CANAL

The CAN Abstraction Layer

Version 1.03 (2004-07-01)

akhe@eurosource.se

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

Canal is tightly coupled with the **Very Simple Control Protocol, VSCP** and the vscpd daemon. This is a protocol constructed for SOHO control situations. The model has been constructed as a two-layer model so that canald 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.

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

Current information about **canal**, **canald** and **VSCP** (***Very Simple Control Protocol)*** can be found at **http://www.vscp.org** and **http://can.sourceforge.net**. There are two mailinglists available on Sourceforge 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

To subscribe to the VSCP list go to http://lists.sourceforge.net/lists/listinfo/can-vscp

CANAL-API Specification

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

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.

BOOL CanalClose(int 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(int handle)

Params
handle – Handle for open physical interface.

Returns
Returns the canal level(s) this interface can handle. This is a bit field and only bit 0 is defined for now. A specific interface can at the same time support several levels. An error is indicated by a zero return value.

BOOL CanalSend(int handle, PCANMSG pCanMsg)

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

Returns
TRUE on success or FALSE on failure.

BOOL CanalReceive(int handle, PCANMSG pCanMsg)

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

Returns
TRUE on success or FALSE on failure.

BOOL CanalDataAvailable(int 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
TRUE if data is available or FALSE if not.

BOOL CanalGetStatus(int handle, PCANSTATUS 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
TRUE on success or FALSE on failure.

BOOL CanalGetStatistics (int 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
TRUE on success or FALSE on failure.

BOOL CanalSetFilter (int 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
TRUE on success or FALSE on failure.

BOOL CanalSetMask (int 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.

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

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

Returns
TRUE on success or FALSE on failure.

BOOL CanalSetBaudrate (int 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
TRUE on success or FALSE 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.

*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.

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. The following error codes are defined at the moment

• **CANAL_ERROR_NONE**	0
• **CANAL_ERROR_BAUDRATE**	1
• **CANAL_ERROR_BUS_OFF**	2
• **CANAL_ERROR_BUS_PASSIVE**	3
• **CANAL_ERROR_BUS_WARNING**	4
• **CANAL_ERROR_CAN_ID**	5
• **CANAL_ERROR_CAN_MESSAGE**	6
• **CANAL_ERROR_CHANNEL**	7
• **CANAL_ERROR_FIFO_EMPTY**	8
• **CANAL_ERROR_FIFO_FULL**	9
• **CANAL_ERROR_FIFO_SIZE**	10
• **CANAL_ERROR_FIFO_WAIT**	11
• **CANAL_ERROR_GENERIC**	12
• **CANAL_ERROR_HARDWARE**	13
• **CANAL_ERROR_INIT_FAIL**	14
• **CANAL_ERROR_INIT_MISSING**	15
• **CANAL_ERROR_INIT_READY**	16
• **CANAL_ERROR_NOT_SUPPORTED**	17
• **CANAL_ERROR_OVERRUN**	18
• **CANAL_ERROR_RCV_EMPTY**	19
• **CANAL_ERROR_REGISTER**	20
• **CANAL_ERROR_TRM_FULL**	21

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

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_STATE_UNKNOWN**	0
• **CANAL_STATE_ACTIVE**	1
• **CANAL_STATE_BUS_OFF**	2
• **CANAL_STATE_BUS_WARN**	3
• **CANAL_STATE_PHY_FAULT**	4
• **CANAL_STATE_PHY_H**	5
• **CANAL_STATE_PHY_L**	6
• **CANAL_STATE_SLEEPING**	7
• **CANAL_STATE_STOPPED**	8

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

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.

History

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.