_128K	128K bps
BAUD_USER	Programmable baud rates available

dwProvSubType

Specifies the specific communications provider type:

Value	Meaning
PST_FAX	FAX device
PST_LAT	LAT protocol
PST_MODEM	Modem device
PST_NETWORK_BRIDGE	Unspecified network bridge
PST_PARALLELPORT	Parallel port
PST_RS232	RS-232 serial port
PST_RS422	RS-422 port
PST_RS423	RS-423 port
PST_RS449	RS-449 port
PST_SCANNER	Scanner device
PST_TCPIP_TELNET	TCP/IP Telnet® protocol
PST_UNSPECIFIED	Unspecified
PST_X25	X.25 standards

dwProvCapabilities

Specifies a bitmask indicating the capabilities offered by the provider. This member can be one of t
following values:

Value	Meaning
PCF_16BITMODE	Special 16-bit mode supported
PCF_DTRDSR	DTR (data-terminal-ready)/DSR (data-set-ready) supporte
PCF_INTTIMEOUTS	Interval time-outs supported
PCF_PARITY_CHECK	Parity checking supported
PCF_RLSD	RLSD (receive-line-signal-detect) supported
PCF_RTSCTS	RTS (request-to-send)/CTS (clear-to-send) supported
PCF_SETXCHAR	Settable XON/XOFF supported
PCF_SPECIALCHARS	Special character support provided
PCF_TOTALTIMEOUTS	Total (elapsed) time-outs supported
PCF_XONXOFF	XON/XOFF flow control supported

dwSettableParams

Specifies a bitmask indicating the communications parameter that can be changed. This member ca be one of the following values:

Value	Meaning
SP_BAUD	Baud rate
SP_DATABITS	Data bits
SP_HANDSHAKING	Handshaking (flow control)
SP_PARITY	Parity
SP_PARITY_CHECK	Parity checking
SP_RLSD	RLSD (receive-line-signal-detect)
SP_STOPBITS	Stop bits

dwSettableBaud

Specifies a bitmask indicating the baud rates that can be used. For values, see the **dwMaxBaud** member.

wSettableData

Specifies a bitmask indicating the number of data bits that can be set. This member can be one of t following values:

Value	Meaning
DATABITS_5	5 data bits
DATABITS_6	6 data bits
DATABITS_7	7 data bits
DATABITS_8	8 data bits
DATABITS_16	16 data bits
DATABITS_16X	Special wide path through serial hardware lines

wSettableStopParity

Specifies a bitmask indicating the stop bit and parity settings that can be selected. This member ca be one of the following values:

Value	Meaning
STOPBITS_10	1 stop bit
STOPBITS_15	1.5 stop bits
STOPBITS_20	2 stop bits
PARITY_NONE	No parity
PARITY_ODD	Odd parity
PARITY_EVEN	Even parity
PARITY_MARK	Mark parity
PARITY_SPACE	Space parity

dwCurrentTxQueue
Specifies the size, in bytes, of the driver's internal output buffer. A value of zero indicates that the value is unavailable.

dwCurrentRxQueue
Specifies the size, in bytes, of the driver's internal input buffer. A value of zero indicates that the value is unavailable.

dwProvSpec1
Specifies provider-specific data. Applications should ignore this member unless they have detailed information about the format of the data required by the provider.

Set this member to COMMPROP_INITIALIZED before calling the **GetCommProperties** functior to indicate that the **wPacketLength** member is already valid.

dwProvSpec2
Specifies provider-specific data. Applications should ignore this member unless they have detailed information about the format of the data required by the provider.

wcProvChar
Specifies provider-specific data. Applications should ignore this member unless they have detailed information about the format of the data required by the provider.

Remarks

The contents of the **dwProvSpec1**, **dwProvSpec2**, and **wcProvChar** members depend on the provider subtype (specified by the **dwProvSubType** member).

If the provider subtype is PST_MODEM, these members are used as follows:

Value	Meaning
dwProvSpec1	Not used.
dwProvSpec2	Not used.
wcProvChar	Contains a **MODEMDEVCAPS** structure.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.

See Also

Communications Overview, Communication Structures, **GetCommProperties**

COMMTIMEOUTS

The **COMMTIMEOUTS** structure is used in the **SetCommTimeouts** and **GetCommTimeouts** functior to set and query the time-out parameters for a communications device. The parameters determine the behavior of **ReadFile**, **WriteFile**, **ReadFileEx**, and **WriteFileEx** operations on the device.

```
typedef struct _COMMTIMEOUTS {
    DWORD ReadIntervalTimeout;
    DWORD ReadTotalTimeoutMultiplier;
    DWORD ReadTotalTimeoutConstant;
    DWORD WriteTotalTimeoutMultiplier;
    DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

Members

ReadIntervalTimeout

Specifies the maximum time, in milliseconds, allowed to elapse between the arrival of two characte on the communications line. During a **ReadFile** operation, the time period begins when the first character is received. If the interval between the arrival of any two characters exceeds this amount, the **ReadFile** operation is completed and any buffered data is returned. A value of zero indicates th interval time-outs are not used.

A value of MAXDWORD, combined with zero values for both the **ReadTotalTimeoutConstant** and **ReadTotalTimeoutMultiplier** members, specifies that the read operation is to return immediately with the characters that have already been received, even if no characters have been received.

ReadTotalTimeoutMultiplier

Specifies the multiplier, in milliseconds, used to calculate the total time-out period for read operations. For each read operation, this value is multiplied by the requested number of bytes to be read.

ReadTotalTimeoutConstant

Specifies the constant, in milliseconds, used to calculate the total time-out period for read operatior For each read operation, this value is added to the product of the **ReadTotalTimeoutMultiplier** member and the requested number of bytes.

A value of zero for both the **ReadTotalTimeoutMultiplier** and **ReadTotalTimeoutConstant** members indicates that total time-outs are not used for read operations.

WriteTotalTimeoutMultiplier

Specifies the multiplier, in milliseconds, used to calculate the total time-out period for write operations. For each write operation, this value is multiplied by the number of bytes to be written.

WriteTotalTimeoutConstant

Specifies the constant, in milliseconds, used to calculate the total time-out period for write operations. For each write operation, this value is added to the product of the **WriteTotalTimeoutMultiplier** member and the number of bytes to be written.

A value of zero for both the **WriteTotalTimeoutMultiplier** and **WriteTotalTimeoutConstant** members indicates that total time-outs are not used for write operations.

Remarks

If an application sets **ReadIntervalTimeout** and **ReadTotalTimeoutMultiplier** to MAXDWORD and sets **ReadTotalTimeoutConstant** to a value greater than zero and less than MAXDWORD, one of the

following occurs when the **ReadFile** function is called:

- If there are any characters in the input buffer, **ReadFile** returns immediately with the characters in the buffer.
- If there are no characters in the input buffer, **ReadFile** waits until a character arrives and then returns immediately.
- If no character arrives within the time specified by **ReadTotalTimeoutConstant**, **ReadFile** times out.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.

See Also

Communications Overview, Communication Structures, **GetCommTimeouts**, **ReadFile**, **ReadFileEx**, **SetCommTimeouts**, **WriteFile**, **WriteFileEx**

COMSTAT

The **COMSTAT** structure contains information about a communications device. This structure is filled b the **ClearCommError** function.

```
typedef struct _COMSTAT {
    DWORD fCtsHold : 1;   // Tx waiting for CTS signal
    DWORD fDsrHold : 1;   // Tx waiting for DSR signal
    DWORD fRlsdHold : 1;  // Tx waiting for RLSD signal
    DWORD fXoffHold : 1;  // Tx waiting, XOFF char received
    DWORD fXoffSent : 1;  // Tx waiting, XOFF char sent
    DWORD fEof : 1;       // EOF character sent
    DWORD fTxim : 1;      // character waiting for Tx
    DWORD fReserved : 25; // reserved
    DWORD cbInQue;        // bytes in input buffer
    DWORD cbOutQue;       // bytes in output buffer
} COMSTAT, *LPCOMSTAT;
```

Members

fCtsHold
Specifies whether transmission is waiting for the CTS (clear-to-send) signal to be sent. If this member is TRUE, transmission is waiting.

fDsrHold
Specifies whether transmission is waiting for the DSR (data-set-ready) signal to be sent. If this member is TRUE, transmission is waiting.

fRlsdHold
Specifies whether transmission is waiting for the RLSD (receive-line-signal-detect) signal to be ser If this member is TRUE, transmission is waiting.

fXoffHold
Specifies whether transmission is waiting because the XOFF character was received. If this membe is TRUE, transmission is waiting.

fXoffSent
Specifies whether transmission is waiting because the XOFF character was transmitted. If this

member is TRUE, transmission is waiting. Transmission halts when the XOFF character is transmitted to a system that takes the next character as XON, regardless of the actual character.

fEof

Specifies whether the end-of-file (EOF) character has been received. If this member is TRUE, the EOF character has been received.

fTxim

If this member is TRUE, there is a character queued for transmission that has come to the communications device by way of the **TransmitCommChar** function. The communications devic transmits such a character ahead of other characters in the device's output buffer.

fReserved

Reserved; do not use.

cbInQue

Specifies the number of bytes received by the serial provider but not yet read by a **ReadFile** operation.

cbOutQue

Specifies the number of bytes of user data remaining to be transmitted for all write operations. This value will be zero for a nonoverlapped write.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.

See Also

Communications Overview, Communication Structures, **ClearCommError**, **ReadFile**, **TransmitCommChar**

DCB

The **DCB** structure defines the control setting for a serial communications device.

```
typedef struct _DCB { // dcb
    DWORD DCBlength;           // sizeof(DCB)
    DWORD BaudRate;            // current baud rate
    DWORD fBinary: 1;          // binary mode, no EOF check
    DWORD fParity: 1;          // enable parity checking
    DWORD fOutxCtsFlow:1;      // CTS output flow control
    DWORD fOutxDsrFlow:1;      // DSR output flow control
    DWORD fDtrControl:2;       // DTR flow control type
    DWORD fDsrSensitivity:1;   // DSR sensitivity
    DWORD fTXContinueOnXoff:1; // XOFF continues Tx
    DWORD fOutX: 1;            // XON/XOFF out flow control
    DWORD fInX: 1;             // XON/XOFF in flow control
    DWORD fErrorChar: 1;       // enable error replacement
    DWORD fNull: 1;            // enable null stripping
    DWORD fRtsControl:2;       // RTS flow control
    DWORD fAbortOnError:1;     // abort reads/writes on error
    DWORD fDummy2:17;          // reserved
    WORD wReserved;            // not currently used
    WORD XonLim;               // transmit XON threshold
    WORD XoffLim;              // transmit XOFF threshold
    BYTE ByteSize;             // number of bits/byte, 4-8
    BYTE Parity;               // 0-4=no,odd,even,mark,space
    BYTE StopBits;             // 0,1,2 = 1, 1.5, 2
    char XonChar;              // Tx and Rx XON character
    char XoffChar;             // Tx and Rx XOFF character
```

```
    char ErrorChar;           // error replacement character
    char EofChar;             // end of input character
    char EvtChar;             // received event character
    WORD wReserved1;          // reserved; do not use
} DCB;
```

Members

DCBlength
Specifies the length, in bytes, of the **DCB** structure.

BaudRate
Specifies the baud rate at which the communications device operates. This member can be an actua baud rate value, or one of the following baud rate indexes:

CBR_110	CBR_19200
CBR_300	CBR_38400
CBR_600	CBR_56000
CBR_1200	CBR_57600
CBR_2400	CBR_115200
CBR_4800	CBR_128000
CBR_9600	CBR_256000
CBR_14400	

fBinary
Specifies whether binary mode is enabled. The Win32 API does not support nonbinary mode transfers, so this member must be TRUE. Using FALSE will not work.

fParity
Specifies whether parity checking is enabled. If this member is TRUE, parity checking is performe and errors are reported.

fOutxCtsFlow
Specifies whether the CTS (clear-to-send) signal is monitored for output flow control. If this memb is TRUE and CTS is turned off, output is suspended until CTS is sent again.

fOutxDsrFlow
Specifies whether the DSR (data-set-ready) signal is monitored for output flow control. If this member is TRUE and DSR is turned off, output is suspended until DSR is sent again.

fDtrControl
Specifies the DTR (data-terminal-ready) flow control. This member can be one of the following values:

Value	Meaning
DTR_CONTROL_DISABLE	Disables the DTR line when the device is opened and leaves it disabled.
DTR_CONTROL_ENABLE	Enables the DTR line when the device is opened and leaves it on.
DTR_CONTROL_HANDSHAKE	Enables DTR handshaking. If handshaking is enabled, it is an error for the application to adjus the line by using the **EscapeCommFunction** function.

fDsrSensitivity
Specifies whether the communications driver is sensitive to the state of the DSR signal. If this member is TRUE, the driver ignores any bytes received, unless the DSR modem input line is high.

fTXContinueOnXoff
Specifies whether transmission stops when the input buffer is full and the driver has transmitted the **XoffChar** character. If this member is TRUE, transmission continues after the input buffer has con

within **XoffLim** bytes of being full and the driver has transmitted the **XoffChar** character to stop receiving bytes. If this member is FALSE, transmission does not continue until the input buffer is within **XonLim** bytes of being empty and the driver has transmitted the **XonChar** character to resume reception.

fOutX

Specifies whether XON/XOFF flow control is used during transmission. If this member is TRUE, transmission stops when the **XoffChar** character is received and starts again when the **XonChar** character is received.

fInX

Specifies whether XON/XOFF flow control is used during reception. If this member is TRUE, the **XoffChar** character is sent when the input buffer comes within **XoffLim** bytes of being full, and th **XonChar** character is sent when the input buffer comes within **XonLim** bytes of being empty.

fErrorChar

Specifies whether bytes received with parity errors are replaced with the character specified by the **ErrorChar** member. If this member is TRUE and the **fParity** member is TRUE, replacement occu

fNull

Specifies whether null bytes are discarded. If this member is TRUE, null bytes are discarded when received.

fRtsControl

Specifies the RTS (request-to-send) flow control. If this value is zero, the default is RTS_CONTROL_HANDSHAKE. This member can be one of the following values:

Value	Meaning
RTS_CONTROL_DISABLE	Disables the RTS line when the device is opened ar leaves it disabled.
RTS_CONTROL_ENABLE	Enables the RTS line when the device is opened an leaves it on.
RTS_CONTROL_HANDSHAKE	Enables RTS handshaking. The driver raises the RT line when the "type-ahead" (input) buffer is less tha one-half full and lowers the RTS line when the buff is more than three-quarters full. If handshaking is enabled, it is an error for the application to adjust th line by using the **EscapeCommFunction** function.
RTS_CONTROL_TOGGLE	Specifies that the RTS line will be high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line will be low.

fAbortOnError

Specifies whether read and write operations are terminated if an error occurs. If this member is TRUE, the driver terminates all read and write operations with an error status if an error occurs. Th driver will not accept any further communications operations until the application has acknowledg the error by calling the **ClearCommError** function.

fDummy2

Reserved; do not use.

wReserved

Not used; must be set to zero.

XonLim

Specifies the minimum number of bytes allowed in the input buffer before the XON character is se

XoffLim

Specifies the maximum number of bytes allowed in the input buffer before the XOFF character is sent. The maximum number of bytes allowed is calculated by subtracting this value from the size, i bytes, of the input buffer.

ByteSize

Specifies the number of bits in the bytes transmitted and received.

Parity

Specifies the parity scheme to be used. This member can be one of the following values:

Value	Meaning
EVENPARITY	Even
MARKPARITY	Mark
NOPARITY	No parity
ODDPARITY	Odd
SPACEPARITY	Space

StopBits

Specifies the number of stop bits to be used. This member can be one of the following values:

Value	Meaning
ONESTOPBIT	1 stop bit
ONE5STOPBITS	1.5 stop bits
TWOSTOPBITS	2 stop bits

XonChar

Specifies the value of the XON character for both transmission and reception.

XoffChar

Specifies the value of the XOFF character for both transmission and reception.

ErrorChar

Specifies the value of the character used to replace bytes received with a parity error.

EofChar

Specifies the value of the character used to signal the end of data.

EvtChar

Specifies the value of the character used to signal an event.

wReserved1

Reserved; do not use.

Remarks

When a **DCB** structure is used to configure the 8250, the following restrictions apply to the values specified for the **ByteSize** and **StopBits** members:

- The number of data bits must be 5 to 8 bits.
- The use of 5 data bits with 2 stop bits is an invalid combination, as is 6, 7, or 8 data bits with 1.5 stop bits.

QuickInfo

Windows NT: Requires version 3.1 or later.
Windows: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winbase.h.

See Also

Communications Overview, Communication Structures, **BuildCommDCB**, **ClearCommError**, **EscapeCommFunction**, **GetCommState**, **SetCommState**

MODEMDEVCAPS

The **MODEMDEVCAPS** structure contains information about the capabilities of a modem.

```
typedef struct modemdevcaps_tag {
    DWORD dwActualSize;            // size of returned data, in bytes
    DWORD dwRequiredSize;          // total size of structure
    DWORD dwDevSpecificOffset;     // offset of provider-defined data
    DWORD dwDevSpecificSize;       // size of provider-defined data
    DWORD dwModemProviderVersion;     // provider version number
    DWORD dwModemManufacturerOffset;  // offset of manufacturer name
    DWORD dwModemManufacturerSize;    // length of manufacturer name
    DWORD dwModemModelOffset;         // offset of model name
    DWORD dwModemModelSize;           // length of model name
    DWORD dwModemVersionOffset;       // offset of version name
    DWORD dwModemVersionSize;         // length of version name
    DWORD dwDialOptions;              // bitmap of supported values
    DWORD dwCallSetupFailTimer;       // maximum in seconds
    DWORD dwInactivityTimeout;        // maximum in tenths of seconds
    DWORD dwSpeakerVolume;            // bitmap of supported values
    DWORD dwSpeakerMode;              // bitmap of supported values
    DWORD dwModemOptions;             // bitmap of supported values
    DWORD dwMaxDTERate;               // maximum value in bit/s
    DWORD dwMaxDCERate;               // maximum value in bit/s
    BYTE abVariablePortion [1];       // variable-length data
} MODEMDEVCAPS, *PMODEMDEVCAPS, *LPMODEMDEVCAPS;
```

Members

dwActualSize
Specifies the size, in bytes, of the data actually returned to the application. This member may be les than the **dwRequiredSize** member, if an application did not allocate enough space for the variable- length portion of the structure.

dwRequiredSize
Specifies the number of bytes required for the entire **MODEMDEVCAPS** structure, including the variable-length portion.

dwDevSpecificOffset
Specifies the offset of the provider-defined portion of the structure, in bytes relative to the beginnir of the structure.

dwDevSpecificSize
Specifies the size of the provider-defined portion of the structure, in bytes.

dwModemProviderVersion
Specifies the version of the service provider. The format and use of this member depends on the service provider.

dwModemManufacturerOffset
Specifies the offset of a text string that contains the name of the modem manufacturer. The offset is specified in bytes relative to the beginning of the structure.

dwModemManufacturerSize
Specifies the length of the modem manufacturer name, in bytes. The string is not null-terminated.

dwModemModelOffset
Specifies the offset of a text string that contains the model of the modem. The offset is specified in bytes relative to the beginning of the structure.

dwModemModelSize
Specifies the length of the model name, in bytes. The string is not null-terminated.

dwModemVersionOffset
Specifies the offset of a text string that gives the version and revision of the attached modem, if the provider could determine the information. The offset is specified in bytes relative to the beginning the structure.

dwModemVersionSize
Specifies the length of the modem version string, in bytes. The string is not null-terminated.

dwDialOptions

Specifies dialing options that are supported by the modem device. This member can be zero or mor of the following values:

Value	Meaning
DIALOPTION_DIALBILLING	Specifies that the modem supports waiting for billing tone (bong).
DIALOPTION_DIALQUIET	Specifies that the modem supports waiting for quie
DIALOPTION_DIALDIALTONE	Specifies that the modem supports waiting for a di tone.

dwCallSetupFailTimer

Specifies the maximum call setup timeout supported by the modem, in seconds. This is the largest value that can be specified for the corresponding member of the **MODEMSETTINGS** structure.

dwInactivityTimeout

Specifies the maximum inactivity timeout supported by the modem, in tenths of seconds. This is th largest value that can be specified for the corresponding member of the **MODEMSETTINGS** structure.

dwSpeakerVolume

Specifies the speaker volume settings supported by the modem. This member can be zero or more the following values:

Value	Meaning
MDMVOLFLAG_LOW	The modem supports low (MDMVOL_LOW) volume.
MDMVOLFLAG_MEDIUM	The modem supports medium (MDMVOL_MEDIUM) volume.
MDMVOLFLAG_HIGH	The modem supports high (MDMVOL_HIGH) volume

dwSpeakerMode

Specifies the speaker mode settings supported by the modem. This member can be zero or more of the following values:

Value	Meaning
MDMSPKRFLAG_OFF	The modem supports the MDMSPKR_OFF speaker mode.
MDMSPKRFLAG_DIAL	The modem supports the MDMSPKR_DIAL speaker mode.
MDMSPKRFLAG_ON	The modem supports the MDMSPKR_ON speake mode.
MDMSPKRFLAG_CALLSETUP	The modem supports the MDMSPKR_CALLSETUP speaker mode.

dwModemOptions

Specifies supported modem options. This member can be zero or more of the following values:

MDM_BLIND_DIAL	MDM_FLOWCONTROL_SOFT
MDM_CCITT_OVERRIDE	MDM_FORCED_EC
MDM_CELLULAR	MDM_SPEED_ADJUST
MDM_COMPRESSION	MDM_TONE_DIAL
MDM_ERROR_CONTROL	MDM_V23_OVERRIDE
MDM_FLOWCONTROL_HARD	

When **MODEMDEVCAPS** is used to set modem options, as part of the **MODEMSETTINGS** structure these values are used as follows:

Value	Meaning
MDM_CCITT_OVERRIDE	When set, CCITT modulations are enabled for V.21 and V.22 or V.23. When clear, bell modulations are enabled for 103 an 212A.
MDM_V23_OVERRIDE	When set, CCITT modulations are enabled for V.23. When clear, CCITT modulations are enabled for V.2 and V.22.

For V.23 to be set, both MDM_CCITT_OVERRIDE and MDM_V23_OVERRIDE must be set.

dwMaxDTERate
Maximum DTE rate in bits per second.
dwMaxDCERate
Maximum DCE rate in bits per second.
abVariablePortion
Contains variable-length information, including strings and any provider-defined information.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in mcx.h.

See Also

Communications Overview, Communication Structures, **MODEMSETTINGS**

MODEMSETTINGS

The **MODEMSETTINGS** structure contains information about a modem's configuration.

```
typedef struct modemsettings_tag {
    DWORD dwActualSize;            // size of returned data, in bytes
    DWORD dwRequiredSize;          // total size of structure
    DWORD dwDevSpecificOffset;     // offset of provider-defined data
    DWORD dwDevSpecificSize;       // size of provider-defined data

    // Static local options (read/write)
    DWORD dwCallSetupFailTimer;      // call setup timeout, in seconds
    DWORD dwInactivityTimeout;       // timeout, in tenths of seconds
    DWORD dwSpeakerVolume;           // speaker volume level
    DWORD dwSpeakerMode;             // speaker mode
    DWORD dwPreferredModemOptions;   // preferred options

    // negotiated options (read only) for current or last call
    DWORD dwNegotiatedModemOptions;  // bitmap specifying options
    DWORD dwNegotiatedDCERate;       // DCE rate, in bits per second

    // Variable portion for proprietary expansion
    BYTE  abVariablePortion[1];      // variable-length data
} MODEMSETTINGS, *PMODEMSETTINGS, *LPMODEMSETTINGS;
```

Members

dwActualSize
Specifies the size, in bytes, of the data actually returned to the application. This member may be les than the **dwRequiredSize** member if an application did not allocate enough space for the variable-length portion of the structure.

dwRequiredSize
Specifies the number of bytes required for the entire **MODEMDEVCAPS** structure, including the variable-length portion.

dwDevSpecificOffset
Specifies the offset of the provider-defined portion of the structure, in bytes relative to the beginnir of the structure.

dwDevSpecificSize
Specifies the size of the provider-defined portion of the structure, in bytes.

dwCallSetupFailTimer
Specifies the maximum number of seconds the modem should wait, after dialing is completed, for indication that a modem-to-modem connection has been established. If a connection is not established in this interval, the call is assumed to have failed. This member is equivalent to register in Hayes® compatible modems.

dwInactivityTimeout
Specifies the maximum number of seconds of inactivity allowed after a connection is established. I no data is either transmitted or received for this period of time, the call is automatically terminated. This time-out is used to avoid excessive long distance charges or online service charges if an application unexpectedly locks up or the user leaves.

dwSpeakerVolume
Specifies the volume level of the monitor speaker when the speaker is on. This member can be one the following values:

Value	Meaning
MDMVOL_LOW	Low volume.
MDMVOL_MEDIUM	Medium volume.
MDMVOL_HIGH	High volume.

The **MODEMDEVCAPS** structure specifies the speaker volumes a modem supports. Actual volumes are hardware-specific.

dwSpeakerMode
Specifies when the speaker should be on. This member can be one of the following values:

Value	Meaning
MDMSPKR_OFF	The speaker is always off.
MDMSPKR_CALLSETUP	The speaker is on until a connection is established.
MDMSPKR_ON	The speaker is always on.
MDMSPKR_DIAL	The speaker is on until a connection is established, except that it is off while the modem is actually dialing.

dwPreferredModemOptions
Specifies the modem options requested by the application. The local and remote modems negotiate modem options during call setup; this member specifies the initial negotiating position of the local modem.

The **dwModemOptions** member of the **MODEMDEVCAPS**structure specifies the modem option supported by the local modem. For a list of modem options, see the description of the **MODEMDEVCAPS** structure.

dwNegotiatedModemOptions

Specifies the modem options that are actually in effect. This member is filled in after a connection established and the local and remote modems negotiate modem options.

The **dwModemOptions** member of the **MODEMDEVCAPS** structure specifies the modem optio supported by the local modem. For a list of modem options, see the description of the **MODEMDEVCAPS** structure.

dwNegotiatedDCERate

Specifies the DCE rate that is in effect. This member is filled in after a connection is established an the local and remote modems negotiate modem modulations.

abVariablePortion

Contains provider-defined information, if any.

QuickInfo

Windows NT: Requires version 4.0 or later.
Windows: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in mcx.h.

See Also

Communications Overview, Communication Structures, **MODEMDEVCAPS**