Wildcat Software

WinDLL Fastrack: Communications Project

For information on how to use help
choose Help \Using Help.

WinDLL Fastrack: Windows Communications Functions

Function Declarations

Device Control Block Functions

DCB Structure

BuildCommDCB
SetCommState
GetCommState

Communications Control Functions

OpenComm
CloseComm
EscapeCommFunction
SetCommBreak
ClearCommBreak

Communications I/O Functions

ReadComm
WriteComm
FlushComm
TransmitComm
UngetCommChar

Communications Status

GetCommErr
ComStat Structure

Monitoring Communications Events

SetCommEventMask
GetCommEventMask

Communications Project Introduction

We have found using windows communications easier than standard basic. Queue buffers and flow control are accomplished automatically by the windows operating environment. Windows controls a communications device using a Device Control Block which we can update or view during run time. Load the Windows Terminal Program, and observe how Microsoft sets up a communications device with the Device Control Block module of the example project.

Communications Example Modules

Device Control Block: Demonstrates working with the Device Control Block (DCB) structure.
Communications Control: Opening \ Closing communication devices. Setting signal lines state.
Communications I/O: Transmitting and receiving data from a communications device. The Example code is not adequate to use as a terminal, although if you have a modem, use the example functions to transmit and receive modem control messages.
Communication Status: Explains the communications status (ComStat) structure, and Bit Flags for the current device return status.
Communications Events: Sets and monitors event triggers that can be trapped.

!! CLOSE ANY DEVICE YOU OPEN PRIOR TO EXITING THE PROGRAM!!

Device Control Block functions:

BuildCommDCB: modify DCB using a mode state.
GetCommState: Get current DCB values.
SetCommState: Set DCB values.

Communications Control functions:

OpenComm: Open a communications device.
CloseComm: Close a communications device.
SetCommBreak: Set the BREAK signal.
ClearCommBreak: Clear the BREAK signal.
EscapeCommFunction: Set other signal lines states.

Communications I/O functions:

ReadComm: Read data from Receive Queue.
WriteComm: Write data to Transmit Queue.
TransmitCommChar: Send char to head of Transmit Queue.
UngetCommChar: send char to head of Receive Queue.
FlushComm: Flush Transmit \ Receive Queues.

Communication Status function:

GetCommError: Current state, Queue sizes and Error status.

Communications Events functions:

SetCommEventMask: Set event masks to monitor.
GetCommEventMask: Get event occurrence status.

'**Windows API Communications Declarations**
'**Note:** Some of these functions require data structures. DCB & ComStat
'**Note:** Expand this view to avoid word wrapping before **COPY** ing.

```
Declare Function BuildCommDCB Lib "User" (ByVal lpDEF$, lpDCB As DCB) As Integer
Declare Function ClearCommBreak Lib "User" (ByVal nCid%) As Integer
Declare Function CloseComm Lib "User" (ByVal ComPort%) As Integer
Declare Function EscapeCommFunction Lib "User" (ByVal nCid%, ByVal nFunc%) As Integer
Declare Function FlushComm Lib "User" (ByVal nCid%, ByVal nQueue%) As Integer
Declare Function GetCommError Lib "User" (ByVal nCid%, lpStat As ComStat) As Integer
Declare Function GetCommEventMask Lib "User" (ByVal nCid%, ByVal nEvtMask%) As
Integer
Declare Function GetCommState Lib "User" (ByVal nCid%, lpDCB As DCB) As Integer
Declare Function OpenComm Lib "User" (ByVal lpComName$, ByVal wInQueue%, ByVal
wOutQueue%) As Integer
Declare Function ReadComm Lib "User" (ByVal nCid%, ByVal lpBuf$, ByVal nSize%) As
Integer
Declare Function SetCommBreak Lib "User" (ByVal nCid%) As Integer
Declare Function SetCommEventMask Lib "User" (ByVal nCid%, ByVal nEvtMask%) As Integer
Declare Function SetCommState Lib "User" (lpDCB As DCB) As Integer
Declare Function TransmitCommChar Lib "User" (ByVal nCid%, ByVal cChar%) As Integer
Declare Function UngetCommChar Lib "User" (ByVal nCid%, ByVal cChar%) As Integer
Declare Function WriteComm Lib "User" (ByVal nCid%, ByVal lpBuf$, ByVal nSize%) As
Integer
```

BuildCommDCB (lpDef$, lpDCB) as Integer

Uses the a DOS MODE type string ***lpDef$*** to modify the device control block structure ***lpDCB*** values for baudrate, parity, data and stop bits.

Returns 0 if successful.

Example DCB Structure

GetCommState (nCid%, lpDCB) as integer

Puts the device control block data of device specified by ***nCid%*** into the structure ***lpDCB***

Returns 0 if successful.

Example DCB Structure

SetCommState (lpDCB) as integer

Sets device control block specified by the ID field to values in the DCB structure ***lpDCB***.

Returns 0 if successful.

Example DCB Structure

'Device Control Block (DCB) Structure

```
Type DCB
        ID   As String * 1
        BaudRate   As Integer
        ByteSize   As String * 1
        Parity   As String * 1          'Parity Values
        Stopbits   As String * 1        'Stop Bit Values

        RlsTimeout   As Integer
        CtsTimeout   As Integer
        DsrTimeout   As Integer

'Bit wise controls two bytes with Bit Flag Names

        BitWise1   As String * 1
        BitWise2   As String * 1

        XonChar   As String * 1
        XoffChar   As String * 1
        XonLim   As Integer
        XoffLim   As Integer

        PeChar   As String * 1
        EofChar   As String * 1
        EvtChar   As String * 1
        TxDelay   As Integer

End Type
```

DCB BitWise 1 Flags

fBinary : 1 = 1
fRtsDisable : 1 = 2
fParity : 1 = 4
fOutCtsFlow : 1 = 8
fOutxDsrFlow : 1 = 16
fDummy : 2 = 32 & 64
fDtrDisable : 1 = 128

DCB BitWise 2 Flags

fOutX : 1 = 1
fInX : 1 = 2
fPeChar : 1 = 4
fNull : 1 = 8
fChEvt : 1 = 16
fDtrFlow : 1 = 32
fRtsFlow : 1 = 64
fDummy2 : 1 = 128

Parity Value Flags

0 = None
1 = Odd
2 = Even
3 = Mark
4 = Space

StopBit Value Flags

0 = 1 Stop bit
1 = 1.5 Stop bits
2 = 2 Stop bits

OpenComm (lpCommName$, wInQueue%, wOutQueue%) as integer

Opens the communications device lpCommName$ and sets the returns the device id. Sets Receive Queue buffer size to ***wInQueue%*** bytes, and Transmit Queue buffer size to ***wOutQueue%*** bytes.

Returns: Device ID or Negative if not successful.

Example Error Codes Device ID

OpenComm Return Errors

-1	Invalid Device ID
-2	Already open.
-3	Not Opened.
-4	Unable to allocate queues.
-5	Error in parameters
-10	Hardware not present.
-11	Invalid Data or Stop bit values.
-12	Invalid baudrate.

CloseComm (nCid%) as integer

Closes the communications device ***nCid%*** after transmitting what is in the queue. Also frees allocated queue space.

Returns 0 if successful.

Example

SetCommBreak (nCid%) as integer

Sets transmission line in break state until ClearCommBreak function is called for device identified by ***nCid%***.

Returns 0 if successful.

Example

ClearCommBreak (nCid%) as integer

Removes transmission line break state for device identified by ***nCid%***.

Returns 0 if successful.

Example

EscapeCommFunction (nCid%, nFunc%) as integer

Specifies extended functions ***nFunc%*** for device ***nCid%***.

Returns 0 if successful.

Example

FlushComm (nCid%, nQueue%) as integer

Flushes the Queue nQueue% for device nCid%.

Returns 0 if successful.

Example

ReadComm (nCid%, lpBuf$, nSize%) as integer

Copies the number of characters specified by ***nSize%*** from the ***nCid%*** device into the buffer ***lpBuf$***.

Returns number of characters read. Negative number indicates an error, *Abs(return%) = number of characters read*. Use GetCommError function to determine cause of error. The return value for parallel ports will be 0.

Example

WriteComm (nCid%, lpBuf$, nSize%) as integer

Writes the number of characters specified by ***nSize%*** to the ***nCid%*** device from the buffer ***lpBuf$***. Could delete data in the queue if there is not enough space. Use GetCommError to determine space, OpenComm to allocates queue space.

Returns number of characters written. Negative number indicates an error, Abs(return%) = number of characters sent. Use GetCommError function to determine cause of error.

Example

TransmitCommChar (nCid%, cChar%) as integer

Places the character ***cChar%*** at the head of the transmit queue of device ***nCid%*** for immediate transmission.

Returns 0 if successful.
Example Using BOOL, BYTE and Char Data

UngetCommChar (nCid%, cChar%) as integer

Places the character specified ***cChar%*** in the receive queue of device ***nCid%*** to be the next character to be read from the queue. Can not make consecutive calls to *UngetCommChar*.

Example Using BOOL, BYTE and Char Data

GetCommError (nCid%, lpStat) as integer

Clears lock placed on the communications port when an error occurs. Places the current status of the device ***nCid%*** in the structure ***lpStat***. Also returns all error codes occurring since last GetCommError call.

Returns 0 if no error occurred or Bitwise Error Code

Example Working with Bitwise Data

```
'Communications Status Structure
        used by GetCommError
Type ComStat

        StatusByte   As String * 1 'Bitwise status
        cbInQue    As Integer                '# chars in receive Q
        cbOutQue    As Integer               '# chars in transmit Q

End Type
```

ComStat StatusByte Flags

fCtsHold	: 1 = 1
fDsrHold	: 1 = 2
fRlsHold	: 1 = 8
fXoffHold	: 1 = 16
fXoffSent	: 1 = 32
fEOF	: 1 = 64
fTxim	: 1 = 128

GetCommError Return Codes

1	Receive queue overflow
2	Overrun, lost character
4	Hardware parity error
8	Hardware framing error
16	Hardware break detected
32	Clear-to-send timeout
64	Data-set-ready timeout
128	Receive-line-signal timeout
256	Transmit queue is full
512	Parallel device timeout
1024	Parallel device I/O error
2048	Parallel device not selected
4096	Parallel device out of paper
32678	Invalid mode or nCid value

SetCommEventMask (nCid%, nEvtMask%) as integer

Enables and retrieves the event mask for device ***nCid%***. ***nEvtMask%*** bits define which events will be enabled.

Returns Bitwise event mask. Occurrence of an event is a bit = 1.

Example Working with Bitwise Data

GetCommEventMask (nCid%, nEvtMask%) as integer

Returns the event mask for device ***nCid%*** and clears the mask. Enabled events are returned in ***nEvtMask%***. Event values are displayed in the project examples.

Returns Bitwise value of current events. Occurrence of an event is a bit = 1.

Example Working with Bitwise Data

cChar Character to be placed in transmit or receive queue.

lpBuf$ String used as communications buffer. Be sure to allocate the space designated by the **nSize%** parameter if Windows will write to this buffer!
ie: lpBuf$ = Space$(nSize%)

lpCommName$ String containing the communication device name . Formatted COMn or LPTn.

lpDef$ Control information string. Format as DOS MODE command.
ie :"COMn:9600,e,7,2"

lpDCB Long pointer to the data structure [DCB] for working with Windows' Device Control Block information. Declared as **'lpDCB as DCB'** in the example code.

lpStat Long Pointer to the structure (COMSTAT) which receives the device status, declared as '**lpStat as COMSTAT**' in the example code.

nCid% Communications device identification. Value of nCid% is returned by the OpenComm function.

nEvtMask% - event Bit Flags and Values

Bit	Value	Title
0	1	Receive any
1	2	Receive specific
2	4	Transmit empty
3	8	Clear-to-send changes state
4	16	Data-set-ready changes state
5	32	Receive-line-signal-detect changes state
6	64	Break received
7	128	Line status error (frame, overrun, parity)
8	256	Ring signal detect
9	512	Printer error

nFunc extended function codes.

1 = Act as if Xoff character received
2 = Act as if Xon character received
3 = Send request to send signal
4 = Clear request-to-send signal
5 = Send data-terminal-ready signal
6 = Clear data-terminal-ready signal
7 = Reset device (when possible)

nQueue% - Specifies Queue. 0 = flush the transmit queue. 1 = flush the receive queue.

nSize% - Specifies the number of characters in a string variable. If Windows writes to the String BE SURE it is at least **nSize%** bytes in length to avoid **Unrecoverable Application Error(s).**

wInQueue% specifies the size of the receiving queue.

wOutQueue% specifies the size of the transmitting queue.

Sub BuildCommDCBButton_Click ()

Source code is shipped with registered disks...

```
Sub GetCommStateButton_Click ()
        'Get Device Control Block Information

        Source code is shipped with registered disks..
```

```
Sub SetCommStateButton_Click ()
        'Reset the Device Control Block to values in DCB structure

        Source code is shipped with registered disks..
```

```
Sub DisplayDCB ()
        'Display the values for current DCB structure

        Source code is shipped with registered disks...
```

```
Sub Bit1FlagIN ()
        'Get Bitwise Flags for Bitwise Byte 1

        Source code is shipped with registered disks...
```

Sub OpenCommButton_Click ()

Source code is shipped with registered disks...

Sub CloseCommButton_Click ()

Source code is shipped with registered disks...

Sub SetCommBreakButton_Click ()

Source code is shipped with registered disks...

Sub ClearCommBreakButton_Click ()

Source code is shipped with registered disks...

Sub EscapeCommFunctionButton_Click ()

Source code is shipped with registered disks...

```
Sub FlushCommButton_Click ()
        'Flush a communications queue

        Source code is shipped with registered disks...
```

```
Sub ReadCommButton_Click ()
        'Read Data from Receive Queue

        Source code is shipped with registered disks..
```

```
Sub TransmitCommCharButton_Click ()
        'Force character to top of Transmit Queue

        Source code is shipped with registered disks...
```

```
Sub UngetCommCharButton_Click ()
        'Write a character to the Receive Queue
```

Source code is shipped with registered disks...

Sub WriteCommButton_Click ()

Source code is shipped with registered disks...

Sub GetCommEventMaskButton_Click ()

Source code is shipped with registered disks...

Sub SetCommEventMaskButton_Click ()

Source code is shipped with registered disks...

```
Sub GetCommErrorButton_Click ()
        'Get Communications Error Status

        Source code is shipped with registered disks...
```

Sub GCE_Status_Change ()

'Evaluate Communication Status Return

Source code is shipped with registered disks...

Sub lpStat_StatusByte_Change ()

'Evaluate ComStat Status Byte Flags

Source code is shipped with registered disks...

Wildcat Software

WinDLL Fastrack: Programming Notes

For information on how to use help:
choose Help - Using Help.

Registering this disk:

Why should YOU register,

You get the **most current version** of this disk.(We have made improvements!)
You get the source code for the WinDLL programs.
All following updates are only $10.00
You are notified of changes to your disk and about new programmers tools.

Suggested registration price: $19

Wildcat Software
PO Box 2607
Cheyenne, Wyoming 82003
Attn: Windll Fastrack

We welcome any suggestions that will help improve this program, please feel free to write or contact us on CompuServe. Our CompuServe Id is 76675,122.

The Window Dynamic Link Libraries

Visual Basic DLL declarations require that we state the Dynamic Link Library where the function is located. There apparently are 4 Windows function libraries: **Kernel**, **User**, **System** and the **GDI**.
If you wish to experiment with functions not covered in this release, try referencing one of those libraries.

Windows Data Types and Visual Basic Equivalents

The following table lists the Windows data type with respect to using Windows function calls. The **VB Parameter** list recommended types to use as a function parameter or return type. Use the **VB Structure** type in structures that the Windows DLL will access.

Windows	VB Parameter	VB Structure
BOOL	Integer (AND)	String * 1
BYTE	Integer (AND)	String * 1
char	Integer (AND)	String * 1
dWord	Long	Long
HANDLE	Integer	Integer
int	Integer	Integer
LONG	Long	Long
LPSTR	String ($)	String * *N*
short	Integer	Integer
void	non-TYPE*	- -
WORD	Integer (+)	Integer (+)

See also: Naming conventions; Microsoft Windows Programmers Reference.

Naming conventions: Microsoft Windows Programmers Reference.

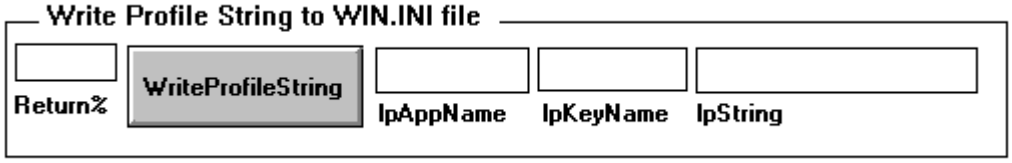
The naming conventions used for parameter names in the Microsoft Windows Programmers Reference were retained in the sample code regardless of data type conversions for Visual Basic variables.

Mircrosoft's parameter names use an italic prefix to indicate the parameters data type. Following is a list of Mircrosofts Prefixes, Data Types and resulting Visual Basics type.

Prefix	Type	Visual Basic Type	Example
b	BOOL	Integer	bStat%
c	BYTE	Integer	cDriveLetter%
c	char	Integer	cChar%
dw	LONG	Long	dwFlag
f	bit flags	Bitwise Character	String*1 or Integer
h	HANDLE	Integer	chWnd%
l	LONG	Long	lParam
lp	LongPointer	String ($)	lpAppName$
n	Short	Integer	nSize%
p	Short	Integer	pMsg
w	Short	Integer	wUnique%

Naming Conventions:
See Also: Naming conventions used in Mircrosofts Windows Programmers Reference.

The sample programs are oriented to give you a quick understanding of the Windows functions without forcing you to dissect elaborate program code. Most of the functions are designed to operate as separate entities, although they are assembled in groups where they can be used together. Each function is displayed as a named command button and associated parameter fields.



The sample above shows a typical function example. To test this example you would supply the field parameters **lpAppName**, **lpKeyName** and **lpString**. Clicking the **WriteProfileString** command button would execute the function with your supplied values. The source code for this function would be found in the subroutine **WriteProfileStringButton_Click().** The the controls containing the supplied parameters are named using the capitol letters of the function name followed by an underscore "_" and the parameter name. (**WPS_lpAppName, WPS_lpKeyName, WPS_lpString and WPS_Return**)

The subroutine, prior to calling the function, converts all the parameters to the proper data type, *using only local variables, except where data structures are used.*
ie:

```
lpAppName$ = WPS_lpAppName.Text
lpKeyName$ = WPS_lpKeyName.Text
lpString$          = WPS_lpString.Text

ret% = WriteProfileString(lpAppName$, lpKeyName$, lpString$)

WPS_Return.Text = Str$(ret%)
```

Of course, you find the sample code a little more complicated than the above example, but we kept it as simple as possible while trying to avoid execution errors.

Unrecoverable Application Errors

Making a Dynamic Link Library call removes us from Visual Basics safety blanket and errors can crash the Windows Operating Environment. Save your program prior to testing it, or risk the AGONY OF DELETE.

While writing this code we caused Unrecoverable Application Errors in two ways.

FIRST METHOD: Using an undefined parameter in a function call.

Visual Basic does not require us to define variables prior to their being used. This can be a problem if we begin to make calls outside the Visual Basic operating environment. If a Windows function returns a value to one of its parameters, we MUST create that parameter prior to calling the function. If the parameter is a string BE SURE IT IS AT LEAST ONE CHARACTER IN LENGTH. Windows does not like basic's null length strings. If the function requests the length of a parameter string, BE SURE THE STRING IS AT LEAST AS LONG AS YOU SAY IT IS.

SECOND METHOD: Not declaring a function return type.

This error caused a hour of confusion for us one day. Every Windows function returns a value which is 'typed' in the function declaration.

i.e. Declare Function GetFocus Lib "Kernel" ( ) as Integer

Not having the 'as Integer' type following the statement would have caused a runtime error, if my program hadn't caused a Unrecoverable Application Error first. This CRASH can be knarly to find because the 'as type' part of the declaration is usually not in view on the edit screen.

Working with Bitwise Data

A quick refresher course on bitwise operations.

Bit operations: Many of the Windows DLL's return values should be read as a Bit Flags. Listed below are eight possible bit flags and values.

Bit Position	Byte	Value	Basic Exponential
0	0000 0001	1	2^0
1	0000 0010	2	2^1
2	0000 0100	4	2^2
3	0000 1000	8	2^3
4	0001 0000	16	2^4
5	0010 0000	32	2^5
6	0100 0000	64	2^6
7	1000 0000	128	2^7

If more than one bit flag is set in the byte the value becomes the sum of the flag values.

Example: If Bit #1, Bit #5 and Bit #6 were set then
Byte is 0110 0010
Value is 2 + 32 + 64 = 98

The basic 'AND' operator allows us to test for Bit Flags.

Example: Testing Byte 0110 0010 = 98

Byte Value	AND	Test Value	= Result	Bit Flag Set
98	AND	1	= 0	No
98	AND	2	= 2	Yes
98	AND	4	= 0	No
98	AND	8	= 0	No
98	AND	16	= 0	No
98	AND	32	= 32	Yes
98	AND	64	= 64	Yes
98	AND	128	= 0	No

The basic exponential allows a fast bit map testing

Example:

Program..

```
ByteVal = 98
For bit = 0 to 7
        If ByteVal AND 2^bit Then Print "Bit "; bit; " set."
Next
```

Prints...

```
Bit 2 set.
Bit 5 set.
Bit 6 set.
```

Sending and Receiving the BYTE, BOOL & Char data types.

The C language BYTE, BOOL & Char data types are one byte variables not supported by Visual Basic, but there is a work around. Sending BYTE data is quite easy since the you can pass any BYTE variable as an integer. (the smallest object that can be 'stacked' in the PC)

Receiving a BYTE result is a little more tricky. Keep in mind that you are receiving an integer with only one byte of valid information. We worked our way around this by using the 'And' operator with an integer equal to 255.

Example: From the GetTempDrive* function that returns a temporary drive letter as a BOOL.

```
Problem:   We expect a return   value range of 0 to 255 for a BOOL
           But instead we get a  the return value;    ret% = 14915

Solution:  tmpDrive% = ret% And 255        'AND' the return with 255
           ? tmpDrive%                              'prints   "67"
           ? Chr$(tmpDrive%)                      ' prints   "C"
```

This example only deals with a single byte. Integer bit flags are occasionally used by the windows routines. You can expect to get a Long with them, also.

The GetTempDrive function is in WinDLL's example code project WIN_SYS.

AND the return value of this parameter with 255, see the section on ***BYTE, BOOL & Char data types***.

We prefer the BASIC **String$** type for parameters, remember to allocate sufficient space for the return string. If Windows writes to the variable, remember that the last character in the string will be a null.

For Structures we must use the VB **String * n** Type. If you use String * n types for parameters remember that the last character in the string is a null. Make **n** equal to the length of the longest expected return string, + 1, for the null character.

A **WORD** is an unsigned integer. If you operate on WORD variables, remember that negative integer values are greater that 32767. Passing a negative integer as a WORD is viewed as an unsigned integer by Windows.

A **Void** is a return only parameter. Declaring a function without the **'AS *TYPE*'** is equivalent to a void Windows function.

hWnd is a reserved name in Visual Basic so we substituted **chWnd** (*c*ontrol *h*andle)

Visual Basic does not support bitflag operations see the section:***Working with Bitwise Data.***