



[VSCP, Back to the Very Simple Control Protocol](#)

CANAL Specification

The CAN Abstraction Layer
Version 1.11 (2007-11-13)
akhe@eurosource.se

<http://www.vscp.org> [<http://www.vscp.org>]

What is CANAL

CANAL (CAN Abstraction Layer) is a very simple approach to interfacing a CAN card or some other CAN hardware. It's consists mostly of open, close, read, write and filter/mask handling. If you need to do more this is not the model for you and we recommend **CanPie**

<http://sourceforge.net/projects/canpie> [<http://sourceforge.net/projects/canpie>]

or **VCA** (Virtual CAN API) used in the OCERA project above

<http://www.ocera.org/download/components/WP7/canvca-0.01.html>
[<http://www.ocera.org/download/components/WP7/canvca-0.01.html>]

which are much more advanced, and capable implementations.

CANAL has been constructed from a need to use CAN in the form of control networks in the SOHO (Small Office/Home) environment. This kind of environment is totally different to the very tough automotive or industry control world where Canal probably would not be the best solution. The goal has not been to construct something that will take over the world just something that does solve a problem at hand.

For CANAL there is some code available. First of all some drivers and also a daemon/service (VSCP Daemon) that can be used to simulate CAN networks on a PC, over Ethernet/TCP/IP etc. The daemon is available for both Windows and Linux and can be found at <http://can.sourceforge.net> [<http://can.sourceforge.net>].

CANAL is tightly coupled with the Very Simple Control Protocol, VSCP and the VSCP daemon. This is a protocol constructed for SOHO control situations. The model has been constructed as a two-layer model so that the VSCP Daemon can also be useful for people that are just interested in CAN and not in VSCP. You can find more information about vscp at <http://www.vscp.org> [<http://www.vscp.org>].

CANAL is open and free to use with no restrictions, as is the above software, which is released under GPL/LGPL.

Current information about CANAL, the VSCP Daemon and VSCP (Very Simple Control Protocol) can be found at <http://www.vscp.org> [<http://www.vscp.org>] and <http://can.sourceforge.net> [<http://can.sourceforge.net>]. There are two mailinglists available on Sourceforge

https://sourceforge.net/mail/?group_id=53560 [https://sourceforge.net/mail/?group_id=53560]

that are about CANAL (can_canal) and VSCP (can_vscp) topics.

! To subscribe to the canal list go to <http://lists.sourceforge.net/lists/listinfo/can-canal>
[<http://lists.sourceforge.net/lists/listinfo/can-canal>]

! To subscribe to the VSCP list go to <http://lists.sourceforge.net/lists/listinfo/can-vscp>
[<http://lists.sourceforge.net/lists/listinfo/can-vscp>]

CANAL-API Specification

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

Opens a CAN channel.

Params

pDevice

Physical device to connect to. This is the place to add device specific parameters and filters/masks. This is a text string. It can be a name, some parameters or whatever the interface creator chooses.

flags

Device specific flags with a meaning defined by the interface creator.

Returns

Handle for open physical interface or -1 on error. For an interface where there is only one channel the handle has no special meaning and can only be looked upon as a status return parameter.

int CanalClose(long handle)

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

Params

handle – Handle for open physical interface.

Returns

TRUE on success or FALSE if failure.

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 standard 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.
- timeout - 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.
- timeout - 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" ?>

<!-- Version 0.0.1      2007-11-13   P R E L I M I N A R Y   ! ! ! !-->

<!-- 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>

```

```

    <item type="string"/>
    <item type="number"/>
    <item type="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.

CANAL - Data Structures

CANMSG

This is the general message structure

unsigned long flags

Flags for the package.

- Bit 0 – if set indicates that an extended identifier (29-bit id) else standard identifier (11-bit) is used.
- Bit 1 – If set indicates a RTR (Remote Transfer) frame.
- Bit 2 – If set indicates that this is an error package. The id holds the error information.
- Bit 3 – Bit 30 Reserved.
- Bit 31 – This bit can be used as a direction indicator for application software. 0 is receive and 1 is transmit.

unsigned long obid

Used by the driver or higher layer protocols.

unsigned long id

The 11-bit or 29 bit message id.

unsigned char data[8]

Eight bytes of data.

unsigned char count

Number of data bytes 0-8

unsigned long timestamp

A time stamp on the message from the driver or the interface expressed in microseconds. Can be used for relative time measurements.

CANALSTATUS

unsigned long channel_status

Current state for CAN channel

CANAL_STATUS_NONE	0x00000000
CANAL_STATUS_ACTIVE	0x10000000
CANAL_STATUS_PASSIVE	0x40000000
CANAL_STATUS_BUS_OFF	0x80000000
CANAL_STATUS_BUS_WARN	0x40000000
CANAL_STATUS_PHY_FAULT	0x08000000
CANAL_STATUS_PHY_H	0x04000000
CANAL_STATUS_PHY_L	0x02000000
CANAL_STATUS_SLEEPING	0x01000000
CANAL_STATUS_STOPPED	0x00800000
CANAL_STATUS_RECEIVE_FIFO_FULL	0x00400000
CANAL_STATUS_TRANSMIT_FIFO_FULL	0x00200000

Bits from 16-31 are reserved, bits from 0-15 are user defined and can be defined by the driver maker.

CANSTAT

This is the general statistics structure

unsigned long cntReceiveFrames

Number of received frames since the channel was opened.

unsigned long cntTransmittFrames

Number of frames transmitted since the channel was opened.

unsigned long cntReceiveData

Number of bytes received since the channel was opened.

unsigned long cntTransmittData

Number of bytes transmitted since the channel was opened.

unsigned long cntOverruns

Number of overruns since the channel was opened.

unsigned long cntBusWarnings

Number of bus warnings since the channel was opened.

unsigned long cntBusOff

Number of bus off's since the channel was opened.

Error codes

CANAL_ERROR_SUCCESS	0	All is OK.
CANAL_ERROR_BAUDRATE	1	Baudrate error.
CANAL_ERROR_BUS_OFF	2	Bus off error
CANAL_ERROR_BUS_PASSIVE	3	Bus Passive error
CANAL_ERROR_BUS_WARNING	4	Bus warning error
CANAL_ERROR_CAN_ID	5	Invalid CAN ID
CANAL_ERROR_CAN_MESSAGE	6	Invalid CAN message
CANAL_ERROR_CHANNEL	7	Invalid channel
CANAL_ERROR_FIFO_EMPTY	8	Nothing available to read. FIFO is empty
CANAL_ERROR_FIFO_FULL	9	FIFO is full
CANAL_ERROR_FIFO_SIZE	10	FIFO size error
CANAL_ERROR_FIFO_WAIT	11	
CANAL_ERROR_GENERIC	12	Generic error
CANAL_ERROR_HARDWARE	13	A hardware related fault.
CANAL_ERROR_INIT_FAIL	14	Initialization failed.
CANAL_ERROR_INIT_MISSING	15	
CANAL_ERROR_INIT_READY	16	
CANAL_ERROR_NOT_SUPPORTED	17	Not supported.
CANAL_ERROR_OVERRUN	18	Overflow.
CANAL_ERROR_RCV_EMPTY	19	Receive buffer empty
CANAL_ERROR_REGISTER	20	Register value error
CANAL_ERROR_TRM_FULL	21	
CANAL_ERROR_ERRFRM_STUFF	22	Errorframe: stuff error detected
CANAL_ERROR_ERRFRM_FORM	23	Errorframe: form error detected
CANAL_ERROR_ERRFRM_ACK	24	Errorframe: acknowledge error
CANAL_ERROR_ERRFRM_BIT1	25	Errorframe: bit 1 error
CANAL_ERROR_ERRFRM_BIT0	26	Errorframe: bit 0 error
CANAL_ERROR_ERRFRM_CRC	27	Errorframe: CRC error
CANAL_ERROR_LIBRARY	28	Unable to load library
CANAL_ERROR_PROADDRESS	29	Unable get library proc address
CANAL_ERROR_ONLY_ONE_INSTANCE	30	Only one instance allowed
CANAL_ERROR_SUB_DRIVER	31	Problem with sub driver call
CANAL_ERROR_TIMEOUT	32	Blocking call timeout

CANAL_ERROR_NOT_OPEN	33	The device is not open.
CANAL_ERROR_PARAMETER	34	A parameter is invalid.
CANAL_ERROR_MEMORY	35	Memory exhausted.
CANAL_ERROR_INTERNAL	36	Some kind of internal program error
CANAL_ERROR_COMMUNICATION	37	Some kind of communication error

Codes up to 0xffff are reserved, codes from 0x10000 and up are user defined.

History

- 2007-12-08 - CanalGetDriverInfo does not need a handle.
- 2007-11-22 - CANAL_ERROR_INTERNAL and CANAL_ERROR_COMMUNICATION added.
- 2007-11-13 - getDriverConfigInfo added.
- 2007-11-12 - added const in front of send messages.
- 2007-11-01 - CANAL_LEVEL_USES_TCPIP CANAL driver Level introduced for use with the VSCP Daemon.
- 2007-10-30 - CanalGetDriverInfo call added.
- 2007-05-13 - CanalOpen device set to const char * instead of incorrect char *
- 2007-03-06 - CANALSTATUS table was wrong. Corrected.
- 2007-01-14 - Handle changed from int to long.
- 2005-07-26 - CanalBlockingOpen call removed. CanalBlockingSend and CanalBlockingReceive added. Fixed returned values that where wrong.
- 2005-03-17 - CanalBlockingOpen call added.
- 2004-07-08 - Bit 31 of the canmsg flag defined as direction bit for application software use.
- 2004-07-01 - Added some clarifications on the CanalSetMask, CanalSetFilter and CanalSetBaudrate methods
- 2004-06-07 - CANALASTATUS had a typo and was called CANALSTATE. Fixed.
- 2004-06-07 - Recovery from version lost in hard disk crash and realest as version 1.00
- 2003-02-18 - Initial version.

canal_specification.txt · Last modified: 2007/12/09 18:11 by 80.65.200.74