Embedded Intelligence CAN

Software interface

Overview:

Embedded Intelligence manufactures a variety of different interfaces used to connect PC hardware to the Control Area Network (CAN) bus. This package provides documentation of the software interface to that hardware and examples for a number of different programming languages and development environments.

The CAN interface hardware is accessed through the use of a device driver running on the host operating system. It's our intention to support a wide variety of different operating systems. Presently there are drivers available for: Windows (XP and later), Linux, and QNX real time OS. Other drivers are likely to be available in the future.

This document describes the interface between the host software and the device driver. We have taken pains to keep this interface as simple as possible and to keep it consistent from one OS to another. In particular, the use of DLLs (Dynamic Link Libraries) have been avoided in favor of a more direct connection to the device driver. This simplifies software installation as it reduces the number of additional files that need to be installed.

Common structures:

Several data structures are common to both the C and C++ language interfaces.

`CanFrame`

This structure is used to represent a message transmitted on the CAN bus. The structure has the following fields.

`type` This field gives the type of message and can take one of the following values defined in the API header file:

- CAN_FRAME_DATA: A normal data message
- CAN_FRAME_REMOTE: A remote request type CAN message.
- CAN_FRAME_ERROR: An error frame containing information about an error that occurred on the bus.

`length` This field gives the number of bytes of data included in the message. Legal values range from 0 to 8.

`id` This field gives the CAN message ID.

The CAN bus supports two different styles of CAN message; standard messages with an 11-bit ID and extended messages with a 29-bit ID. Bit 29 of this field is used to identify the type of message

For standard CAN messages, bit 0-10 of this field contain the message ID and bits 11-31 should be clear.

For extended messages, bits 0-28 of this field contain the ID and bit 29 must be set to identify the message as an extended message. Bits 30-31 are not used and should be clear.

`timestamp` On received messages, this field gives the time of reception. The time is in units of microseconds.

`data` This array of 8 bytes holds the CAN message data. CAN bus messages hold anywhere from zero to eight bytes of data.

CanErrorInfo

This structure is used to represent an error frame received on the CAN bus. The layout of this structure is identical to a CanFrame structure, but the meaning of the fields is slightly different.

`type`	For error messages, this type will always be CAN_FRAME_ERROR
`length`	This field gives the number of bytes of data included in the message.
`mask`	This field holds a bit-mask indicating which errors were detected on the bus. See the API header file for a list of error bit meanings.
`timestamp`	This field gives the time of the error is in units of microseconds.
`data`	This array holds additional information about the error.

Byte	Meaning
0	Transmit error counter
1	Receive error counter
2-7	Reserved for future use

CanCardInfo

This structure holds some information about the CAN card such as firmware version number.

`serial`	Card serial number
`hwType`	Integer identifying the type of CAN card
`fwVer`	Firmware version number.
`bootVer`	Boot loader version number
`fpgaVer`	FPGA image version number
`driverVer`	Driver version number

C Language Interface:

The C subdirectory in this package holds a very simple C language interface to the CAN card. This interface does not require a dll, rather it directly interfaces with the device driver. Simply add files eican.c and eican.h to your C language project file or Makefile. These source files are designed to work in Windows as well as Linux development environments.

The following functions comprise the C language interface to the CAN driver:

`void *EICanOpen( int port, int baud, int *error );`

This function opens an interface to the CAN port. It should be the first function called when interfacing to the CAN driver.

Parameters:

port	This parameter is used to identify which CAN interface to access on systems with multiple interfaces. Possible values for this parameter range from 0 to *N*-1 for a system with *N* CAN ports installed.
baud	This parameter specifies the bit rate at which CAN communications will take place. It should be one of the values defined in the eican.h header file. For example, to communicate at 500k bits/second, pass the value EICAN_BITRATE_500000.
error	This pointer can optionally be used to retrieve an error code in the event that the function fails. If provided, the error code will be returned in the memory location referenced by this value. It's legal to pass NULL here in which case no additional error information will be returned.

Return:

On success, this function returns a pointer which should be passed to future function calls in the CAN interface library. On failure, a NULL pointer is returned and an error code is copied to the passed error parameter (if provided).

`int EICanClose( void *local );`

When finished using the CAN interface, this function should be called to close the interface.

Parameters:

Local	The pointer value returned from EICanOpen should be passed here.

Return:

An error code is returned on failure, or 0 on success.

`int EICanRecv( void *local, CanFrame *frame, int32_t timeout );`

Receive the next CAN frame from the network.

Parameters:

local	The pointer value returned from EICanOpen should be passed here.
frame	A pointer to a CanFrame structure where the frame will be returned.
timeout	A timeout, in milliseconds. If no message is immediately available when this function is called, the function will block for up to this amount of time waiting for a message. If a zero timeout is passed, the function will return immediately will an error code if no message is available. A negative timeout is treated as infinity.

Return:

An error code is returned on failure, or 0 on success.

`int EICanXmit( void *local, CanFrame *frame, int32_t timeout )`

Transmit a CAN frame on the network.

Parameters:

local	The pointer value returned from EICanOpen should be passed here.
frame	A pointer to a CanFrame structure holding the frame to be transmitted.
timeout	A timeout, in milliseconds. This timeout gives the maximum amount of time to spend trying to add the message to the transmit queue. A successful return from this function only indicates that the frame was added to the transmit queue, not that it's been transmitted on the bus. A zero timeout prevents any blocking in the event that the queue is full when the function is called. A negative timeout is treated as infinite.

Return:

An error code is returned on failure, or 0 on success.

`int EICanReadCardInfo( void *local, CanCardInfo *info )`

Read some information about the CAN card and return it in the pass structure.

Parameters:

local	The pointer value returned from EICanOpen should be passed here.
info	A pointer to a CanCardInfo structure where the card information will be returned.

Return:

An error code is returned on failure, or 0 on success.

C++ Language Interface:

The CPP subdirectory in this package holds a simple C++ language interface to the CAN card. This interface does not require a dll, rather it directly interfaces with the device driver. Simply add files eican.cpp and eican.h to your C language project file or Makefile. These source files are designed to work in Windows as well as Linux development environments.

The API defines a class called EIcan which represents the CAN interface. This class has the following public member functions.

`int EIcan::Open( int port, int baud );`

This member function opens an interface to the CAN port. It should be the first function called when interfacing to the CAN driver.

Parameters:

port	This parameter is used to identify which CAN interface to access on systems with multiple interfaces. Possible values for this parameter range from 0 to *N*-1 for a system with *N* CAN ports installed.
baud	This parameter specifies the bit rate at which CAN communications will take place. It should be one of the values defined in the eican.h header file. For example, to communicate at 500k bits/second, pass the value EICAN_BITRATE_500000.

Return:

An error code will be returned on failure. On success, this function will return zero.

`int Eican::Close( void );`

When finished using the CAN interface, this function should be called to close the interface.

Return:

An error code is returned on failure, or 0 on success.

`int Eican::Recv( CanFrame *frame, int32_t timeout );`

Receive the next CAN frame from the network.

Parameters:

frame	A pointer to a CanFrame structure where the frame will be returned.
timeout	A timeout, in milliseconds. If no message is immediately available when this function is called, the function will block for up to this amount of time waiting for a message. If a zero timeout is passed, the function will return immediately will an error code if no message is available. A negative timeout is treated as infinity.

Return:

An error code is returned on failure, or 0 on success.

`int Eican::Xmit( CanFrame *frame, int32_t timeout )`

Transmit a CAN frame on the network.

Parameters:

frame	A pointer to a CanFrame structure holding the frame to be transmitted.
timeout	A timeout, in milliseconds. This timeout gives the maximum amount of time to spend trying to add the message to the transmit queue. A successful return from this function only indicates that the frame was added to the transmit queue, not that it's been transmitted on the bus. A zero timeout prevents any blocking in the event that the queue is full when the function is called. A negative timeout is treated as infinite.

Return:

An error code is returned on failure, or 0 on success.

`int Eican::ReadCardInfo( CanCardInfo *info )`

Read some information about the CAN card and return it in the pass structure.

Parameters:

info	A pointer to a CanCardInfo structure where the card information will be returned.

Return:

An error code is returned on failure, or 0 on success.