IBD189 ASCII Command Set

Users Manual

Version 1.0

2012/10/03

1.0 Introduction

The IBD189 is an USB to CAN adapter with VCP (Virtual COM Port) serial interface. Simply connect it to any PC running Windows XP/Vista/Windows 7, and “talk” with the module in standard ASCII format.
The IBD189 handles both the 11bit ID format (standard) as well as the 29bit ID format (extended), built in FIFO queues, status information and simple power up through a few commands.

1.1 Installation

IBD189 devices support Windows operating systems.
The ASCII protocol is used to exchange data and control information with the devices and hence a serial interface is required to enable the communication.

IBD189 has only Windows driver, and requires Windows driver installation from the by-packed CD. Please read the relevant installation instructions attached for your CDC device.

In fact, the USB CDC devices (Virtual COM Port) don't usually need additional driver installation but only need to be guided to find the built-in virtual serial driver on Windows (usbser.sys) that can be used directly after you designate the INF file.

1.2 Testing the IBD189

Test the IBD189 by installing it to a PC's Virtual COM port.
When the IBD189 receives power, the LED2 (green) will turn on 2~3 seconds and turn off after module has been initialized.
Afterwards, start Windows HyperTerminal software (or your favorite terminal software) and set it up to e.g. 115200baud, 8 data bits, no parity, 1 stop bit (just ignore this because the VCP is just as an interface for serial application to IBD189 module), and set local echo on so you can see what you are typing and set the check flag so that it appends a line feed when it receives an end of line. Finally, make sure you have hardware and software handshaking off and that no LF (ASCII10) is added on outgoing CR (ASCII13).
Now, make sure you are connected and press, "ENTER" and it will make a new line, then press V and "ENTER" and it will print/reply Vhsss, where h is the hardware version and sss is the software version (e.g. V0106. Now you know you have full contact with the IBD189 module and can set it up with a CAN speed and open the CAN port, send and receive frames.

Note that you must have 2 nodes at least in CAN bus network to send/receive CAN frames and that the CAN cable network is terminated at both ends with 120 ohm over the CANL and CANH lines plus that a twisted pair CAN cable is used.
There are two CAN ports on IBD189, so that you may connect them with a CAN cable with 120 ohm terminal resistor for simple test. IBD189 is set to accept all frames by default, so there is no need to set filters etc. for testing. The IBD189 can also be tested with the demo programs "IBD189_DEMO.exe".

Example for testing IBD189:
V[CR] (should reply version, e.g. V0106[CR])
S6[CR] (set up CAN speed to 500Kbps)
O[CR] (open the CAN channel)
t1001FF[CR] (sends ID=0x100 hex with DLC=1 and data 0xFF one byte)

1.3 Application Design Guide

Always start each session (when your program starts) with sending 2-3 [CR] to empty any prior command or queued character in the IBD189 (many times after power up there could be false characters in the queue or old ones that was from a previous session), then check the CAN version with V command (to be sure that you have communication with the module at correct speed), then set up the CAN speed with S command, then open the CAN port with O, then the IBD189 is in operation for both sending and receiving CAN frames. Send frames with the t or T command and wait for a response back to see if it was placed in the CAN FIFO transmission queue or the queue was full. Incoming frames from the CAN bus will be sent out at once on the serial port or queued in the FIFO if the serial port is full. Then once in a while send the F command to see if there are any errors (e.g. each 100-500ms or if you get an error back from the IBD189).

If you get too many errors back after sending commands to the module, send 2-3 [CR] to empty the buffer, then issue the commands again. If this continues, alert the user or application within your program that there is a communication error. Try closing the virtual serial port and try again.

2.0 Available IBD189 ASCII Commands:

Note: All commands to the IBD189 must end with [CR] (ASCII=13) and they are case sensitive.

V[CR]	Get Version number of both IBD189 hardware and software This command is always available. Example: V[CR] Get Version numbers Returns: V and a 1 bytes hex value for hardware version and a 3 byte hex value for software version plus CR (ASCII 13) for OK. E.g. V0106[CR]

There are 2 CAN bus ports on IBD189, so that you have to add a prefix mark (‘) for module to distinguish which CAN port you want to control.

The commands for CAN1 and CAN2 are in similar format, just different in the prefix character of command prior with “ ‘ “ (ASCII=39).
Example:
O[CR] open the channel CAN1
‘O[CR] open the channel CAN2
Returns:
[CR] CAN1 reply OK
‘[CR] CAN2 reply OK

Sn[CR]	Setup with standard CAN bit-rates where n is 0-8. This command would active if the CAN channel is closed. S0 Setup 10Kbit S1 Setup 20Kbit S2 Setup 50Kbit S3 Setup 100Kbit S4 Setup 125Kbit S5 Setup 250Kbit S6 Setup 500Kbit S7 Setup 800Kbit S8 Setup 1Mbit Example: S6[CR] Setup CAN to 500Kbit. Returns: CR (ASCII 13) for OK or BELL (ASCII 7) for ERROR. This command would become active if the CAN channel is closed.
O[CR]	Open the CAN channel in normal operation mode (sending & receiving). This command would become active if the CAN channel is closed and has been set up prior with the S command (i.e. initiated). Example: O[CR] Open the channel. Returns: CR (ASCII 13) for OK or BELL (ASCII 7) for ERROR.

C[CR]	Close the CAN channel. This command would become active if the CAN channel is open. Example: C[CR] Close the channel Returns: CR (ASCII 13) for OK or BELL (ASCII 7) for ERROR.
F[CR]	Read Status Flags. This command would become valid if the CAN channel is open. Example: F[CR] Read Status Flags. Returns: An F with 2 bytes hex value plus CR (ASCII 13) for OK. If CAN channel isn't open it returns BELL (ASCII 7). See descriptions below. E.g. F03[CR] Bit 0 Error warning flag, see STM32F105 datasheet Bit 1 Error passive flag, see STM32F105 datasheet Bit 2 Bus-off flag, see STM32F105 datasheet Bit 3 reserved Bit 4 reserved Bit 5 reserved Bit 6 reserved Bit 7 CAN port opened

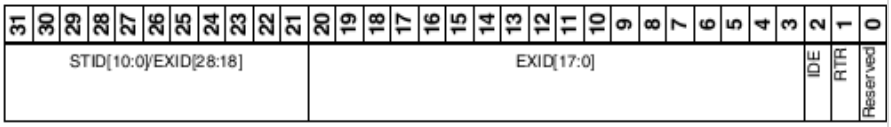
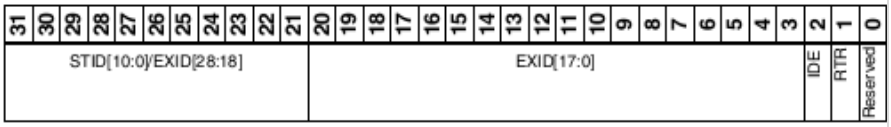
7	6	5	4	3	2	1	0
CAN port opened	Res.	Res.	Res.	Res.	BOFF	EPVF	EWGF
					r	r	r

Bit 7 CAN port opened.
Bit 6 Reserved.
Bit 5 Reserved.
Bit 4 Reserved.
Bit 3 Reserved, must be kept at reset value.

Bit 2 **BOFF**: Bus-off flag
This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255.

Bit 1 **EPVF**: Error passive flag
This bit is set by hardware when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter>127).

Bit 0 **EWGF**: Error warning flag
This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter≥96).

mxxxxxxxx[CR]	Sets CAN Message Mask Register (CAN filter mode register of STM32F105). This command would become active if the CAN channel is initiated and not opened. xxxxxxxx CAN Message Mask in hex with MSB first , in form STDID[10:0], EXTID[17:0], IDE bit, RTR bit, 0(Reserved).  For more info, see ST STM32F105 datasheet. Example: m00000000[CR] Set CAN Message Mask to 0x00000000 This is default when power on, i.e. it can receive all frames. Returns: CR (ASCII 13) for OK or BELL (ASCII 7) for ERROR.
Mxxxxxxxx[CR]	Sets CAN Message Filter Register (CAN filter registers of STM32F105). This command would become active if the CAN channel is initiated and not opened. xxxxxxxx CAN Message Filter in hex with MSB first, in form STDID[10:0], EXTID[17:0], IDE bit, RTR bit, 0(Reserved).  For more info, see ST STM32F105 datasheet. Example: M00000000[CR] Set CAN Message Filter to 0x00000000 This is default when power on, i.e. it can receive all frames. Returns: CR (ASCII 13) for OK or BELL (ASCII 7) for ERROR.

tiiildd...[CR]	Transmit a standard (11bit) CAN frame. This command would become active if IBD189 is open in normal mode. iii Identifier in hex (000-7FF) l Data length (0-8) dd Byte value in hex (00-FF). The number of the bytes must be equal to the data length field. Example 1: t100255AA[CR] Sends an 11bit CAN frame with ID=0x100, 2 bytes with the value 0x55 and 0xAA. Example 2: t0210[CR] Sends an 11bit CAN frame with ID=0x21 & 0 bytes. Returns: IBD189 replies z[CR] for OK or BELL (ASCII 7) for ERROR.
Tiiiiiiiildd...[CR]	Transmit an extended (29bit) CAN frame. This command would become active if IBD189 is open in normal mode. iiiiiiii Identifier in hex (00000000-1FFFFFFF) l Data length (0-8) dd Byte value in hex (00-FF). The number of the bytes must be equal with the data length field. Example 1: T00000100255AA[CR] Sends a 29bit CAN frame with ID=0x100, 2 bytes with the value 0x55 and 0xAA. Returns: IBD189 replies Z[CR] for OK or BELL (ASCII 7) for ERROR.

riiil[CR]	Transmit a standard RTR (11bit) CAN frame. This command would active if the IBD189 is open in normal mode. iii Identifier in hex (000-7FF) l Data length (0-8) Example 1: r1232[CR] Sends an 11bit RTR CAN frame with ID=0x123 and DLC set to two (2 bytes). Returns: IBD189 replies z[CR] for OK or BELL (ASCII 7) for ERROR.
Riiiiiiiil[CR]	Transmit an extended RTR (29bit) CAN frame. This command would become active if the IBD189 is open in normal mode. iiiiiiii Identifier in hex (00000000-1FFFFFFF) l Data length (0-8) Example 1: R000001232[CR] Sends an 11bit RTR CAN frame with ID=0x123 and DLC set to two (2 bytes). Returns: IBD189 replies z[CR] for OK or BELL (ASCII 7) for ERROR.

Setting CAN Message Filter and Mask registers

The CAN Message Filter Register and the CAN Message Mask Register works together and they can filter out a group of messages. For more information on how it works, look in the STM32F105/F107 datasheet. In 11bit ID´s it is possible to filter out a single ID this way, but in 29bit ID's it is only possible to filter out a group of ID's.

Set the bits of this one 32bits mask/filter with: 1 is must match or 0 as 'don't care'.

The example below will set a filter to only receive all 11bit ID's from 0x300 to 0x3FF.
Commands Comments
m60000000[CR] check the message Id bits with mask 0x300 & IDE=0 & RTR=0
M60000000[CR] filter message Id 0x3nn (nn = 00~FF) & IDE=0 & RTR=0

The first command tells the filter to match 2 bits and if they are not set (in this case it corresponds to 0x3nn, the 3).
The second command tells the nn to be 'don't care', so it could be from 0 to FF.

The RTR bit in the mask could also be 1 instead of 0, then we filter out the RTR bit as well and you won't accept RTR frames.
The IDE bit is similar to setting the acceptance of extended identifier.

Example:
m60000002[CR] check the message Id bits with mask 0x300 & IDE=0 & RTR=1
M60000000[CR] filter message Id 0x3nn (nn = 00~FF) & IDE=0 & RTR=0

In this case, the RTR frame won't be accepted even though the message Id is between 0x300 and 0x3FF.
On the other hand, if you want the RTR frame with message Id 0x300~0x3FF to be accepted, you should set the 32bits Filter as 0x60000002 with ASCII command M60000002[CR].