CAN2USB technical information

Preliminary version 0.0.3 2008-04-19 akhe@dofscandinavia.com

Communication with the CAN2USB adapter is through a virtual COM port with a custom driver. The protocol used is the VSCP serial protocol which is described here http://www.vscp.org/wiki/doku.php?id=vscp_specification_-_vscp_level_i_over_serial_links.

A driver for the adapter is available both for Windows and Linux that confirms to the CANAL standard interface http://www.vscp.org/wiki/doku.php?id=canal_specification.

A sliding window protocol is used to talk to the adapter for best efficiancy.

Features

- Galvanic isolation 1000 V
- Voltage supply via USB bus
- USB 2.0 full speed
- CAN bus baud rate up to 1 Mbit/s.
- Compliant with CAN specifications 2.0A (11-bit ID) and 2.0B (29-bit ID)
- Firmware upgrade via boot loader
- Drivers for Windows XP/Vista and Linux

Hardware

Pin configuration

```
1.Not connected
2.CAN-L
3.CAN-GND
4.Not connected
5.CAN-SHLD
6.CAN-GND
7.CAN-H
8.Not connected
9.Not connected
```

LED Indications

Off – No power to adapter.
Red steady – Not initialized.
Green steady – Adapter not connected to CAN bus.
Green blinking - USB initialized. Adapter connected to CAN.
Red + Green blinking - Bus warning.
Red blinking - Bus off.

When characters are received/sent the LED's will flicker.

Frame format

As the CAN2USB is a general CAN adapter it handle CAN frames instead of VSCP frames. That means that to send a VSCP Level I frame one has to pack this frame into a CAN frame.

CAN data is packed in the frames dataspace as follows

<table>
<tr><th>Byte offset</th><th>Description</th></tr>
<tr><td>0-3</td><td>CAN ID of CAN message in bigendian form.
Bit 31 of byte 0 is set for extended id, cleared for standard id.
Bit 30 of byte 0 is set for Remote frame, cleared for standard frame.</td></tr>
<tr><td>4-11</td><td>Databits.
Only the needed number of bytes used.</td></tr>
<tr><td>12-15</td><td>Timestamp on bigendian form.
Note that the offest is only valid if there is eight databytes if lower data count the time stamp start at an earlier position.
<table>
<tr><th>Offset start</th><th>Data count</th></tr>
<tr><td>4</td><td>0 Bytes.</td></tr>
<tr><td>5</td><td>1 Bytes.</td></tr>
<tr><td>6</td><td>2 Bytes.</td></tr>
<tr><td>7</td><td>3 Bytes.</td></tr>
<tr><td>8</td><td>4 Bytes.</td></tr>
<tr><td>9</td><td>5 Bytes.</td></tr>
<tr><td>10</td><td>6 Bytes.</td></tr>
<tr><td>11</td><td>7 Bytes.</td></tr>
<tr><td>12</td><td>8 Bytes.</td></tr>
</table>
</td></tr>
</table>

Available Commands

Code	Arguments	Description
1	0 – Standard. 1 - Autoconnect. 2 – Passive mode. 3 - Loopback mode.	**Open CAN channel** - In standard mode the adapter is opened with the last set bit rate. - In Auto connect mode the adapter test the different standard bit rates and connect to the bus when a frame is received with that bit rate reporting the bit rate to the host. - In passive mode the adapter just listen on CAN traffic it does not ACK any frames. - In loop back mode sent frames will be received back. The adapter is decoupled form the bus.
2	None	**Close** CAN Channel
3	Speed,reg1,reg2	**Set Bit rate** It is possible to set a stock bitrate 10Kbit, 20Kbit, 50Kbit, 100Kbit, 125Kbit, 250Kbit, 500Kbit, 800Kbit, 1000Kbit or a custom bit rate.
4	Position,Filter,mask	**Set filter/mask** Several filter/mask pairs can be set.
5	on/off	**Turn on/off error reporting.** By turning on error reporting it is possible to receive a special frame when the interface go into an error state or goes out of an error state.
6	None	**Get status.** Gets the 32-bit status value for the adapter.
7	Code for statistics to get	**Get Statistics** Get CANAL statistics from the adapter. 0 - Number of received frames. 1 - Number of transmitted frames. 2 - Number of received data bytes, 3 - Number of transmitted data bytes. 4 – Number of overruns. 5 – Number of bus warnings. 6 – Number of bus errors.
8	None	**Get serial number** Get the serial number from the adapter.
9	on/off	Enable/Disable timestamps.
10	position	**Store frame.** Store a frame in EEPROM for later transmission or auto response.
11	64 bit bit field.	Send one or several stored frames.
12	Position, auto response id	**Store auto response frame information.** Set the id that will make the adapter respond with the stored frame if function is enabled.

13	Position, enable/disable	**Enable/disable auto response frames.**
14	None	**Clear** statistics.

Error frames

Error frame code	Data	Description
1	Bit rate	Auto connect bit rate response. If bit rate 0 is returned the adapter was unable to discover any valid bit rate.
2	on/off	Error warning
3	on/off	Bus Off.
4	Status data	Status change

Updating adapter firmware

Driver interface

The driver can be interfaced directly using the VSCP serial interface. This interface is documented here http://www.vscp.org/wiki/doku.php?id=vscp_specification_-_vscp_level_i_over_serial_links or by using the usb2can.dll/usb2can.so driver.

The driver confirms to the CANAL (CAN Abstraction Layer) interface (http://www.vscp.org/wiki/doku.php?id=canal_specification). This means that software written for this adapter/driver can be used with all other adapters that have a CANAL driver

*long CanalOpen(const char *pConfigStr, unsigned long flags)*

Opens the CAN channel.

The **pConfigStr** defines configuration parameters for the interface. This string is a semicolon separated string on the form para1;param2;param3;.....

Parameters are

Position	Name	Default value	Description
1	Comm-port	1	Specifies the communication port to use. 1 is COM1, 2 is COM2 etc. 0 can be used as a special value for a simulated interface.
2	Channel	1	This is for adapters that have several CAN channels. For USB2CAN it should always be 1.
3	Baudrate	115200	Baudrate for the serial channel.
4	Bitrate	500 kbits	Bitrate for the CAN channel. This bitrate can be set to one of the standard bitrates 10 kbits 20 kbits 50 kbits 100 kbits 125 kbits 250 kbits 500 kbits 800 kbits 1 Mbit or as a couple of register value settings which allow for custom settings. The register settings consist of three bytes. Byte1 Bit 7 - Reserved, set to 0. Bit 6-4 – TSEG2 Bit 3-0 – TSEG1 Byte 2 Bit 7-6 – SJW Bit 5-0 – BRP Byte 3 BRPR register value. Defaults to 0 if not given. The register values is entered as a comma separated list of values on the form byte1:byte2: byte3.
5	Acceptance Filter	0x00000000	CAN acceptance filter.Default means all messages will be received.

6	Acceptance Mask	0x00000000	CAN acceptance mask. Default means all messages will be received.
7	Read timeout	Infinite	Read timeout for blocking read operations.
8	Write timeout	Infinite	Write timeout for blocking write operations.

.The **flags value** is also a is a configuration parameter in numerical form. Bits can be set in different combinations to enable/disable functionality.

Flag	**Value**	**Description**
CANUSB_FLAG_QUEUE_REPLACE	0x00000001	If input queue is full remove oldest message and insert new message.
CANUSB_FLAG_BLOCK	0x00000002	Block receive/transmit
CANUSB_FLAG_ERROR	0x00000004	Errors will be received from the adapter.
CANUSB_FLAG_USE_SW	0x00000008	Activate sliding windows protocol. This means slower but more secure communication.
CANUSB_FLAG_NO_LOCAL_SEND	0x00000010	Don't send transmited frames on other local channels for the same interface.
CANUSB_FLAG_SILENT	0x00000020	Just listen to CAN bus traffic
CANUSB_FLAG_LOOPBACK	0x00000040	Set loopback. Can be combined with CANUSB_FLAG_SILENT for HOT loopback

Sample configuration string

4;1;115200;500

This uses

- port COM4
- CAN Channel 1
- Baudrate 115200
- bitrate 500 kbits
- Will receive all CAN messages.
- Will block indefinitely for both the send and receive blocking routines or in both if the blocking flag is defines.

int CanalClose(long handle)

Close the channel and free all allocated resources associated with the channel.

Returns TRUE (non zero) on success or FAILURE (zero) if something goes wrong.

unsigned long CanalGetLevel(long handle)

Params

handle – Handle for open physical interface.

Returns

Returns the canal level this interface can handle. An error is indicated by a zero return value. At the moment the following levels are defined

CANAL_LEVEL_STANDARD	A starndard CANAL driver as of this specification.
CANAL_LEVEL_USES_TCPIP	This driver does not respond to any of the methods, even if they must be implemented. The driver will instead use the TCP/IP interface of the VSCP daemon for its data transfer. This driver type is called a VSCP Level II driver because it is constructed to work just with VSCP Level II events which does not fit in standard CAN frames. This type of driver does not need any buffers as no data is exchanged through the interface.

int CanalSend(long handle, const PCANALMSG pCanMsg)

Params

- handle – Handle for open physical interface.
- pCanMsg – Message to send.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

int CanalBlockingSend(long handle, const PCANALMSG pCanMsg, unsigned long timeout)

Params

- handle – Handle for open physical interface.
- pCanMsg – Message to send.
- timout - Timeout in milliseconds. 0 to wait forever.

Returns

CANAL_ERROR_SUCCESS, CANAL_ERROR_TIMEOUT on timeout on success or an error code on failure.

int CanalReceive(long handle, PCANALMSG pCanMsg)

Params

- handle – Handle for open physical interface.
- pCanMsg – Message to send.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

int CanalBlockingReceive(long handle, PCANALMSG pCanMsg, unsigned long timeout)

Params

- handle – Handle for open physical interface.
- pCanMsg – Message to send.
- timout - Timeout in milliseconds. 0 to wait forever.

Returns

CANAL_ERROR_SUCCESS on success, CANAL_ERROR_TIMEOUT on timeout or an error code on failure.

int CanalDataAvailable(long handle)

Check if there is data available in the input queue for this channel that can be fetched with CanalReceive.

Params

handle – Handle for open physical interface.

Returns

Number of frames available to read.

int CanalGetStatus(long handle, PCANALSTATUS pCanStatus)

Returns a structure that gives some information about the state of the channel. How the information is interpreted is up to the interface designer. Typical use is for extended error information.

Params

- handle – Handle for open physical interface.
- pCanStatus – Status.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

int CanalGetStatistics (long handle, PCANALSTATISTICS pCanalStatistics)

Return some statistics about the interface. If not implemented for an interface FALSE should always be returned.

Params

- handle – Handle for open physical interface.
- pCanalStatistics – Statistics for the interface.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

int CanalSetFilter (long handle, unsigned long filter)

Set the filter for a channel. There is only one filter available. The CanalOpen call can be used to set multiple filters. If not implemented FALSE should always be returned.

Enable filter settings in the open call if possible. If available in the open method this method can be left unimplemented returning false.

Params

handle – Handle for open physical interface. filter – filter for the interface.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

int CanalSetMask (long handle, unsigned long mask)

Set the mask for a channel. There is only one mask available for a channel. The CanalOpen call can be used to set multiple masks. If not implemented FALSE should always be returned.

Enable mask settings in the open call if possible. If available in the open method this method can be left unimplemented returning false.

Params

handle – Handle for open physical interface. mask – filter for the interface.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

int CanalSetBaudrate (long handle, unsigned long baudrate)

Set the bus speed for a channel. The CanalOpen call may be a better place to do this. If not implemented FALSE should always be returned.

Enable baudrate settings in the open call if possible. If available in the open method this method can be left unimplemented returning false.

Params

handle – Handle for open physical interface. baudrate – The bus speed for the interface.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

unsigned long CanalGetVersion (void)

Get the Canal version. This is the version derived from the document that has been used to implement the interface. Version is located on the front page of the document.

Returns

Canal version expressed as an unsigned long.

unsigned long CanalGetDllVersion (void)

Get the version of the interface implementation. This is the version of the code designed to implement Canal for some specific hardware.

Returns

Canal dll version expressed as an unsigned long.

*const char * CanalGetVendorString (void)*

Get a pointer to a null terminated vendor string for the maker of the interface implementation. This is a string that identifies the constructor of the interface implementation and can hold copyright and other valid information.

Returns

Pointer to a vendor string.

*const char * CanalGetDriverInfo(void)*

This call returns a documentation object in XML form of the configuration string for the driver. This string can be used to help users to enter the configuration data in an application which allows for this.

```
<?xml version = "1.0" encoding = "UTF-8" ?>

<!-- Configuration strings are given as a semicolon separated list          -->
<!-- This list is described with and item tag for each configuration option -->
<!-- Items can be of different types, string, number                        -->
<config>
   <description>Description of the driver<description>
   <level>1|2</level>
   <blocking>yes|no</blocking>
   <!-- pos is the position on the configuration line i.e.
             item0;item1;item2;item3;item4.....
   -->
   <item pos="nn" type="string"/>
   <item pos="nn" ttype="number"/>
   <item pos="nn" ttype="choice">
       <choice>first option</choice>
       <choice>second option</choice>
       <choice>.............</choice>
       <choice>last option</choice>
   </item>
</config>
```

Returns

Pointer to a configuration string or NULL if no configuration string is available.

Params

- handle – Handle for open physical interface.
- pDriverInfo – Pointer to unsigned long that holds driver information after call.
  Bit 31 - Driver support blocking calls.
  all other bits undefined at the moment.

Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.